UNIVERSITY
of
GLASGOW

Department of Computing Science

SUIT – A Methodology and Framework for *S*election of *U*ser *I*nterface Development *T*ools

Joanna Marie Lumsden

A Thesis submitted for the degree of Doctor of Philosophy
to
The University of Glasgow

February 2001

ABSTRACT

This thesis investigates the issues involved in supporting user interface development tool (UIDT) selection according to suitability within a given context of use. This thesis argues that current UIDT evaluation facilities do not adequately support context-sensitive UIDT selection. As a result, selected UIDTs reflect evaluator bias and often do not function well in their environment of use due to lack of acceptance or suitability. Without an explicit model of project-specific context of use, evaluators acting on behalf of a software development team or organisation cannot reasonably be expected to focus systematically on and consider the criteria necessary to assess suitability when rating UIDTs for selection.

This thesis describes the findings of an industrial survey that identified the context of use for software development projects. This context of use is parameterised and combined with a categorisation of UIDT functionality to produce an extensible and tailorable reference model or framework for UIDT evaluation and selection. An accompanying methodology - which together with the framework is known as SUIT (*S*election of *U*ser *I*nterface Development *T*ools) - guides the use of the framework such that project-specific context of use can be modelled and thereafter systematically considered during UIDT selection. This thesis proposes that such focussed and documented consideration of context of use during UIDT selection increases the quality of a selection decision and therefore facilitates reuse of UIDT evaluation and selection results.

An evaluative study is described which demonstrates the effectiveness and viability of the SUIT framework and methodology as a paper-based UIDT evaluation facility. The same study also identifies the need for a computer-based tool to support the management of UIDT evaluation data and to assist its comparison and analysis. Experiences with this study, the results of the industrial study, and the structure of the framework and methodology provided input into a set of requirements for a computer-based visualisation environment that supports the comparison and analysis of UIDT data.

The SUIT data visualisation environment and its qualitative evaluation are described. The evaluation results identify the usefulness and practicability of the SUIT approach when supported by the visualisation environment. They also suggest a number of refinements and extensions to the tool. The results provide an initial corpus of knowledge regarding practical strategies used by evaluators to compare and analyse UIDT evaluation data. These strategies are modelled using a novel purpose-built graphical notation that focuses on sequencing, flexibility, and patterns of activity.

Lastly, the evaluation studies of SUIT reveal its potential for use in the evaluation of other software applications, not least CASE tools. Furthermore, they highlight the potential for expansion of SUIT into an open-source UIDT evaluation and selection facility.

ACKNOWLEDGEMENTS

Without Phil Gray's supervision and encouragement this thesis would not have been possible. I would also like to thank my second supervisor, Ray Welland, for his valuable ideas, practical advice, realism, and support. Together, they gave me the confidence to believe in my work and helped me see the wood for the trees. Thanks guys.

Chris Johnson was a solid source of inspiration and advice during my first and second year vivas, asking some essential questions and pointing out several issues that were subsequently worked on in the thesis. I have also to thank other members of staff in the department to whom I looked for advice throughout the duration of this research.

I have to thank Roy Tait and Francis Hillen of Buchanan International Ltd. who provided essential information without which much of my evaluative work would not have been possible. At the sharp end of software development, their encouragement and enthusiasm for my work heightened its sense of purpose and supplied increased incentive to provide a practical solution for user interface development tool selection.

I would like to thank the people who gave willingly of their time and effort to assist in my evaluative studies. In particular, I would like to thank Helen Purchase for her advice, enthusiasm, and assistance.

Thanks to all my family and friends for putting up with me and for being interested - it can't have been easy. I would especially like to thank my parents for their continued love and support - both financial and emotional - without which I would probably not have made it.

Finally, thanks to Keith who has supported me mentally, physically, and financially, and has done a wonderful job. I promise the proof-reading is over - I hope that you think it has all been worth it !

CONTENTS

TABLE OF FIGURES

LIST OF TABLES

CHAPTER 1 : INTRODUCTION

1.1 Introduction

Developers of interactive software are confronted by a variety of software tools to assist in the process of designing and implementing the interactive aspects of applications. Not only do these tools fall into different categories in terms of functionality, but within each category there is a growing number of competing tools with similar, but not identical, features. Choice of user interface development tool (hereafter referred to as UIDT) is therefore becoming increasingly complex.

1.2 Problem Solving and Decision Making in UIDT Selection

To select a UIDT is to solve a problem by deciding on one from among many possible options. To better understand the process of UIDT selection, it is therefore important to identify and recognise relevant issues of problem solving and decision making (Harris, 1998a, Harris, 1998b, Simon, 1986, Loch and Conger, 1996, Jacobs and Holten, 1995, Sauter, 1999, Rumble, 1991, Albers, 1996).

Sauter identifies four styles of problem solving/decision making: (1) left-brain style; (2) right-brain style; (3) accommodating; and (4) integrated (Sauter, 1999). The *left-brain style* stresses analytical and quantitative techniques and employs rational and logical methods or reasoning; problems are broken down and each sub-problem is approached using data and logic. Problems are explicitly defined, solution methodologies are determined, orderly information searches are conducted, and analysis is increasingly refined in an effort to achieve predictability and minimise uncertainty. This style of decision making works best when all relevant variables can be controlled or predicted, measured, quantified, and when complete information is available. Vastly different from left-brain style decision making, the *right-brain style* is based on intuitive techniques and typically places more importance on feelings than facts. *Accommodating* decision makers are willing to use their non-dominant style when they realise that it will work best in a given situation. Finally, *integrated style* decision makers are able to effectively combine the left- and right-brain styles; analytical processes are adopted to filter information and intuition is used to contend with uncertainty and complexity. If intuition leads the decision process in a different direction, integrated style decision makers reason and analyse again to verify and elaborate on their thought process.

There is evidence to suggest that industrial software developers currently rely heavily upon right-brain style (or intuitive) decision making when selecting UIDTs with the result that they often select UIDTs that are inappropriate for their development projects (see chapter 3). This research acknowledges the existence and place for intuition or non-rational decision making in the overall process of UIDT evaluation and selection. However, for reasons outlined in the remainder of this chapter, and thereafter this dissertation, this work aims to facilitate *integrated style* UIDT selection - that is, to provide evaluators with a mechanism which allows them to approach UIDT selection from an analytical perspective but which additionally does not suppress intuitive decision making so that, where necessary, it can be used to handle areas of uncertainty such as trade-offs. By promoting an analytical approach to UIDT selection it is possible to guard against inappropriate use of intuition (Sauter, 1999).

The first step when solving a problem is to identify assumptions that establish selection boundaries - they provide a framework that limits and simplifies the problem (by reducing the number of items to consider, assumptions make a problem more manageable to solve) and reflect values that should be maintained in the solution (Harris, 1998b). The setting in which UIDT selection takes place is shaped by a complex set of variably negotiable, co-operating, and/or competing goals and constraints. The UIDT selection decision space is defined by a set of high-level factors such as the budget allocation assigned to UIDT purchase and installation, the availability of staff training for the deployed UIDT, programming language restrictions, time constraints (primarily on the project duration but subsequently on the time available for UIDT selection), development platform restrictions, availability of information, cultural binding (whether the selection is limited because of attitudes in the culture or practice of the organisation), cost/benefit ratio, and the amount of energy available to complete the selection. A similar set of factors establishes the boundaries for other aspects of project-specific decision making (for example, decisions regarding hardware or other software selection).

Once delineated, a problem must be represented in a way that facilitates its solution (Simon, 1986, Jacobs and Holten, 1995, Albers, 1996). The representation of a problem influences the quality of the solution found - major advances in human knowledge are frequently the result of new ways of thinking about problems (Simon, 1986). Models are used to present a problem in a way that allows someone to understand and solve it - by seeing a problem from a different perspective it is often easier to gain the insight necessary to find a solution (Harris, 1998b, Sauter, 1999). Models can be structural or conceptual visual, physical, mathematical, or metaphorical representations of a problem (Harris, 1998b).

The delimited decision space for UIDT selection includes criteria that are determined by the design and development requirements of a specific project, and by some of the delimiting constraints. In particular, UIDT selection criteria are informed by the functional requirements demanded of the selected UIDT - for example, the user interface-specific components required for the end product user interface and the necessary software engineering administrative support - and are influenced by the remaining limiting assumptions (see section 1.2.1). A *decision matrix* (mathematical model) enables a problem solver to 'quantify subjectivity'[1] and to be sure that all criteria are taken into account to the desired degree. A *recipe model* emphasises ingredients and proportions - giving minor consideration to items that add 'spice' or 'flavour' - and is often considered to be a list or formula for success (Harris, 1998b). Despite their suitability, use of these models is limited in existing UIDT selection techniques (see chapter 2), most of which make no attempt to model the problem in these ways.

Once modelled, a problem is solved by deciding between different solutions. Making a decision implies that there are a number of choices to be considered - the increasing number of UIDTs makes for a large range of options. When deciding on one of the many UIDTs, the principal aim should be to choose the one that best fits with identified goals and values (Harris, 1998a, Jacobs and Holten, 1995, Albers, 1996).

UIDT selection involves two essential decision types: (1) decisions *whether*; and (2) decisions *which*. Although the former - that is, whether a UIDT should be purchased/used - would typically be decided prior to choosing a UIDT, the selection process itself may be required to confirm or clarify the decision. Assisting the latter - that is, the selection of one UIDT from the range of options - is the primary focus of established UIDT evaluation techniques and is the focus of this thesis.

Every decision is made within a decision environment - the collection of information, options, values, and preferences available at the time of the decision. Decision making is the process of sufficiently reducing (it would not be feasible to eliminate) uncertainty and doubt about the options to allow a reasonable choice to be made from among them. This stresses the importance of the information-gathering function of decision making (Harris, 1998a, Sauter, 1999) and of identifying different options.

Some of the factors limiting the UIDT decision space (in particular, budget allocation, programming language, and development platform) determine the set of UIDTs that are considered during the selection process - UIDTs can be assessed against these criteria, and their suitability determined, without having to evaluate the low-level detail of the tools.

---

[1] (Harris, 1998b).

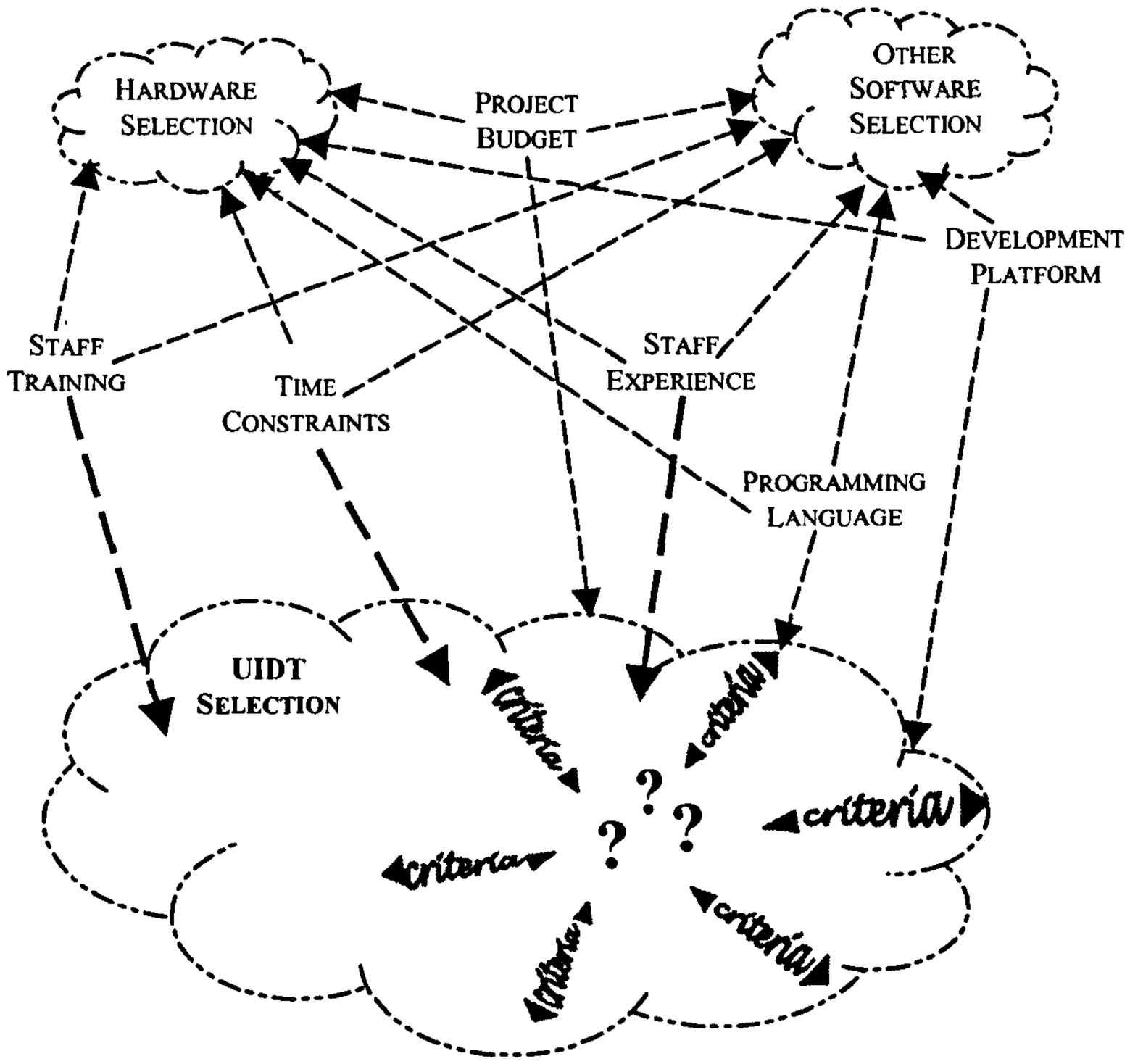


Figure 1.1 - *A model of the matrix of UIDT evaluation*

Consider now, the function of information-gathering. Decision makers tend to seek more information than required to make a good decision (Harris, 1998a). This often leads to: (1) delay in the decision because of the time required to collect and process the extra information - the effectiveness of the decision is ultimately impaired; (2) information overload which in turn leads to decline in decision making ability; (3) selective use of information to support preconceived solutions; (4) mental fatigue which returns slower and poorer quality work; and (5) decision fatigue which results in fast, careless decisions or even decision paralysis (Harris, 1998a). UIDTs are assessed or rated according to the degree to which they meet identified criteria, and it is these criteria that determine the information that needs to be collected for each candidate UIDT. Figure 1.1 illustrates the decision matrix in which UIDT selection occurs - showing examples of limiting constraints and suggesting some typical decision spaces that may occupy the same matrix. The UIDT selection decision space illustrates the internal criteria against which UIDTs are measured.

There are several documented strategies for decision making - for example, optimising, satisficing, maximax, and maximin (Harris, 1998a, Simon, 1986). Although uncommon, the latter two may be used occasionally for UIDT selection. Optimising is a strategy of identifying as many different options as possible and choosing the best. How thoroughly this can be performed depends on: (1) the importance of the problem; (2) the time available for solving it; (3) the availability of resources and knowledge; and (4) the value or desirability of each outcome. Software development deadlines, together with existing

UIDT selection techniques, make optimised UIDT selection impracticable and therefore rarely performed (see chapter 3). Instead, satisficing – whereby the first satisfactory option is chosen rather than the best option – although not perhaps ideal, is used in reality (see chapter 3). When satisficing, lower level goals are substituted for maximised goals (Simon, 1986) in a process of goal adjustments and trade-offs. Where no UIDTs are found to completely meet identified criteria, a complex process of goal, constraint, and criteria adjustment and trade-off is initiated. Criteria within the UIDT selection decision space may be traded-off against each other to facilitate a selection. Where this is insufficient, the limiting boundaries of the decision space may also have to be adjusted; these activities may ultimately affect changes to the boundaries limiting the decision spaces occupying the same matrix (see Figure 1.1). Decision making for UIDT selection therefore has not only to deal with the complexity of its own criteria, but it also has to consider the decision stream (Harris, 1998a) of which it is part.

The quality of a UIDT selection decision is a rating of whether the decision is good or bad. A good decision is logical - based on the available information - and reflects context-sensitive values set for the problem solution. A bad decision is based on inadequate information and does not reflect the identified values (Harris, 1998a). The quality of a decision is not reflected in its outcome - a good decision can have either a good or bad outcome; a bad decision can still have a good outcome. Decision quality is judged according to the following criteria: (1) the decision must meet the stated problem objectives as thoroughly and completely as possible; (2) the decision must meet the objectives efficiently with concern for cost, energy, and side effects; and (3) the decision must take into account valuable by-products or indirect advantages (Harris, 1998a). In particular,

> *'it is better to expend a little more energy to solve a problem well the first time than have to redo the entire thing after a half energetic solution'* (Harris, 1998a)

1.2.1 The Importance of Context

The importance of context is recognised in the design of software. Clarke comments that:

> *'designers often neglect to take account of contextual factors due to their focus on the artefact itself.'* (Clarke, 1997, pp. 10)

Disastrous consequences can arise when important contextual factors are ignored during software design and development - for example, with hindsight it was discovered that lack of consideration of the context in which the Patriot system (for Scud missile interception) would be used, contributed to loss of life in the Gulf War (Clarke, 1997).

That said, Cockton *et al* (Cockton et al., 1996) describe situations in which designers collected but failed to use contextual information systematically - rather than use the contextual information to generate context-specific requirements it was simply unused and general, non-context-specific requirements were generated instead. Cockton *et al* suggest that the problem with using contextual information is in using it fully and effectively due to the lack of effective representations of this information:

> *'The problem can be reduced simply to understanding the relationships between human contexts and systems designs. To understand these relationships, we require descriptions of human contexts and systems designs and ways of linking between these descriptions'* (Cockton et al., 1996, pp. 523).

Clarke further suggests that making the relationships between context and design explicit allows accurate judgements to be made about the use of contextual information in design (Clarke, 1997).

Considering context of use during UIDT selection is similarly important. In this case, it is evaluators who fail to consider adequately contextual information when selecting a UIDT due to their focus on the tools themselves. As a result, often inappropriate UIDT selection is made (Kemerer, 1992, McKirdy, 1998).

Often the choice that would perhaps be obvious to an evaluator may not function in the context in which it is to be used due to cost, time, and most importantly, lack of acceptance (Harris, 1998a). Problem solving and decision making changes when an individual evaluator is asked to assume an organisational position to select a UIDT not for himself, but for the members of a group (see chapter 7). In these circumstances, evaluators are required to adapt their goals and values to their responsibility (Simon, 1986) - namely, the context of use of the selected UIDT. Without an adequate model of the UIDT's context of use, evaluators are prone to reverting to their individual set of preferences and goals (see chapter 7). It is therefore important to identify and adequately model the context of use for the selected UIDT so that it can be considered when rating candidate UIDTs against selection criteria during the decision making process. Additionally, by identifying the context-sensitive criteria to be considered during the decision making process it may be possible to focus information-gathering for UIDT selection and thereby potentially prevent the hazards of excess information discussed in section 1.2.

One of the most important considerations in decision making - and therefore in UIDT selection - is the people factor. A selection decision must always be made in light of the people who will be required to use the UIDT (Harris, 1998a) - those who must use the selected UIDT must accept it if it is to be used effectively and efficiently. Acceptance is critically important in problem solving - a UIDT that is technically brilliant or that only

reflects the preferences of the evaluator may be 'sociologically stupid'[2] with respect to the anticipated UIDT users, and would not therefore represent a good decision (Simon, 1986). To increase acceptance of a selected UIDT within a specific context of use, the project team members who will have to use the selected tool should be considered when making the selection decision. Acceptance is further increased if the drawbacks of the selected UIDT are outlined in addition to the projected benefits - users are more likely to accept a decision if they understand the risks and believe that they have been given due consideration (Harris, 1998a, Rumble, 1991). A good quality UIDT selection decision, based on identified criteria and context of use, should be adequately substantiated to make the presentation of these facts possible (Sauter, 1999).

As with context in design, it is not simply enough to consider the context of use of UIDTs during their evaluation and selection. There needs to be a mechanism by which to record this context during the selection process, and to explicitly represent its influence over the suitability of any given UIDT.

Chapter 2 reviews the currently available UIDT evaluation facilities and demonstrates the absence of a means to represent and therefore adequately consider the relationship between context of use and UIDT functionality within each. Whether this causes or simply exacerbates the lack of consideration of context of use during UIDT evaluation is unclear. What is clear, however, is that despite its influence and importance none of the available UIDT evaluation facilities either promote or support the consideration of context of use during UIDT evaluation and selection. They do not, therefore, facilitate good quality decisions according to the definition given in 1.2.

1.3 Problem Statement

The matrix modelled in Figure 1.1 is an illustrative model of UIDT selection. Together with the discussion in section 1.2.1, it demonstrates the importance or influence of context during UIDT selection. This thesis argues that none of the UIDT evaluation facilities currently available (see chapter 2) adequately support or encourage consideration of context. While they all contribute to de-contextualised UIDT evaluation, they all fail to fully consider the context in which a UIDT is to be used and thus do not adequately support the complexities of the decision making process described in section 1.2.

It would therefore appear that the main problem facing UIDT evaluators is that they are either unaware of the importance of considering context of use during UIDT evaluation or

[2] (Simon, 1986).

they find it impossible to adequately incorporate contextual information in the evaluation process using current UIDT evaluation facilities. Since the influence of contextual information is not recorded in existing evaluation techniques, evaluators cannot easily determine the suitability of a UIDT specifically for their given project.

1.4 Thesis Statement

The thesis of this dissertation is:

> *User interface development tool selection according to suitability in a given context can be supported by an explicit methodology[3] and associated tools that are both useful and practicable.*

As such, the research represented in this dissertation has one main aim:

> *To investigate the means by which to guide and support consideration of context of use during evaluation of user interface development tools such that these tools are selected according to their suitability within a given context.*

The result of this investigation will be the development of a facility that promotes and supports contextually-sensitive evaluation and selection of user interface development tools. In order to achieve this, the context in which user interface development tools are used will need to be investigated. Hence, this thesis also aims to investigate the issues that comprise the context of use for user interface development tools, particularly within industrial software development.

Given the scope and complexity of the setting in which UIDT evaluation takes place, it is necessary to restrict the scope of the above support for the purpose of this thesis. On the basis of the selection model illustrated in section 1.2, this thesis will focus on the means by which to represent the context of use for UIDT selection in isolation - it will not support consideration of the means by which other decision spaces (that is, aspects of the associated decision stream) influence that context. In particular, it will: (1) consider the means by which to establish and represent the criteria for UIDT selection based on functional requirements and context of use (see section 1.2); and (2) will investigate a facility whereby the conformance of different UIDTs to these criteria can be compared and analysed to inform the trade-offs and context alteration discussed in section 1.2. It should be noted that it is not the intention of this thesis to take the decision out of the hands of

[3] 'Methodology' is used throughout this thesis to refer to a systematised process for carrying out some process.

evaluators. Instead, it aims to present UIDT data in such a manner as to make the need for, or instances requiring, contextually-sensitive trade-offs apparent.

1.5 Methodological Approach

To generate an explicit methodology and associated tools which usefully and practicably support context-sensitive selection of user interface development tools, an integrated series of investigations must be undertaken. This section briefly identifies the methodological approach to investigation adopted by this research, as illustrated in Figure 1.2 - in which arrows show the logical flow of information between the various activities. The detail of each investigation is outlined in the corresponding chapter hereafter.

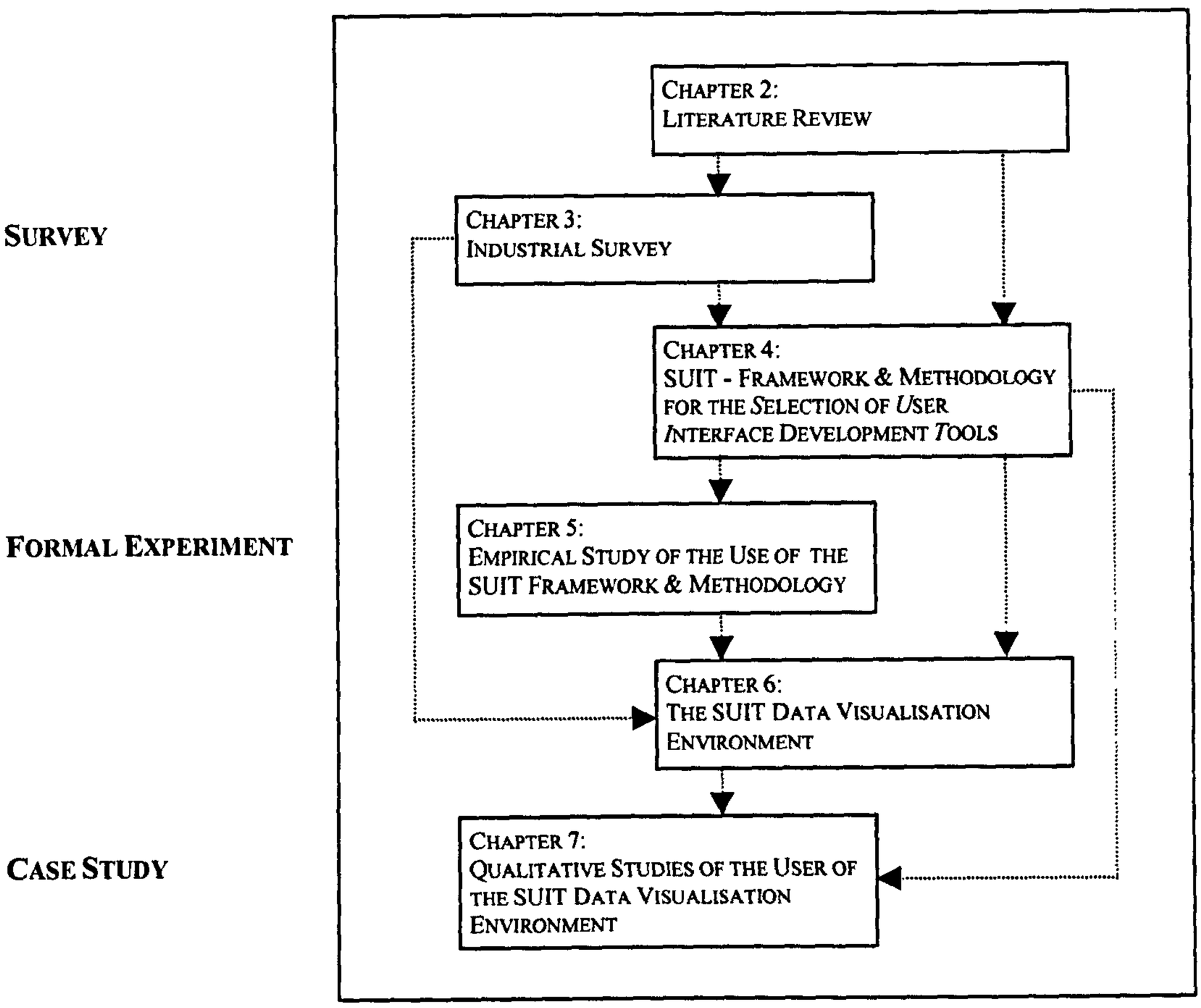


Figure 1.2 - *Methodological approach to investigation*

To develop an evaluation methodology that supports selection of UIDTs according to suitability within a given context, it is first necessary to determine what constitutes context-sensitivity and suitability within industrial software development projects. That is, it is necessary to investigate the context in which user interface development and therefore UIDT use takes place and from that to determine the aspects of user interfaces, user interface development, and UIDT use that influence UIDT suitability. To ensure the

general applicability of the results and thereby justify their use as the basis for further development of the proposed selection methodology, it is important to collect and analyse data from a wide variety of projects. This therefore necessitates the use of a survey (Kitchenham et al., 1994). It could be argued that ethnographic studies represent an alternative means by which to approach this investigation. Ethnographic studies of individual software development projects typically collect the richest data and highlight issues that questionnaires and interviews (the accepted instruments for conducting surveys) may often miss. However, ethnographic studies generally require significant dedication of time to the observation of each individual project and thus, given the time restrictions and requirement for data from a wide range of projects, it is not considered a feasible alternative for use in this investigation; a survey is therefore used instead. Chapter 3 outlines in detail the structure and results of the survey.

As demonstrated in Figure 1.2, the results of the survey (together with the findings of a review of existing UIDT evaluation mechanisms) are fed into the development of an explicit methodology for UIDT selection according to context-sensitive suitability. Having developed the methodology, it is necessary to test the hypothesis that the methodology is effective in terms of directing evaluator attention to issues of context of use during project-specific UIDT selection. According to Kitchenham *et al* a formal experiment is a means of testing, using the principles of experimental design, whether a hypothesis (that is, a statement of the expected benefit of using a method/tool compared with another method/tool) can be confirmed (Kitchenham et al., 1994). In order to prove the identified hypothesis, it is therefore necessary to employ formal experimental techniques using an alternative UIDT evaluation mechanism as a control. Following the guidelines suggested by Kitchenham *et al* the experiment is designed to minimise, within the limitations imposed upon the investigation, the effects of extraneous factors by ensuring representativeness, randomisation, and replication in the experimental design (Kitchenham et al., 1994). In accordance with the recommendation of Kitchenham *et al* (Kitchenham et al., 1994) and that of an expert statistician (Tweedie, 2000), the results of the experiment avoid inappropriate use of statistical analysis techniques that assume normal distribution. The study and its results are presented in detail in chapter 5.

On the basis of the findings of both the survey and formal experiment, a data visualisation environment is developed that embodies the paper-based evaluation methodology and allows evaluators to better visualise and then analyse UIDT evaluation data. Given the unique nature of the visualisation environment, it is important to investigate its usefulness and practicability whilst additionally identifying usability defects in its design. Since a formal hypothesis is inappropriate at this time, and it is not the intention of this investigation to conduct statistical analysis of quantitative data, a formal experiment

(according the Kitchenham *et al* definition above) is inappropriate and unnecessary. Instead, a series of case studies, whereby the visualisation environment is used on a full scale representative UIDT data comparison and analysis task, is performed. These observational studies return qualitative data that may be used to further enhance and inform future development of the SUIT data visualisation environment. Chapter 7 describes the design of the case studies and discusses the resultant observations.

1.6 Summary of Dissertation

To achieve the aim of this thesis, chapter 2 reviews the current literature regarding available facilities for user interface development tool evaluation and selection. It also considers selected CASE tool evaluation to draw comparisons. The review highlights the notion that context of use is essential when evaluating and selecting user interface development tools and establishes that it is not adequately supported in current facilities. Chapter 3 investigates relevant issues concerning industrial user interface development. These issues - investigated according to application domain - include: the characteristics of applications; the user interface development strategies followed in practice; the processes adopted to select user interface development tools; the type of user interface development tools chosen; and the extent to which these tools are used. The results from this survey motivate and inform the basis of the design and development of the context-sensitive evaluation facility for user interface development tools described in chapter 4. Chapter 5 outlines the structure, and describes the results, of an empirical study that demonstrates the viability of the paper-based evaluation facility described in chapter 4, and validates its effectiveness in terms of directing evaluator attention to issues of context of use. The chapter describes this validation process, and identifies a requirement for software support to assist evaluators in the use of the UIDT selection facility. In chapter 6, the design and development of this software support is discussed, and the resultant tool is described. Chapter 7 presents an evaluative study of the software tool. As well as identifying usability defects in the tool, the results of the study inform a model of UIDT evaluation data comparison and analysis strategies that further enhances the facility described in chapter 4. Chapter 8 revisits the aim of this thesis and, drawing the previous chapters together, evaluates the contribution of this thesis to that aim, including identified future work.

Please note: work published by the author prior to this thesis is referenced under the surname McKirdy throughout

---

CHAPTER 2 : LITERATURE REVIEW

2.1 Introduction

Despite increasing numbers of software systems designed to aid user interface development, developers have been provided with little support to assist them in effectively and systematically discriminating between such tools (Bass et al., 1994, Chikofsky et al., 1992, Hix and Ryan, 1992, Hix and Schulman, 1991, Hix et al., 1989, Mosley, 1992, Mosley, 1995, Poston and Sexton, 1992, Sundaram and Ramamurthy, 1996, Valaer and Babb, 1997, Jorgensen et al., 1995, McKirdy, 1998). Software developers are not satisfied with, or adequately supported by, the currently available user interface development tool evaluation and selection facilities (McKirdy, 1998)

This chapter presents a review of the literature concerning user interface development software and its evaluation. It begins by examining, and providing definitions of, the key terminology in this area. The central sections of this chapter focus on the few evaluation facilities that are presently available, outlining the principal components of each and demonstrating the developmental links between them.

Some might argue that the relative lack of evaluation facilities for user interface development software is not important given their similarity to, and relationship with, the established genre of CASE tools, for which evaluation facilities also exist. Drawing comparison between user interface development software and CASE tools in terms of their rôle and available evaluation facilities, this chapter also addresses this issue.

Concluding with a discussion of the advantages and disadvantages of the various evaluation facilities, the chapter justifies the need for and establishes a rationale for the development of a further user interface development software evaluation and selection facility.

2.2 Terminology

2.2.1 User Interfaces

According to Myers, in his review of user interface software tools, the user interface of an application is:

> *"the component that handles output to the display and input from the user – the rest of the system is application semantics."*(Myers, 1994 pp. 2)

or, as defined more fully in an earlier publication, the user interface is:

> *"the software component of an application that translates a user action into one or more requests for application functionality, and that provides to the user feedback about the consequences of his or her action."* (Myers and Rosson, 1992 pp. 3).

These definitions suggest that there is a theoretical boundary between the perceptual representation of an application and its underlying functionality. There are many examples of system code in which the user interface components can be clearly identified and separated from the components concerned with underlying system functionality. For example, consider a standard form-based user interface to a database system - typically, the text boxes and buttons on the user interface possess no system-specific functionality; they simply permit users to enter data for use by, and to invoke, underlying system functionality. There are, however, many situations where such separation is not as easily identifiable. For example, consider a generic drawing tool - the shapes represented on the user interface correspond directly to the underlying model/data. The boundary between the user interface and the underlying system functionality is therefore not always immediately obvious - it could feasibly lie at any point between physical presentation and system-specific data. Furthermore, there are at least three levels of abstraction at which this separation or distinction can be drawn: (1) at the conceptual level in describing the interaction; (2) at the design level in determining the allocation of functionality to either the user interface or underlying system code; and (3) at the implementational level - for example, much functionality can be hidden within user interface features (some of which is application-oriented).

Myers' definition of a user interface might lead one to assume that it is easy to distinguish between the user interface and the underlying functionality of an application. However, the preceding discussion suggests that what counts as the user interface for a given application is a matter of design and developmental separation. Furthermore, this boundary can be influenced by the skill and/or programming style of an application developer. This therefore suggests that when software developers select a tool to assist in user interface development they may: (1) have potentially different expectations of the functionality such a tool should offer and (2) place their selected tool in different rôles within their overall development process. Given that this issue and the potential dichotomy of user interface distinction are taken into consideration, this thesis adopts Myers' definition of a user interface.

2.2.2 User Interface Development Software Systems

Consider now the software systems designed to assist with user interface development. For many years, the term *user interface management system* (UIMS) has been used to refer to such support (Hix and Hartson, 1993). Historically, the term was used to refer primarily to the run-time support for user interface software. Its definition then broadened to refer to an integrated set of interactive programs for the overall process of developing user interfaces - including their design, representation, prototyping, execution, evaluation, and maintenance (Hix and Hartson, 1993). A UIMS must therefore provide design time support and a development environment for producing a user interface, and run-time and/or prototyping support for executing a user interface. Thus, a UIMS needs the means to construct a graphic presentation, to link it to the computational code (or underlying functionality), run-time support mechanisms, and a generator for producing the user interface code from user interface definitions (Hix and Hartson, 1993). Specifically, for a software application to qualify as a UIMS, it must provide the following set of basic elements (Hix and Hartson, 1993):

1. a direct manipulation (non-programming) environment to support display layout;
2. support for linking displays and their computational component using some mechanism other than programming;
3. run-time support mechanisms for sensing user actions on objects and then providing feedback to the display;
4. and a generator for producing user interface code from interface definitions.

An additional category of user interface development systems are known as *user interface development environments* (UIDE). Although the terms UIMS and UIDE are often used synonymously, UIDEs are tools developed specifically for the design and construction of a user interface, and therefore exclude run-time support. According to the definitions of UIMS and UIDE, the latter is essentially an integral part of the former - as shown in Figure 2.1.

Figure 2.1 illustrates the relationships between the functional components of a UIMS[4] and the elements of an application and selected development processes. As shown, a UIMS typically provides the means to design, develop, and evaluate user input and system output, and to link these together; it also provides run-time support to mediate interaction between the user, the interface, and the functional core of the application. Outlining the functional

[4] As listed by Hix and Hartson (Hix and Hartson, 1993).

components of a UIDE, Figure 2.1 illustrates the relationship between the two - a UIDE can be considered a specialised sub-component of a UIMS.

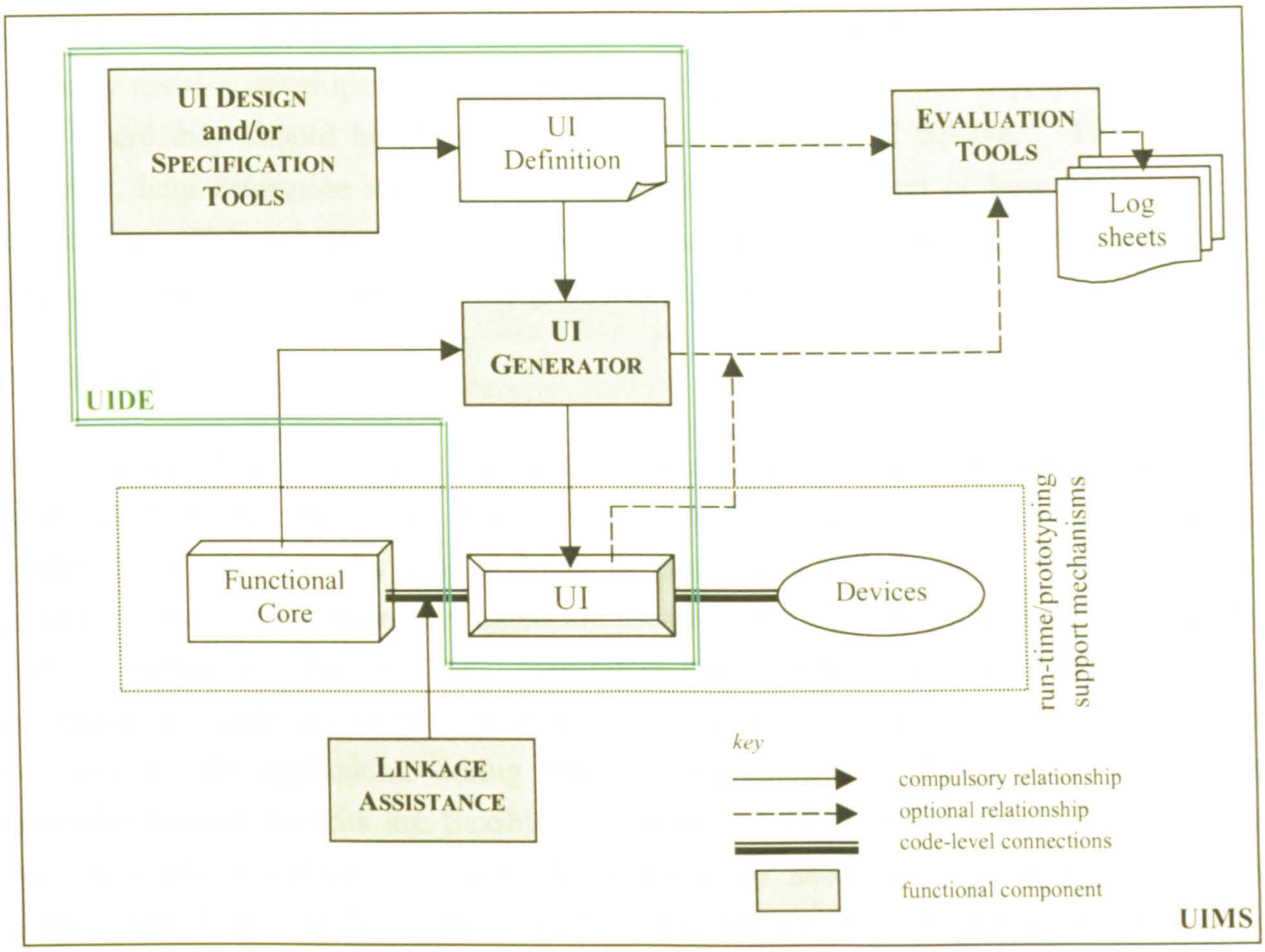


Figure 2.1 - *diagrammatic representation of a UIMS (incorporating a UIDE)*

Finally, in common use today, the term *user interface development tool* (UIDT) refers to:

> *"a software package that provides automated support for any part of the interface development process."* (Hix and Hartson, 1993, pp. 342)

Assuming the above definition, the term UIDT covers the functionality provided by both UIDEs and UIMSs and, to avoid ambiguity or confusion, is used throughout the remainder of this dissertation as an umbrella term to capture the notion of any software assistance for (any part of) the development of a user interface.

2.3 User Interface Development Tools

The stated definition of a UIDT means that a UIDT can be anything from a complete development environment to a single class library. UIDTs can be categorised as follows (Hix and Hartson, 1993):

1. Interaction Style Support Tools

These interactive systems enforce a consistent look-and-feel based upon a specific style and/or standard – for example, OSF/Motif™ and OpenLook™ (Myers, 2000)[5]. They normally assist a developer in determining what specific interaction objects should look like, where they should be placed, and even how they should behave. They do not, however, help determine when to use a given interaction object or how best to use it effectively. Although they help ensure consistency across applications, these tools do not guarantee a product that demonstrates good design or high usability.

2. Toolkits

A toolkit is a library of callable routines or classes used by developers to implement low-level interface features. The basic components of a toolkit - ubiquitously known as 'widgets' - encapsulate the look-and-feel of interaction techniques or activities within graphical objects. Instances of widgets are used to compose a user interface (for example, buttons, menus, scrollbars etc.) and they interact with underlying system functionality or computational code via method invocations which are defined by the programmer and executed by the application during run-time operation. Although their function is generally limited, toolkits are flexible in comparison with other UIDTs. However, to produce a user interface using a toolkit a developer needs to write source-code. In general, therefore, used in isolation, toolkits only support implementation and tend to be the domain of programmers rather than user interface designers. Toolkits are often incorporated into UIDTs such as UIMSs (see below). The Java™ Swing™ or AWT™ class packages are examples of modern toolkits (Eckstein et al., 1998, Jaworski, 1998).

3. Interface Development Tools (IDT) or Interface Builders

In general, interface builders (or interface development tools) are drawing or graphics packages that support little more than the design of static screen displays. Usually based on a specific toolkit, they let a designer select and position widgets to construct a user interface. Limited to supporting the layout of the static parts of a user interface that can be created out of widgets, they cannot handle the parts of a user interface that involve graphical objects moving around (Myers and Rosson, 1992). UIMSs often incorporate interface builders to provide support for user interface specification and layout - for example, VisualCafé™ (Symantec, 1999) has a direct manipulation user interface layout facility which lets a designer select Java™ Swing™ or AWT™ components and include

[5] Examples taken from Myers' catalogue of UIDTs and their reviews.

them in a user interface. Other examples include Display Construction Set by AT&T, WindowsMaker™, and Visaj (Myers, 2000)[5].

4. UIMS

As previously defined, a UIMS is an integrated set of interactive programs for the overall process of developing user interfaces - including their design, representation, prototyping, execution, evaluation, and maintenance (Hix and Hartson, 1993). Of the UIDT classifications outlined here, UIMSs represent the most comprehensive collection of functionality and are subsequently the most *potentially* powerful, and therefore *potentially* beneficial UIDTs. Aside from the facilities a tool must provide to qualify as a UIMS, these tools often incorporate specific toolkits and provide similar facilities to interaction style support tools. For example, VisualCafé™ includes Java™ Swing™ and/or AWT™ and permits a designer to specify a desired 'look-and-feel' (Symantec, 1999). Modern UIMSs can be *informally* (or *loosely*) classified as:

- *application development environments*

  This perhaps the most common sub-category of UIMS, with greatest commercial alliance. In general, application development environments provide the most comprehensive collection of functionality[6] and normally include: the facility to lay out a user interface using some mechanism other than programming or specification languages[7]; a strong basis on a specific toolkit that may or may not support different user interface 'look-and-feel'; the facility to generate all or part of the physical code needed to implement the user interface, and to link this with the underlying system (or computational) code; the facility to execute the generated application user interface - either as a finished artefact or in design/prototyping mode during which debugging and analysis are possible; and facilities to directly program the user interface and/or underlying application code - for example, code editors, compilers, project management etc.

  Application development environments normally support a specific programming language and its associated toolkit(s). Examples include: VisualCafé™, VisualJ++™, JBuilder™ - all of which are based on Java™ and its user interface toolkits, Swing™ and/or AWT™ (Eckstein et al., 1998, Jaworski, 1998); VisualBasic™ which supports development in Basic;

[6] With respect to the accepted defining functionality of a UIMS.

[7] It may even be via declarative models (see below), in which case the environment could be said to include model-based UIDT functionality.

Borland's Delphi™ which is based on Object Pascal; and VisualC++™, based on C++ (Myers, 2000)[5].

- *declarative model-based development tools*

These UIDTs tend to be the result of academic research. Based on a series of models, designers construct a declarative specification of a user interface and its associations with the underlying system code. The models *usually* include: a task and/or dialog model to let a designer abstract over the end-user tasks - including their sequencing - supported by the application being developed; a presentation model that specifies the physical appearance of the user interface[8]; a domain model that represents the structure, or functionality, of the underlying application code; and a user model that models attributes of the end-users of the generated user interface[9]. Using declarative model-based development tools the run-time behaviour of a user interface and application is normally specified by explicitly linking components across the different models. On the basis of such specifications, these tools normally generate all or part of the physical code required to implement the user interface.

Despite a lack of commercial acceptance[10], several model-based UIDTs have been developed, including: Trident, Tadeus, Humanoid, Teallach, Adept, and Mastermind (Bodart et al., 1995, Schlungbaum and Elwert, 1996, Szekely et al., 1992, Griffiths et al., 1998, Johnson et al., 1995, Szekely et al., 1996).

- *multi-media data visualisation tools*

Multi-media data visualisation tools are sometimes little more than interface builders - they generally support dynamic data representation and visualisation, and let designers specify static displays to which they can attach animation. They are perhaps most beneficial for horizontal (and potentially revolutionary) prototyping of user interfaces. Two well known examples of these tools are MacroMedia Director™ and Hypercard™; others include LabVIEW and LabWindows and Vermont Views With Designer (Myers, 2000)[5].

This identified categorisation of UIMS sub-types suggests a potential nesting of functionality within the three categories - the potential for application development

---

[8] This is normally an abstract specification of physical appearance.

[9] According to designer-specified user classifications.

[10] The reason for their lack of commercial acceptance (when compared to application development environments) is as yet unclear and requires further investigation.

environments to include the functionality listed under the remaining classifications was highlighted. Despite a relatively clear classification of user interface development software, allocation of individual tools to specific categories is therefore not always immediately obvious.

2.3.1 Importance of UIDTs

Consider now, the significance or potential benefit of UIDTs – both theoretically (Myers, 1996, Hix and Hartson, 1993), and based on empirical research (Myers and Rosson, 1992).

In independent publications, Myers, Hix, and Hartson agree that use of UIDTs is potentially beneficial (Myers, 1996, Hix and Hartson, 1993). They suggest that user interfaces developed using UIDTs are superior to those developed without the assistance of UIDTs. They attribute this to the following (Myers, 1996, Hix and Hartson, 1993):

i. The facility for rapid prototyping readily accommodates iterative user interface design, which is generally considered paramount for successful user interface development. Modification of user interface designs is therefore made easier which encourages active response to formative evaluation results. Furthermore, the evaluative support provided by many UIDTs makes evaluation easier, and therefore more likely to be conducted;

ii. Tools enable the active participation of a range of interaction development specialists - for example, psychologists - in the user interface design process. A user interface can therefore benefit from additional expertise beyond that of programmers;

iii. Since the tools enable user interface component reuse and can provide interaction style support, user interfaces generated using such tools are more likely to be consistent in terms of look-and-feel;

iv. Since more effort can be applied to developing the UIDT itself than any one user interface, the tools often provide assistance for the most complex aspects of user interface design - for example, implementing help and undo. Applications developed using such tools are therefore more likely to include these features, increasing their potential for higher usability ratings.

Myers, Hix, and Hartson also suggest the following advantages to using UIDTs in terms of the code produced (Myers, 1996, Hix and Hartson, 1993):

i. Their facilities for auto-generation of code mean that there is fundamentally less code to write, making it more economical to create and maintain;

ii. Since UIDTs can potentially enforce issues of structuring and modularity, generated code is likely to be reusable. This code

reusability enables development effort to span several projects, making development more economical and efficient;

iii. Given their provision of evaluative support, the code produced by UIDTs has the potential to be well tested and subsequently more reliable. This is further enhanced by the fact that UIDTs often represent the specification of user interfaces such that specifications can be evaluated and validated more easily;

iv. UIDTs often generate code that is easier to port across platforms, making generated user interfaces more generic and therefore saving implementation effort;

v. Since the level of expertise of the user interface designer and implementers can be lower, the costs involved in user interface development can be reduced (Myers, 1996).

Although one might question some of these claims, Myers, Hix and Hartsons' perspective on UIDTs suggests that there is much to be gained from their use (Myers, 1996, Hix and Hartson, 1993).

Despite their many advantages, UIDTs also have a number of shortcomings. Hix and Hartson suggest that while UIDTs provide admirable support for user interface development, their own user interfaces are often poorly designed and lack attention to detail. Many UIDTs are therefore hard to learn and use - Hix and Hartson go so far as to suggest that the functionality and usability of a UIDT are often inversely related (Hix and Hartson, 1993). They imply that many UIDTs provide only limited functionality, and that to logically and physically separate the user interface and non-user interface components of an application is often very difficult[11]. Most importantly, Hix and Hartson indicate that problems associated with designing user interfaces are not necessarily solved with the introduction of new technology (Hix and Hartson, 1993).

In theory, therefore - as argued strongly by Myers, and with some reservation by Hix and Hartson - the use of UIDTs presents many potential benefits. Is this the case in practice? In 1992, Myers and Rosson conducted a questionnaire-based survey of user interface programming to provide a 'snapshot' of practical user interface design and implementation and thus, in part, attempt to answer this question (Myers and Rosson, 1992). In particular, their goal was to determine the proportion of system code and development time that is typically dedicated to the user interface.

Myers and Rosson published their survey questionnaire in the SIGCHI Bulletin and SIGPLAN Notices[12], distributed it on several electronic bulletin boards, and sent it

[11] Despite being required in many UIDTs.

[12] After acting upon invited feedback regarding a draft version of their questionnaire which they circulated via the SIGCHI e-mailing list.

explicitly to select people. Their responses[13] therefore included some which were recruited directly, but the majority were self-selected. Given that the majority of subjects were self-selected, it is possible that some of the more unusual or less structured development strategies have been omitted from the survey due to factors such as fear of reprisals from the software community or confidentiality restrictions. This should therefore be taken into consideration during the following discussion of their results.

Although the majority of responses came from the USA, proportionally small numbers were received from Europe, Canada, and other countries. The organisations - most of which belonged to the software industry - were asked to complete the questionnaire for a single recently-developed application. When the responses were analysed, Myers and Rosson found that the majority of applications were commercial, military or internal to an organisation, and most were designed to be used by people outside the development team. Applications covered varied domains, were written for a variety of hosts, and had been developed using a selection of programming languages - although C was most common. In general, the respondents were experienced programmers working in teams of between 2 and 10 people - the respondent was normally the project manager.

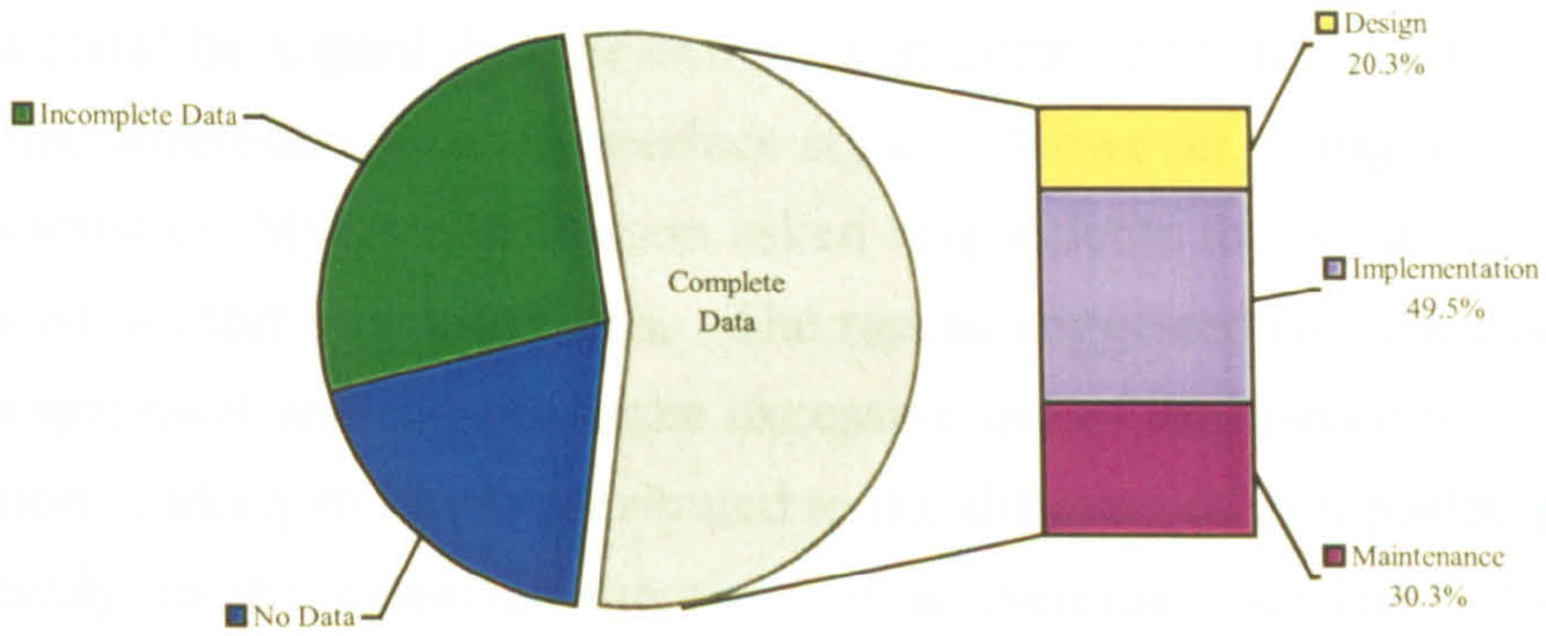


Figure 2.2 - *percentage breakdown of project duration*

Myers and Rosson asked respondents to indicate the percentage of project time that was spent on each of the principal stages of development. Of the 74 respondents, 40 provided complete data for this question. 20 projects did not have sufficient data to answer the question fully, and 14 did not give any response. The results - see Figure 2.2 - illustrate that at least 20% of project duration is typically devoted to system design and 50% to implementation. Therefore, of the overall design and development time of a system, 70% is attributable to tasks in which there is *potential* for substantial UIDT use.

Given that the survey was primarily directed at the user interface-specific aspects of application development, respondents were asked to estimate the percentage of code that

[13] Numbering 74 which were gathered over an eight month period in 1991.

was dedicated to the user interface, the percentage of time spent designing the user interface, the percentage of time spent on its implementation, and finally the time spent on maintaining user interface-specific components of an application. The results for the latter three are shown in Figure 2.3. Myers and Rosson explicitly stated that, when answering these questions, respondents were to exclude user interface components concerned with producing hard-copy output. Although this is insignificant for some application domains, report generation is a substantial component of many database applications. Its exclusion from the reported results may have significantly altered the true figures, and this should therefore be taken into consideration during the following discussion.

An estimate for the percentage of code dedicated to the user interface was provided by 71 respondents. These varied from 1% to 100% and averaged at 47.6%. The extreme variation in the estimated percentages is likely to have been heavily influenced by the difficulty of isolating user interface-specific components (see section 2.2.1) - it may have significantly affected different developers' perspective on component separation and their resultant software architecture, and their ability to identify separation in a consistent manner.

Alternatively, it could be argued that the extremes in estimated percentages were the result of project-specific differences in user interface styles. However, using a checklist of user interface characteristics, Myers and Rosson asked respondents to indicate the components that were included in their user interfaces. The results suggested that the majority of user interfaces were graphical and did not make excessive use of multi-media. Although user interface variation is likely to have contributed to the differences in reported percentages – and most probably to the extremes therein – it is therefore not likely to be the sole influence. Ignoring influence, the average estimated percentage suggests that user interface-specific code accounts for a substantial part of overall system code. It is therefore a significant factor in application development - one that can be affected by UIDT use.

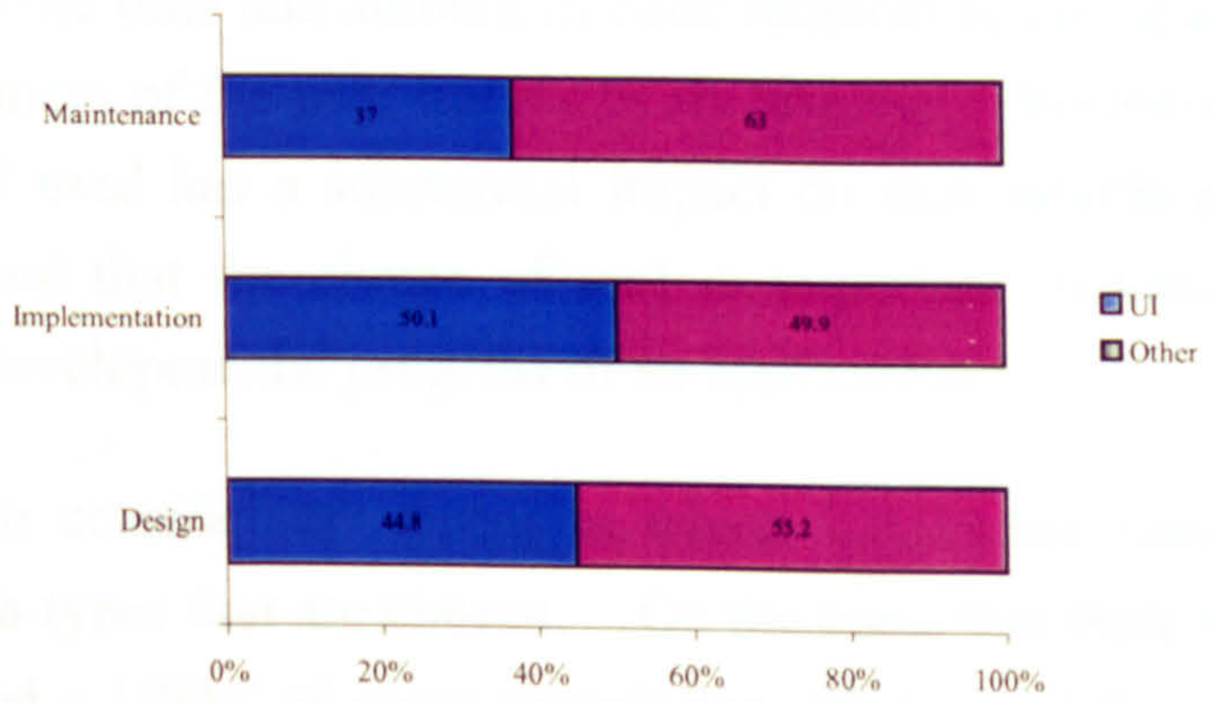


Figure 2.3 - *% breakdown of core project stages*

Figure 2.3 shows that the user interface typically commands a substantial proportion of design and implementation effort. Relative to the project as a whole, there is therefore considerable potential scope for the introduction of dedicated software to assist in the design and/or implementation of the user interface.

When they correlated the breakdown of core project stages with reported UIDT use, Myers and Rosson found that projects that only used a toolkit devoted more time and code to the user interface (64.9% and 57% respectively) than projects that used no tools (44.0% and 45.2% respectively). They attribute this to the fact that the systems developed using toolkits had more sophisticated user interfaces.

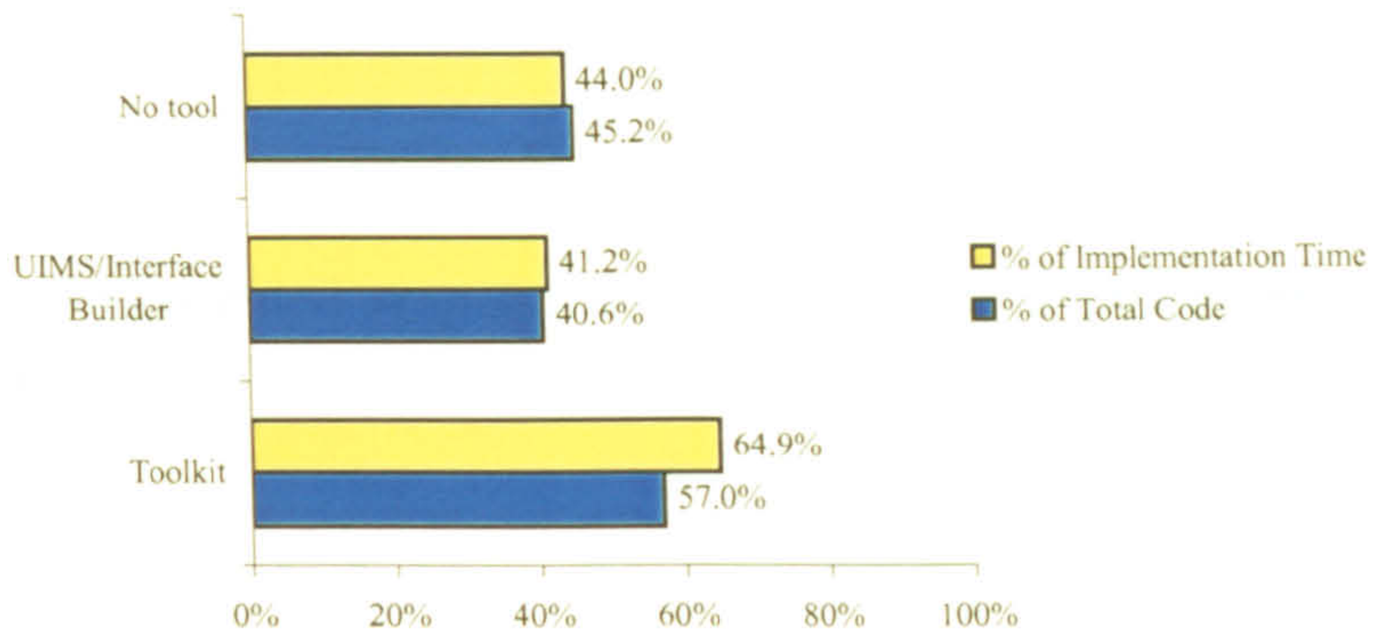


Figure 2.4 - *% of time and code devoted to a user interface in relation to tool type use*

Figure 2.4 shows that the projects that used UIMSs or interface builders spent the least time and code on the user interface (41.2% and 40.6% respectively). If, as Myers and Rosson suggest, projects which employed a UIDT were attempting more in terms of their user interfaces than projects which used no tools, the difference in the figures for toolkits and UIMSs/interface builders could be a consequence of toolkits' reduced facility for rapid prototyping impacting on both the time and the amount of code required.

Ignoring extraneous factors, these results suggest that the use of UIDTs can potentially influence not only the time and amount of code required to create a user interface, but also the ambitious element of the user interfaces themselves. Furthermore, they indicate that the type of UIDT used has a substantial impact on user interface development. It can therefore be argued that the choice of tool is important in terms of its benefit and/or influence on the developmental progress of an application.

It is interesting to consider the extent to which UIDTs are used, and in particular the specific UIDT sub-types that are chosen. On the basis that their survey showed that 74% of companies used a UIDT of some description, Myers and Rosson claimed that UIDTs were in extensive commercial use. Figure 2.5 shows that approximately one third of all projects used a toolkit. A similar proportion used a UIMS – although often in conjunction

with a supporting toolkit and/or interface builder. Given the types of tools listed, it would appear that they were being used more for implementation purposes - for example, the high use of toolkits - than for design[14].

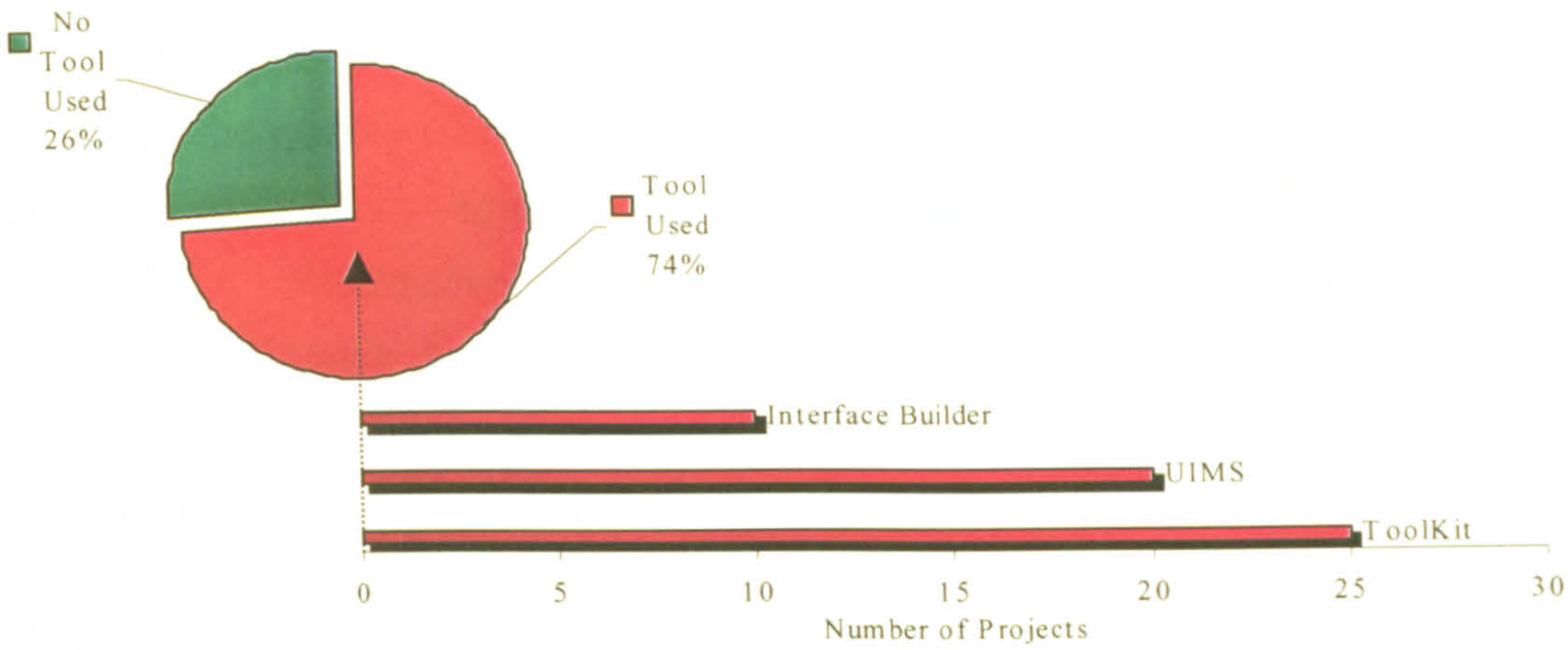


Figure 2.5 - *breakdown of tool use*

Myers and Rosson investigated respondents' developmental processes. The results are shown in Figure 2.6.

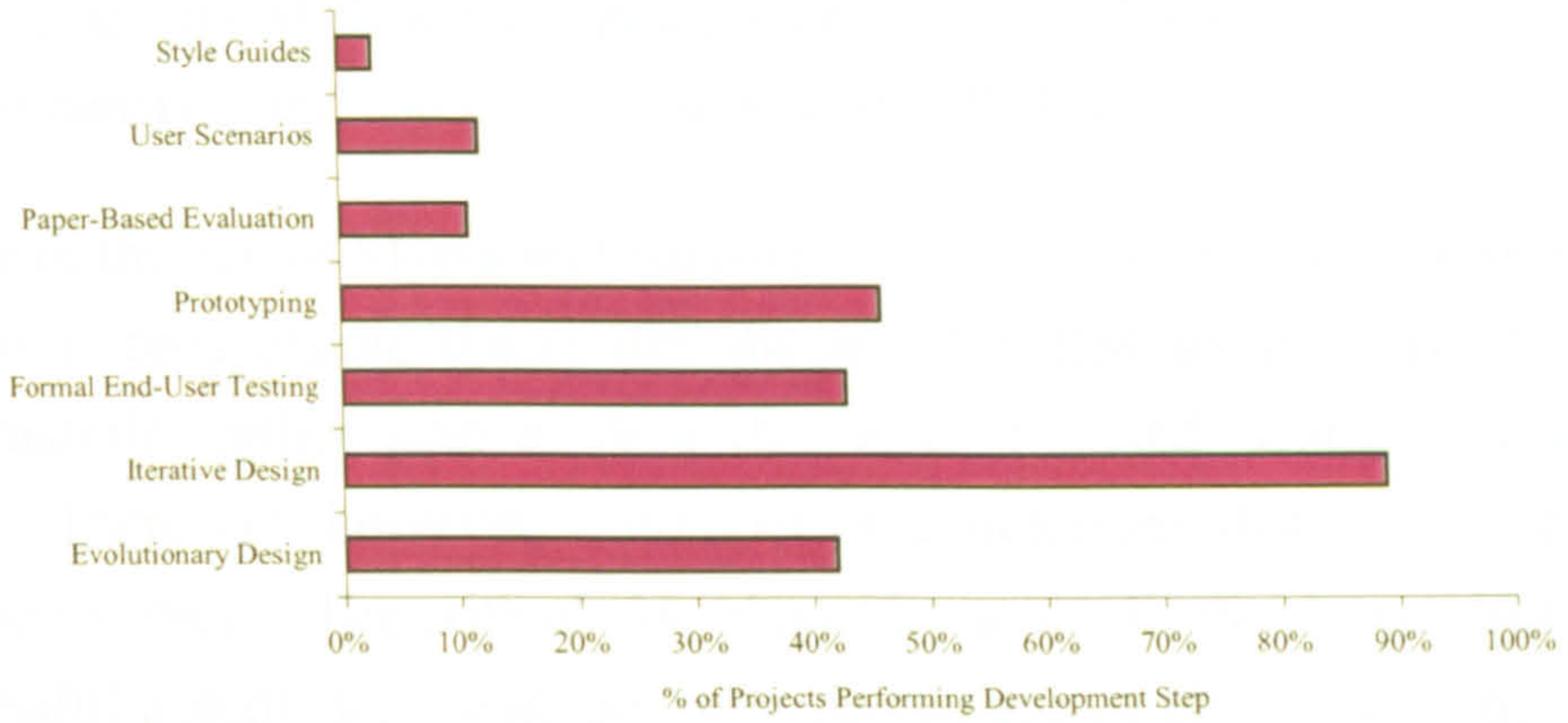


Figure 2.6 - *adoption of user interface development steps*

These results were not correlated with UIDT use and so no relationship between user interface development strategies and tool use can be identified. They do, however, highlight the potential scope for UIDT use and illustrate the facilities a UIDT must provide to adequately support the way in which developers work. In particular, a UIDT would complement the identified development strategies if it supported evolutionary iterative design and development, including the facility to prototype designs and to gather usability or evaluation data.

[14] Since Myers and Rosson make no mention of the manner in which the tools were actually being used, this observation cannot be substantiated..

When asked about code reuse and modularity, 51% of respondents reported that they had been able to reuse code developed using a UIDT, and 76% claimed that the systems they developed were sufficiently modularised to allow user interface modification without altering the underlying system functionality. However, code produced using a toolkit was reported to be so tightly coupled to the toolkit that it lacked modularisation. This suggests that, in terms of the quality of resultant code, developers may have had to trade the power and flexibility of toolkits against the prototyping-empowered, but somewhat more restrictive, UIMSs. Alternatively, these results may illustrate the influence of tool type in terms of the style in which developers were able to program – toolkits allowing developers relatively free rein in terms of software architecture, in contrast to the more (often enforced) structured style of UIMS programming.

Myers and Rosson asked respondents to comment on advantages and disadvantages of UIDT use. In general, they included speed of development and improved productivity as advantages of using UIDTs. Disadvantages were listed as: performance problems and bugs in the UIDTs themselves; difficulty learning to use the tools; and the fact that the tools themselves were often slow. While confirming Hix and Hartsons' observation that UIDTs often have user interfaces that are poorly designed and lack attention to detail (see section 2.3.1), the listed disadvantages illustrate issues that could potentially be avoided given comprehensive, and contextually sensitive, UIDT selection.

On the basis of the age of Myers and Rossons' survey, together with the speed at which the UIDT sector is progressing, the results discussed in this section have been included as indicative material rather than as data that is central and strongly evidentiary to this research[15]. There are, however, a number of conclusions that may be drawn from the results of their survey. The critical rôle played by an application's user interface - in terms of its successful acceptance - and the large percentage of effort and code dedicated to its development suggests that user interface development *must* be supported as far as is effectively technologically possible. Further, Myers and Rossons' results suggest that increased user interface development support is linked to increased ambition in terms of the user interface design[16]. Several software developers have identified the potential scope for UIDT use within software development projects and have responded accordingly by producing a multitude of UIDTs - see section 2.3 - to establish a distinct software market sector. Together, these factors confirm the demand, if not the inherent need, for UIDTs in software development.

---

[15] A similar survey has been conducted as part of this thesis, and is discussed in detail in chapter 3.

[16] Although it is not clear which was the source of the link - they may, in fact, be mutually dependent or supportive.

Myers and Rossons' results demonstrate the impact of tool type on a project. Their identified practical user interface development activities highlight the facilities that UIDTs need to provide to be sympathetic to the manner in which developers work. Together with the opinions expressed by the respondents, these observations illustrate some of the important aspects of UIDTs that should be evaluated when selecting a tool for use with a project. Furthermore, they indicate that UIDTs need to be evaluated within the context in which they are to be used - especially given the potential number of UIDTs from which to choose.

2.4 Selection Techniques for UIDTs

Given the established rôle, demand for, supply, and uptake of UIDTs, the lack of effective and systematic support for UIDT selection is surprising. This section considers the available means - prior to the work in this thesis - by which designers were supported when attempting an informed UIDT selection.

2.4.1 Selection of UIDTs According to Hix *et al*

The work of Hix *et al* pioneered UIDT evaluation, highlighting the *need* for methodologies designed specifically for evaluation or selection of UIDTs (Hix, 1986, Hix et al., 1989, Hix and Schulman, 1991, Hix, 1991, Hix and Ryan, 1992).

More than a decade ago, Hix, Tan and Schulman recognised the absence of a *systematic* procedure for evaluating UIDTs. Based on evaluation techniques for applications such as text editors, they developed and empirically tested an evaluation procedure for UIDTs that aims to produce quantifiable criteria for the comparison of such tools (Hix et al., 1989). Their mechanism relies on 'hands-on' learning of each of the tools being evaluated. When familiar with a tool, an evaluator completes a 28 page form to record specific information about the functionality and usability of the tool. The first of these dimensions - functionality - refers to the capabilities of the system, and is divided into three subsections: types of interface supported; types of support for interface development; and general characteristics. Each of these is further subdivided into lists of appropriate items. For example, the list for types of interface support includes menus of which there are 14 types listed (Hix et al., 1989, Hix, 1991, Hix and Schulman, 1991, Hix and Ryan, 1992).

The second dimension - usability - refers to how satisfactorily a tool performs identified functions. Hix, Tan and Schulman suggest that this is assessed: subjectively - how difficult or easy the functions are to perform; and objectively - how efficiently the tool can be used to perform a series of benchmark tests. An evaluator also records the primary

specification, or implementation, technique adopted by the tool to access each individual function. To reduce ambiguity and individual interpretation across evaluators, they include a glossary of terms with their evaluation forms (Hix et al., 1989, Hix, 1991).

After completing a form for each of the tools being considered, an evaluator enters the data into a purpose-built spreadsheet - which is then used to calculate three evaluation ratings. A functionality rating represents the percentage of the possible interface functions that are actually provided by a tool. A usability rating is a percentage representation of the actual usability score against the highest possible usability score[17]. Finally, a specification/implementation rating reflects the extent to which a particular interaction technique is adopted by a tool for accessing provided functions.

Hix, Tan and Schulman conducted empirical studies of the use of their checklists. Each subject spent between 6 and 10 hours learning to use a tool before completing the evaluation forms as outlined above. Hix, Tan and Schulman then calculated Cronbach's alpha to assess consistency across evaluators. They found that high alpha ratings[18] were clustered around certain types of functionality - in particular, those relating to input and output devices. In contrast, they discovered lowest alpha ratings corresponded to aspects of usability, which therefore had the greatest scope for cross-evaluator variance. They found that, in some cases, one evaluator marked a feature impossible where another marked it possible. Hix, Tan and Schulman suggested two potential reasons for this inconsistency: differences in the way in which evaluators interpreted the glossary terms; and variation in evaluator experience of a particular tool (Hix et al., 1989). In response to their study, Hix, Tan and Schulman conceded the probability that their procedure can only be used effectively by expert evaluators who are conversant with their checklists, and who thoroughly learn the systems they are evaluating (Hix et al., 1989). However, they also record that their procedure has been used successfully in several commercial environments.

In 1992, Hix and Ryan revised the original procedure. New interaction components were included, the procedure was enhanced to reflect the needs of multi-media interfaces, and interface implementation issues were taken into consideration. Following these extensions, a complete tool evaluation comprises: a general description of the tool; information about the materials used to prepare the evaluation; an executive summary of the functionality and usability ratings for the tool; and a glossary containing definitions for each term on the form - a physical extension of 30 pages (Hix and Ryan, 1992, Hix, 1991).

---

[17] It is calculated only against functions that are provided.

[18] Equating to high evaluator reliability.

Hix and Ryan conducted a further series of empirical studies on their modified checklists. Subjects spent approximately 20 hours learning and exploring each tool, before taking 5 to 6 hours to complete each evaluation form. Each subject then spent a further 2 to 3 hours generating the ratings using the electronic spreadsheet. After analysing their findings, Hix and Ryan acknowledged that, despite the considerable length of time devoted to learning each tool, subjects may not have been able to determine every available function (Hix and Ryan, 1992). They therefore conceded that evaluators will often be unable to complete every line in their checklists with total confidence - a potential drawback to their method. They suggested that developers should not select a tool solely on the basis of an evaluation completed using their procedure. Instead, the results should be viewed as quantitative data to inform and assist the decision making process (Hix and Ryan, 1992).

Stressing that their approach aims to evaluate UIDTs and *not* the user interfaces produced using the tools, Hix and Hartson justified the choice of factors included in the checklists (Hix and Hartson, 1993). Their evaluation procedure is based on the following high-level discriminators (Hix and Hartson, 1993):

1. functionality of the tool;
2. usability of the tool;
3. easy-to-use graphical interface of the tool itself;
4. ability to produce direct manipulation user interfaces via direct manipulation;
5. styles supported by the tool;
6. customisation (addition/modification) of widgets;
7. creation of dynamic interaction objects;
8. support for formative evaluation and iterative refinement;
9. type of control structures and callbacks;
10. portability of the user interface produced by the tool;
11. changes to the user interface design independent of code generation;
12. run-time performance of the interface produced by the tool;
13. cost, documentation and customer support.

From this list, centred around the three dimensions already discussed, they developed their 58 page form-based checklist. With the introduction of their checklist-based evaluation procedure, Hix *et al* had broken new ground in terms of UIDT evaluation and selection (Hix, 1986, Hix et al., 1989, Hix and Schulman, 1991, Hix, 1991, Hix and Ryan, 1992).

2.4.2 An Extension to the Hix *et al* UIDT Evaluation Procedure

In 1996, Sundaram and Ramamurthy extended the work of Hix *et al* (Sundaram and Ramamurthy, 1996). They added two dimensions - run-time efficiency and learning curve - and, while they acknowledged that the original dimensions of functionality and usability are comprehensively dealt with by Hix *et al*, suggested that a five point scale for usability be introduced to increase discrimination (Sundaram and Ramamurthy, 1996).

Sundaram and Ramamurthy stated that it is important that an application's user interface is not detrimental to the run-time efficiency of the software. To test this, they suggested that run-time benchmark tests be introduced. Although such benchmarks are usually characterised along the dimensions of space, time, and control, they were concerned with the execution speed of the object code and its related overheads and latency. They also included other run-time parameters, such as space utilisation and capacity (assessed by measuring memory areas such as the stack and heap). They therefore measure run-time efficiency as shown in Figure 2.7 (Sundaram and Ramamurthy, 1996).

$$\text{Run-time Efficiency of UIDT} = T_u / T_h$$

where: T_u = time taken by the application to execute with the user interacting through the interface developed with the UIDT
T_h = time taken by the application to execute if all data was hard-coded (no user interface)

(Sundaram and Ramamurthy, 1996)

Figure 2.7 - *run-time efficiency*

They suggested that the quality of a UIDT is also indicated by its associated learning curve (Sundaram and Ramamurthy, 1996). This is supported by the findings of Kemerer, who discovered that 70% of the features and techniques in a CASE tool are never used due to the effect of the learning curve (Kemerer, 1992). Sundaram and Ramamurthy quoted studies that show most users abandon software environments before appreciating their advantages due to frustration and difficulties in the initial adaptation stages (Sundaram and Ramamurthy, 1996). On this basis, they concluded that to be accepted, a UIDT should be easy to learn and have a gentle-sloping learning curve (Sundaram and Ramamurthy, 1996). They assess the learning curve associated with a UIDT according to either of the following measures (Sundaram and Ramamurthy, 1996):

1. the time taken to learn to use the UIDT - this can be measured by a standardised test that evaluates a user's knowledge for each item in the functionality dimension;

2. the time taken to develop a benchmark application.

Sundaram and Ramamurthy acknowledged the indirect influence of cost on UIDT selection, but claimed that because it does not directly affect the quality of a user interface it would not be included in their methodology. They do not reference the intended use for which the original method was developed, but suggest that their extended method could be used by UIDT *developers*, prior to releasing their software, to improve their products. It is therefore unclear whether their intention was to restrict the use of their augmented methodology to this new scope, or whether they anticipated that it also be used according to its original remit. Nevertheless, by introducing additional and wider perspectives on UIDT evaluation, they have made a valuable contribution to the work of Hix *et al* (Sundaram and Ramamurthy, 1996).

2.4.3 An Alternative UIDT Evaluation Facility

In 1994, Bass, Abowd and Kazman introduced an alternative perspective on UIDT evaluation (Bass et al., 1994). Focusing on user interface builders, they proposed an evaluation framework based on selected quality attributes of both the tool, and the systems constructed using the tool - the target systems. They suggest that the preceding work of Hix *et al* only evaluates one quality attribute - functionality (Hix, 1991, Bass et al., 1994).

Acting on a hypothesis that interface builders should be evaluated according to a set of quality attributes, Bass, Abowd and Kazman considered a number of software taxonomies[19]. These taxonomies specify different relationships between quality attributes - which are often split into primary and secondary attributes - of a software system. The overall quality of software is evaluated by rating the system with respect to individual quality attributes, and then weighting these ratings to achieve an overall rating. Bass, Abowd and Kazman focus on what they consider to be the primary quality attributes with respect to the evaluation of interface builders, and abdicate responsibility for generating the overall rating to the organisation (Bass et al., 1994).

Not only do they suggest that evaluation of interface builders needs to consider the tool and target systems, they also require that the person(s) most concerned with the quality being evaluated - the tool user (the developer) or the target system user (the end-user) - are also considered. Their evaluation framework focuses on the six quality attributes the authors consider most important on the bases of their personal experience. These are:

[19] For example, ISO 9126 which has become accepted as an international standard.

1. *Modifiability* of the target system.

Drawing on Oskarsson's characterisation of classes of modifications (Oskarsson, 1982), Bass, Abowd and Kazman consider the following to be the most relevant to the user interface domain: (1) extension of capabilities - adding new functionality and enhancing existing functionality; and (2) adaptation to new operating environments. They suggest that, despite the above classifications, modification is an abstract concept. For example, the ease of modification can vary across systems depending on the manner in which the collection of functionality has been implemented. They therefore suggest that an evaluator needs to define a set of likely modifications and then conduct the evaluation with respect to samples from that set.

2. *Construction efficiency* (or time taken to develop the target system);

Bass, Abowd and Kazman identify two aspects of construction efficiency: (1) the efficiency of constructing the target system components; and (2) the efficiency of integrating the components.

3. *Usability* of the target system (as indicated by the style of interaction provided to the end-user);

Bass, Abowd and Kazman state that to evaluate interface builders it is necessary to consider the types and styles of user interfaces that can be constructed. They refer to the Hix *et al* checklists as the means by which to perform this aspect of evaluation (Hix and Schulman, 1991).

4. *Compatibility*, both of the tool within the development environment, and of the user interface generated by the tool within the rest of the target system;

Acknowledging that an interface builder is only one part of a development environment, Bass, Abowd, and Kazman state that the compatibility of an interface builder is dependent on its ability to operate in an integrated fashion with other elements in the environment. Further, they observe that the user interface developed using the tool is only one part of the final target system and so must integrate and operate with the other parts. Language and platform compatibility therefore become significant.

5. *Reusability* of previously existing components within the target system;

According to Bass, Abowd and Kazman there are three types of reuse: (1) reuse of components within the current development environment; (2) reuse of components from previously developed applications; and (3) reuse of components for the benefit of future development environments. On the basis that there are several standard user interface widget sets available, their evaluation framework focuses on the second of these. They claim that reuse of components is dependent on their compatibility with the target system in terms of their functionality and the internal types from which they are constructed. Since these cannot be considered in the abstract sense, reusability must be evaluated on the basis of an evaluator-selected toolkit.

6. *Response time* of the target system.

The structural types used for computation and co-ordination must be inspected to evaluate response times.

	Tool	Target System
Developer	Compatibility Construction efficiency	Modifiability Compatibility Reusability
End User		Interaction Style (Usability) Response Time

Table 2.1 – *categorisation of quality attributes*

Table 2.1 shows Bass, Abowd and Kazmans' categorisation of quality attributes (Bass et al., 1994). Significantly, Bass, Abowd, and Kazman make a clear distinction between the framework of the evaluation - the process - and techniques used to perform the evaluation - the mechanism(s). They highlight four evaluation techniques - feature inspections, architectural inspections, direct developer input, and benchmarking - and suggest which technique is most appropriate for the evaluation of each of their identified quality attributes. The primary rôle of the listed mechanisms is to support data gathering. Although these mechanisms are not therefore the immediate focus of this thesis, they highlight an interesting research issue as to which technique(s) would most effectively support data-collection for the central evaluation methodology proposed by this thesis. They are therefore discussed as further work in chapter 8. By focusing on quality attributes, Bass, Abowd and Kazman introduced a radically different approach to evaluation of interface builders.

2.4.4 A Less Structured Selection Technique Introduced by Myers

By identifying a list of important functions that UIDTs should provide, Myers suggested an informal checklist technique for evaluating UIDTs (Myers, 1996). The functions include:

1. help to design the user interface given a specification of the end-users' tasks;
2. help to implement the user interface given a specification of the design;
3. help to evaluate the user interface and propose improvements;
4. ability to create 'easy-to-use' user interfaces;
5. facility for a designer to rapidly investigate different designs;
6. facility for a non-programmer to design and implement user interfaces;
7. facility to enable the end-users to customise the user interfaces;
8. portability;
9. and ease-of-use.

Myers further suggests that these functions can be achieved by having the tool do the following (Myers, 1996):

1. automatically choose which styles/widgets to use;
2. help with screen layout and design;
3. validate user inputs;
4. handle user errors;
5. handle aborting and undoing of operations;
6. provide appropriate feedback to show inputs received;
7. provide help and prompts;
8. update the screen display when application data changes;
9. notify the application when a user modifies graphical objects;
10. handle field scrolling and editing;
11. help with sequencing of operations;
12. insulate the application from device dependencies and underlying software and hardware systems;
13. provide customisation facilities to end-users;
14. and evaluate the graphic design and layout, usability, and learnability of the user interface.

The second list can be used to augment the first such that a richer high-level checklist may be constructed and used to evaluate UIDTs.

2.4.5 User Interface Development Tool Evaluation According to Valaer and Babb

The most recent addition to UIDT selection was introduced in 1997 by Valaer and Babb (Valaer and Babb, 1997). Although they do not cite it, their research appears to be in accordance with much of the work previously discussed in this chapter. Basing their work on the assumption that UIDTs can have drawbacks as well as advantages, they suggest that a tool which is hard to use retards the learning curve, and subsequently reduces the efficiency of development (Valaer and Babb, 1997). They therefore state that the fundamental decision a developer must make is whether or not to use a UIDT at all, and that this decision should be based on whether:

- the inclusion of a tool would help simplify the development process;
- the inclusion of a tool would help create a better product.

They suggest that if the introduction of a tool would not help achieve these goals, it may not be a worthwhile investment (Valaer and Babb, 1997).

Valaer and Babb have designed a checklist-based, two-step evaluation process to inform UIDT selection. Using their method, a developer first selects a category of tools and then evaluates the tools within that category. Consider first, their categorisation of tools.

2.4.5.1 UIDT Categorisation Proposed by Valaer and Babb

Asserting that UIDTs can be divided into categories according to their advertised functionality, Valaer and Babb categorised such tools[20] as shown in Table 2.2 (Valaer and Babb, 1997).

They appreciate that tools for HTML web page construction, development of virtual reality environments, and Java™ programming continue to emerge and that each warrants independent investigation. Recognising their potential for future research, Valaer and Babb suggest that these genres are outside the immediate scope of their work.

Valaer and Babb state that, when selecting a UIDT, developers must first decide which of the tool classifications (see Table 2.2) best describes the type of tool they require for their specific projects. Having focused on the most appropriate category (or categories) of tools, they must identify a set of tools to evaluate and from these make an informed selection based on their evaluation results. The following section describes the evaluation process Valaer and Babb developed to assist comparison of candidate tools.

[20] Valaer and Babb focus on UIDTs for the development of graphical user interfaces.

GUI Tool Category	Description
GUI Builder *or* Interface Development Tool	provides a simple interface development environment in which a developer can create and manipulate user interfaces using a WYSIWYG interaction mechanism
User Interface Management System (UMIS)	provides the functionality of a GUI builder as well as the ability to define and execute the functionality of the interface – either from within the tool or by generating code to allow independent execution of the interface
Graphical User Environment *Or* Data Visualisation Tool	provides dynamic data representation and visualisation which allows the developer to define static displays and then attach animation to the objects
Specialised Widgets	provide functionality beyond the basic capabilities of a typical widget set
Plotting & Analysis Tool	provides the designer with the ability to display technical data via a variety of generic and/or custom graphs or plots – some tools also provide the facility to perform technical analysis on such data
3-Dimensional Visualisation Tool	provides specialised support for creation and manipulation of 3D graphics – most practical on hardware platforms which support the calculations required in 3D graphics
Cross Platform Development Tool	provides the ability to generate a common look-and-feel across different platforms via the use of a layered application program interface (API) which adds a layer of software between the application and the native windowing system
GUI Porting Tool	provides automatic porting of user interface code to different platforms
Application Development Tool	provides miscellaneous capabilities beyond user interface design and development – the additional functionality (e.g. database access) may be useful in developing the rest of the application

Table 2.2 – *GUI development tool categorisation*

2.4.5.2 Valaer and Babbs' Evaluation Criteria

Valaer and Babb identified a number of criteria which they claim assist tool comparison. They suggest it is important to evaluate actual copies of the candidate tools against these criteria, which are as follows (Valaer and Babb, 1997):

- *Usability* - a tool should be easy to install and use, should provide appropriate levels of help, and should be suited to the skills mix of the team;
- *Functionality* - the basic functionality of the tool should match the requirements of the interface;
- *Flexibility* - a tool should be responsive to changing requirements in the application;
- *Portability* - the platforms which are supported by the tool and the ease with which the interface can be ported between them;

- *Support* - the inclusion of documentation, training, online help, and other materials;
- *Cost* - the costs related to obtaining and using a tool.

To guide tool comparison according to their identified criteria, Valaer and Babb generated a checklist that includes a set of questions under each criterion (Valaer and Babb, 1997). They do not claim that the list of questions is exhaustive, and state that the questions are not intended as the basis for numerical UIDT ratings. Their purpose was to provide a list of the critical issues they suggest a developer should consider when evaluating candidate UIDTs. Furthermore, they claim that their questions should trigger the generation of more application-specific questions (Valaer and Babb, 1997).

Valaer and Babb recognise that a developer's choice of UIDT is rarely black and white, and that it will be affected by project-specific requirements. They suggest that a developer should choose the tool which is not only the most efficient with respect to its intended use, but also makes the biggest contribution to creating an effective user interface for the given application (Valaer and Babb, 1997).

2.5 Related Concepts – CASE Tools and Their Evaluation

Although this research focuses on UIDT evaluation or selection, it is worth considering (for comparison) evaluation of other software development tools - namely, computer assisted software engineering (CASE) tools.

CASE tools, like UIDTs, are relatively generic - they are applicable across a wide range of software development and can be put to many uses. Furthermore, CASE tools can be used in all situations where UIDTs can be employed. This therefore makes their evaluation facilities an ideal choice for comparison with those for UIDTs.

2.5.1 What Constitutes a CASE Tool ?

Brown *et al* define a CASE tool as:

> *"a computer-based produce aimed at supporting one or more software engineering activities within a software development process"* (Brown et al., 1994)

A CASE tool is therefore a software tool designed to aid software engineering activities which include (but are not limited to) requirements analysis/capture, software design, code production, testing, document generation, quality assurance, configuration management, and project management (Jorgensen et al., 1995).

2.5.2 CASE Tool Evaluation

In 1992, around the same time as UIDT evaluation facilities were emerging, Chikofsky, Martin, and Chang investigated the state of software tool assessment (Chikofsky et al., 1992). They concluded that software development techniques, and the technology used to apply them, were very under-researched. They found no widely accepted, systematic approaches to evaluating a tool's utility. Although they acknowledged the existence of credible and thorough tool reviews, they warned that such reviews are based on subjective information, the criteria upon which they are based alters with each report, they are not repeatable, and they lack an evaluation framework (Chikofsky et al., 1992).

Chikofsky, Martin, and Chang claim that many organisations make decisions about tools based purely on highly visible criteria, and that assessment of a tool's usefulness is often closely tied to the success or failure of a project (Chikofsky et al., 1992). They state that:

> *"Good articles that disseminate sound work on tools assessment require a lot of effort – they are a rarer commodity than you might think."* (Chikofsky et al., 1992, pp. 20).

They proposed that tool evaluation must at least include the following five aspects, which they suggested should be accommodated by any future evaluation method (Chikofsky et al., 1992):

1. evaluation of the strengths and weaknesses of tools;
2. comparison of tools;
3. selection of tools;
4. tailoring or customisation of tools;
5. and reviewing of tools in use to determine what users did with them and why.

Consider now, the various attempts which have been made to develop evaluation facilities for CASE tools.

2.5.3 CASE Tool Evaluation According to Vessey *et al*

In 1989, Vessey, Jarvenpaa, and Tractinsky investigated the type and extent of support that commercially available CASE tools provide for structured systems analysis and design (Vessey et al., 1992). They categorised the philosophies adopted by analysis and design CASE tools for controlling and/or checking the validity of the various artefacts produced using these tools. The categories include: (1) *restrictive* - CASE tools which encourage normative use; (2) *guided* - CASE tools which encourage but do not enforce normative

use; and (3) *flexible* - CASE tools which allow their users complete freedom. Having theoretically identified differences in the manner in which CASE tools validated their products, Vessey, Jarvenpaa, and Tractinsky investigated the type and extent of product control/validity checking that is adopted by commercially available structured systems analysis and design CASE tools (Vessey et al., 1992). To conduct their investigation, Vessey, Jarvenpaa, and Tractinsky developed an evaluation tool which they based on generally accepted rules for structured analysis and design techniques (Vessey et al., 1992). These techniques included:

- *structured analysis* - the products of which include a set of data flow diagrams and supporting data dictionary;
- *entity-relationship modelling* - which produces entity-relationship diagrams and entries in a data dictionary;
- *structural design* - the products of which are structure charts;
- *logical data design* - which produces data structure diagrams.

The result was: (1) a restricted set of questions focussing on the control and checking mechanisms in place within a CASE tool during the generation of the artefact associated with each technique; and (2) the control and checking mechanisms in place when linking the various artefacts produced using each technique (Vessey et al., 1992). Using their question-based evaluation mechanism, an evaluator must determine the answer to questions such as *'must parent processes be specified before their child processes?'* and *'must entities be specified before their relationships?'*. Guided by their evaluation mechanism, Vessey, Jarvenpaa, and Tractinsky were able to generically compare CASE tools in terms of the number and type of checks adopted during the generation of their associated artefacts (Vessey et al., 1992).

Several years later, Vessey, Jarvenpaa, and Tractinsky identified the significance of the use of CASE tools in collaborative software design (Vessey and Sravanapudi, 1995). They developed a model of collaborative support in which they identified taskware, teamware, and groupware levels. Suggesting that current CASE tools only support systems design at the taskware level (that is, support for tasks performed by an individual), they modelled the hierarchy of support that a CASE tool would need to provide for it to fully accommodate teamware and groupware use (Vessey and Sravanapudi, 1995). These components include elements for: (1) collaborative support; (2) co-ordination support; and (3) co-operation support. For each component in each model, Vessey, Jarvenpaa, and Tractinsky identified a list of high-level questions that constitute a checklist of the requirements a tool must satisfy to support the associated collaborative element - in essence, a model of collaborative support requirements coupled with an evaluation instrument by which to organise the assessment of the extent to which CASE tools provide collaborative support.

Using the Vessey *et al* evaluation mechanism, evaluators must answer questions such as *'is it possible to simultaneously display the same diagram on different workstations?'* and *'is it possible to access the data dictionary concurrently?'* to generically compare CASE tools in terms of their ability to support collaborative systems design.

2.5.4 CASE Tool Evaluation According to Mosley

As a member of the Software-Tools Evaluation Committee (STEC)[21], Mosley made significant contributions to CASE tool evaluation. In particular, she helped create a six page template that later evolved into the standard by which many engineers reviewed and classified tools (Mosley, 1992, Mosley, 1995). In 1986, Mosley became involved in the Software Engineering Institute (SEI) and Westinghouse joint initiative to further develop the set of standards and generate a related tool assessment guide. The resulting SEI/Westinghouse Guide to tool assessment includes (Mosley, 1992):

- a tool taxonomy that captures information like the tool name and vendor, release date, what life cycle phases and methods it supports, and the objects (for example, diagrams and code) it produces;
- and questions designed to determine how well a tool does what it was intended to do.

The set of questions includes 140 generic questions which are divided into the following categories (Mosley, 1992):

1. ease-of-use – the ease with which the developer can interact with the user interface of the tool (including the developer's ability to tailor the user interface);
2. power – the performance of the tool (for example, how fast does it run);
3. robustness – issues such as compatibility between versions of the tool;
4. functionality – the aspects of developmental methodology that are followed, and the actual facilities that are provided;
5. ease of insertion – the ease with which the tool can be introduced into a working environment (for example, the ease of installation and range of platform compliance);
6. and quality of support – the extent to which support such as maintenance assistance, hotlines, online tutorials etc. are provided.

When she tried to use the questionnaire, Mosley found that it was of little use in its raw form (Mosley, 1992). In particular, she found that there was no way to record responses consistently, comprehensively, and completely. In practice, more weight was being given

[21] Formed in 1985.

to questions which dealt with a tool's functionality and its ease-of-use, than to other components in the evaluation. Mosley therefore decided to apply a 0 – 10 scoring criterion and assign a weight to each question (to four or five decimal point accuracy). It took 500 man-hours and over four months to introduce these facilities into the evaluation questionnaire. Unfortunately, the revised weighted list was found to be too time-consuming, and it was found to be difficult to distinguish differences in scoring. The assignment of four or five digit weights to each question was found to be overkill (Mosley, 1992). Mosley modified the approach by minimising the scoring to three figures and adhering to the STEC's recommended weights for each category. The method was further altered to include a set of between 30 and 100 questions, tailored on the basis of tool type. An evaluation template can therefore include between 170 and 240 questions.

Mosley recognised the fact that, all too often, tool evaluations are completed by the tool's vendor, whom she claimed was someone who (Mosley, 1992):

- just scans the brochures and manuals;
- is unfamiliar with the tool;
- lacks understanding of the user's requirements;
- and performs useless examples which fall short of assessing the tool's functionality;

To avoid such circumstances, the STEC acquired evaluation copies of all the tools being used in a trial evaluation of the method and only used senior practitioners as evaluators – engineers who understood the methods the tools supported, how each tool could be applied to the project, and how each tool would meet the needs of the customer requirements. This established a 'try-before-you-buy' lab at which demonstrations, training, and evaluations could be obtained – a third-party evaluation facility.

By 1990, the STEC had devised a five step process for the use of the evaluation method (Mosley, 1992):

1. classification (of the tool);
2. brief evaluation (how well the tool performed);
3. quantitative assessment (generic set of 140 questions with a tailored set of functional questions to suit the type of tool being evaluated);
4. tailored summary;
5. and a consultation service.

Using this method, a developer evaluates a CASE tool using a template tailored on the basis of the tool's *type*. This evaluation method should be recognised for considering the

way in which a tool fits in with, or supports, the life cycle and associated development methods within an organisation.

2.5.5 Evaluation and Selection of Testing Tools According to Poston and Sexton

Poston and Sexton focussed on evaluation and selection of testing-tools (Poston and Sexton, 1992). They observed that some organisations managed to consistently buy testing-tools which served their software testers and developers well. They attributed this, in part, to the fact that these companies used systematic data-collection methods - many of which were based on forms or checklists - to evaluate tools. They did, however, also acknowledge that data-collection systems which could assist successful tool selection had to be very carefully devised (Poston and Sexton, 1992).

In 1987, a group of experienced tool users and software developers (including Poston and Sexton) came together to develop (by self-confessed unorthodox means) a Reference Model for Computing System-Tool Interconnections - IEEE Standard 1175. Although this was a standard devised to enable tools to communicate, the group realised that it was important to consider tool communication during evaluation. Interconnections affect, and are effected by, how a tool works within an organisation, the platforms it runs on, and how it shares information with other tools. If ignored, aspects of interconnections can thwart successful tool implementation. Subsequently, when the group completed the definition of the standard it produced a spin-off from its work – a testing-tool evaluation system. Poston and Sexton suggest that forms prevent data repetition and ensure that only relevant information is recorded. They therefore adopted a variety of forms as the basis for data-collection when using their evaluation system, including: forms from the IEEE Standard 1175; a needs analysis form; forms for organising evaluation criteria; and forms for recording data about criteria weightings and ratings. Together, these allow the evaluator to (Poston and Sexton, 1992):

- identify and quantify the needs of the company;
- establish tool selection criteria;
- find available tools;
- and select tools and estimate investment costs.

Poston and Sexton sub-divide selection criteria into (Poston and Sexton, 1992):

- general criteria – for example, productivity gain and quality gain;
- environment-dependent criteria – the maximum allowed cost for issues such as: organisational changes for new or revised policies, techniques, training courses etc; platform changes in terms of hardware, operating systems, database systems etc; and tool-interconnection changes;

- tool-dependent functional criteria – the minimum acceptable function in relation to an evaluator-itemised list of tool functions;
- and tool-dependent non-functional (characteristic) criteria – minimum acceptance in terms of performance (response time), human factors such as time to learn to use and time to use, and reliability (mean time between failures).

To best identify these criteria, Poston and Sexton stress the importance of user-needs analysis - they claim that by determining whether perceived needs are actual needs it prevents a variety of mistakes. Each of the above groups of criteria acts as a filter for the subsequent group; only the tools most likely to succeed are considered further (Poston and Sexton, 1992). Each item in each category is given a criteria value (a tolerance threshold) and a weighting factor, and is subsequently awarded an evaluator-assigned rating and a score. The assigned weights are the result of consensus reached among the testers who will be using the selected tool. Weights are assigned according to the rule that each criterion *must* have a weight, and no two criteria may have the same weight. Poston and Sexton highlight the significance of the process of assigning weights to criteria - it brings tool users to an agreement on the relative importance of requirements. In contrast to Mosley - and the developers of other evaluation facilities - Poston and Sexton claim that assigning weights is easy (Poston and Sexton, 1992).

Of Poston and Sextons' many observations, perhaps the most significant (with respect to tool evaluation) are: (1) that, in terms of data-collection mechanisms, forms or checklists were used by those organisations that returned the most effective testing-tool selection results; (2) that a tool cannot be effectively evaluated without considering the context in which it must operate – in their case, considering testing-tool operation within an environmental context defined by maximum cost allowance for operational/structural changes; and (3), that evaluation criteria are not static and uniform across all organisations and situations and so it is necessary to observe the requirements of an organisation.

2.5.6 Evaluation Method for CASE Tools Proposed by Jorgensen, Bygdas, & Lunde

Jorgensen, Bygdas and Lunde examined the use of typical analytical reports, product demonstrations, and opinions as the basis for CASE tool evaluation (Jorgensen et al., 1995). Their investigation identified the following problems (Jorgensen et al., 1995):

1. typical analytical CASE tool evaluations are based on a 'tick-it' philosophy where 'more-is-better' – this is not necessarily the case;
2. tool demonstrations were not designed to give balanced information, but to hide weaknesses and highlight advantages;
3. and subjective assessments are opinions not facts.

These observations led Jorgensen, Bygdas and Lunde to produce a method for evaluating CASE tool efficiency. They claim it can be used in three ways (Jorgensen et al., 1995):

1. use the *method* when evaluating new CASE tools;
2. use the evaluation *results* when selecting between CASE tools for the next development project;
3. and use the evaluation *results* in effort estimation of new projects supported by one of the evaluated CASE tools.

Based on a framework described in 'IEEE Recommended Practice for the Evaluation and Selection of CASE Tools', their method breaks down into five steps (Jorgensen et al., 1995):

1. preparation of an evaluation task definition statement which includes:
   - the purpose of the evaluation;
   - the scope of the evaluation;
   - any critical assumptions and limiting constraints of the evaluation;
   - and data-collection and analysis steps;
2. selection and definition of the evaluation criteria, which should be reviewed to ensure consistency with the task definition statement;
3. selection of the CASE tools to be evaluated;
4. evaluation of the CASE tools relative to the criteria;
5. and reporting of the results.

Jorgensen, Bygdas and Lunde claim that use of their method provides important information for cost/benefit analysis of eventual investment in new CASE tools (Jorgensen et al., 1995). By applying their method, they found that the variance between projects supported by the same CASE tool was much lower than that between projects[22] supported by different CASE tools. They claimed this was evidence that CASE tools had significant systematic impact on the development and maintenance efficiency of projects (Jorgensen et al., 1995).

Jorgensen, Bygdas and Lunde warned that the following limitations should be taken into consideration with respect to their method (Jorgensen et al., 1995):

1. it is an evaluation of the impact of the CASE tools, and not of the skill of the software engineer;
2. they only studied medium to large development projects, and as such, recognised the possibility that the results could be very different if very large projects were examined;

[22] Unfortunately, they do not describe the exact nature of the projects examined – for example, whether they were in the same application domain, or spanned several different domains.

3. and only the efficiency impact of the tools has been examined – not quality, performance, or portability impact for example.

Despite the noted limitations, their systematic method of CASE tool evaluation represents significant progress with respect to accurate CASE tool evaluation. Other than the fact that the measure of efficiency is based upon selection of typical benchmark tasks, little information is available as to how the evaluation is actually performed. It does, however, appear to be flexible such that it can be moulded to meet the needs of a company.

2.6 Grounds for a New UIDT Evaluation Facility ?

Given the UIDT and CASE tool evaluation facilities outlined thus far, what - if any - are the grounds for a new UIDT evaluation and selection methodology? The following sections present a critical discussion of these facilities to address this question.

2.6.1 Examining UIDT Evaluation Facilities

Existing UIDT evaluation facilities fall into three distinct categories:

- checklist-based evaluation (Hix, 1986, Hix et al., 1989, Hix and Schulman, 1991, Hix, 1991, Hix and Ryan, 1992, Sundaram and Ramamurthy, 1996, Valaer and Babb, 1997);
- quality attribute-based; (Bass et al., 1994);
- 'wish-list'-based or abstract evaluation (Myers, 1996).

2.6.1.1 Checklist-Based UIDT Evaluation

Hix *et al* were the founders of checklist-based UIDT evaluation (Hix, 1986, Hix et al., 1989, Hix and Schulman, 1991, Hix, 1991, Hix and Ryan, 1992). When Hix, Tan, and Schulman introduced their checklist-based UIDT evaluation method it broke new ground both conceptually and physically - previously, there was no systematic and structured means of UIDT evaluation. Up to that point researchers had categorised interaction techniques and activities, but made only minimal reference to 'components'[23] that could be used to support the different activities (Foley et al., 1984). More recently, work has been done to categorise user interface components (Myers, 1990a, Ahlberg and Truve, 1995, Card et al., 1990) but it has generally focussed on subsets of these (Myers, 1990b, Myers, 1990a, Ahlberg and Truve, 1995, Card et al., 1990). The Hix *et al* checklists therefore

[23] The term 'component' (or widget) did not really exist until 1987 when object-orientation came to the fore, and so they were not using it in the modern sense.

established a comprehensive categorisation of components relative to user interfaces and their development.

A decade after their inception, these component lists are now dated and incomplete. Lacking the extensibility which would have been necessary to let them to remain contemporary, their value as a means of evaluating current UIDT functionality has been compromised. They have, however, influenced the development of successive UIDT evaluation facilities. Not only do other facilities promote the Hix checklists (Bass et al., 1994), but checklist systems in general have been observed to be the most effective means of data-collection for tool evaluation (Poston and Sexton, 1992). The Hix *et al* checklists are therefore a solid starting point for future checklist-based UIDT evaluation facilities.

The strength of the Hix *et al* approach lies in generic tool comparison - that is, tool comparisons uninfluenced by specific projects or organisations. By advocating a 'more-is-better' perspective of tool evaluation[24], their method is unsympathetic to consideration of project-specific requirements during UIDT tool evaluation. By suggesting that a higher cumulative rating equates to a better tool, the 'more-is-better' philosophy hides the detail of the differences between UIDTs. Since these detailed differences potentially determine the suitability of a UIDT for a given task, this supports Jorgensen, Bygdas, and Lundes' observation that such methods are oftentimes misguided (Jorgensen et al., 1995).

Hix, Tan, and Schulman stipulate that their checklists should only be used by someone who is conversant with their method and is expert in the use of the tools being evaluated. Given the fast pace of modern software development and the relative infrequency with which organisations evaluate tools (see chapter 3), these conditions are impractical. The length and complexity of their checklists and method (see section 2.4.1) place considerable demands on evaluators' time, further exacerbating the impracticability. Hix, Tan and Schulmans' stated criteria for the use of their method effectively establishes it as a third party UIDT evaluation facility - an evaluation process that can only be efficiently performed by commissioned specialist (external) evaluators. The concept of third-party evaluations raises several complex issues. To increase UIDT acceptance it is important to include the anticipated UIDT users in the decision making process (see chapter 1). Although team members may potentially feel marginalised when UIDT evaluation is the responsibility of a delegated project member, this potential may be increased when UIDT evaluation is performed by an external organisation. Yet to be investigated, there are interesting issues relating to trust in the result of a third-party recommendation - this is discussed further in chapter 8.

---

[24] Highlighted by their various ratings.

The Hix *et al* checklists' focus on issues of functionality and usability was criticised by the developers of other UIDT evaluation facilities. However, Mosley's observation that evaluators concentrate on these issues supports the content of the Hix *et al* checklists. The checklists measure functionality in terms of which of the listed components a tool provides; the implementation/specification technique used to access each provided component is recorded. Usability is measured according to whether a function was easy, moderately hard, or hard to perform. Although conceptually the notion of usability deserves its place in the evaluation method, the manner in which it is measured has been shown to lead to ratings that demonstrate evaluator bias. Although inconsequential when the evaluator is the anticipated tool user, it is potentially more serious when the evaluator and tool user are not the same person. Given that the Hix *et al* checklists can only be used by an expert, there is scope for this evaluator bias to be further accentuated[25]. Hix *et al* briefly mention that usability is measured via a series of benchmark tests. However, this is not explained in detail and receives no mention in the actual evaluation methodology. Assuming benchmarks do feature in the evaluation process, they are liable to increase the time required to complete an evaluation and so further worsen its practicability. If not associated with objective measures, benchmarks may heighten evaluator bias due to cross-evaluator differences in experience and ability to perform the tasks and/or to dissimilarities between the evaluator and the anticipated UIDT user. Finally, if the benchmarks are not representative of the project-specific tasks to be performed by the anticipated tool users, they are inappropriate as a measure of quality.

Aside from the categorisation of factors relevant to user interface development, the Hix *et al* UIDT evaluation method gains credibility and strength on the basis that it assesses qualities and attributes of the UIDTs rather than the target-user interfaces developed using the UIDTs. Often idealistic, the pro-UIDT arguments presented in section 2.3.1 fail to acknowledge that the quality of a user interface is heavily influenced by the experience and skill of its developer - the quality can merely be enhanced using a UIDT. Hix and Hartson suggest that the problematic and complex nature of user interface development will not be resolved simply by introducing additional technology (Hix and Hartson, 1993). This is especially true of poorly designed UIDTs which are difficult to use by even experienced user interface developers (see section 2.3.1). By highlighting the hazards of introducing UIDTs into user interface development practice, Hix and Hartson identified the need to consider qualities of a UIDT independently from those of the user interfaces it is used to develop. The quality of the latter - either good or bad - is at least partially dependent on the skill and experience of the developer, and therefore cannot be entirely attributable to

[25] An expert's subjective rating of usability is likely to be considerably higher than a novice.

the UIDT. On this basis, Hix *et al* decided that their evaluation methodology should focus solely on attributes of the UIDT.

When they extended the original Hix *et al* evaluation methodology, Sundaram and Ramamurthy both improved and degraded it (Sundaram and Ramamurthy, 1996). By considering the run-time efficiency of the code produced by the UIDT they removed the UIDT-oriented focus of the original method. Their measurement of run-time efficiency uses factors of user interface and application design that are dependent - at least in part - on developers. By introducing evaluator bias, their measurement does not solely reflect the quality of the UIDT. They have therefore compromised the underlying principle of the original methodology (see above).

Given the influence of learning curve on tool acceptance (Kemerer, 1992), and the fact that it is a direct measure of the design of a UIDT[26], Sundaram and Ramamurthy enhanced the original Hix *et al* evaluation procedure when they extended it to include consideration of learning curve (Sundaram and Ramamurthy, 1996). Although this extension does not compromise the UIDT-oriented focus of evaluation, the means by which they measure learning curve suffers from the same potential evaluator bias as usability - they suggest it can be measured using standardised tests to assess the evaluator's knowledge of each item on the functionality dimension, or by the time required to develop a benchmark application. Learning curve is only one aspect of learnability which, given its situated nature, is only truly meaningful within a given context - see discussion in section 4.2.2.4. Therefore, rather than focus on and measure learning curve in a generic sense, it would be more beneficial to consider the learnability of a UIDT relative to the specific users and task the UIDT is to support. If unrepresentative of the project-specific tasks and if not performed by the anticipated tool users, standardised tests and benchmarks are susceptible to evaluator bias and do not reflect a meaningful measure of UIDT learnability. Although it could be argued that measuring learnability during UIDT evaluation represents an unacceptable additional cost - as well as further reducing the practicability of the method - the significance of learnability in terms of UIDT acceptance justifies its consideration.

Sundaram and Ramamurthy attempted to increase the accuracy with which evaluators could record usability by extending the Hix *et al* three-point usability scale to five points. Not only would this further increase the time required to complete their already lengthy evaluation process, but also - as illustrated by Mosley - such scales are very hard to use and so potentially retard rather than assist an evaluation (Sundaram and Ramamurthy, 1996).

[26] Although characteristics of tool users will vary the speed and ease with which they learn to use a UIDT, learning curve is a direct measure of the UIDT rather than a measure of a developer dependent product of the UIDT.

Finally, the most recent proponents of checklist-based UIDT evaluation are Valaer and Babb (Valaer and Babb, 1997). Although their method does not extend the Hix *et al* checklists, the influence of the latter is reflected in: (1) the hierarchical nature of their criterion-based questions; (2) their inclusion of usability and functionality dimensions; and (3) their method's UIDT-oriented focus. Additionally, their method places greater emphasis on some of the lesser components in the Hix *et al* checklists - for example, portability and flexibility. Valaer and Babb based their method on empirical studies of UIDT selection processes adopted by personnel at Hughes Information Technology Systems. Although the potential unrepresentativeness and lack of random selection of their study population may restrict the applicability of their approach, the similarities between the Hix *et al* checklists and Valaer and Babbs' method strengthens support for the general approach and content of each.

Valaer and Babb deal with functionality in a more abstract and informal manner than Hix *et al*. Although beneficial given that detailed knowledge of each UIDT is not required, important aspects of the UIDTs may be overlooked. For example, by simply asking whether or not a UIDT provides a standard widget library, Valaer and Babb do not specifically determine whether the UIDT supports the subset of widgets required for a specific user interface.

Some of Valaer and Babbs' questions lack specificity with the result that they are open to interpretation. This is aggravated by the lack of structure in which to ask and answer questions. For example, consider their question *'does the tool provide a method to customise widgets?'*: are they referring to customisation of appearance or behaviour?; what do they mean by 'method' - rewriting widget code or using a graphical editor? - and what is considered an acceptable method? Similarly, consider the question *'can the user create prototypes quickly?'*: what kind of prototypes are they referring to?; what counts as 'quickly'?

By abandoning the numerical ratings advocated by Hix *et al* they have limited the 'more-is-better' approach to UIDT evaluation. In so doing, they have introduced a degree of project-specificity to the evaluation process, which is further supported by their stated intention that the method should highlight critical issues to the evaluator and trigger the generation of more application-specific questions. The fundamental aim of their method is to support the selection of the UIDT that contributes most to creating an effective user interface for a specific project. However, since tailorability in their method centres on tool type, the strength of their method is in generic comparison of UIDTs according to type. They mention UIDT selection in relation to a specific application but do not specify which aspects of a tool are most important in this respect. Furthermore, they do not provide the

means to model project requirements or the facility to compare these requirements with actual tool data.

Valaer and Babb suggest that if the introduction of a UIDT would not assist in the achievement of specific project goals, then it lacks worth in that context. They suggest that the first step in any UIDT selection process should be to determine this. Although introducing UIDT technology does not always improve development (Hix and Hartson, 1993) it can be argued that it is potentially hard – if not impossible – to make this decision without having first evaluated candidate tools. It is therefore perhaps more appropriate to consider Valaer and Babbs' suggestion as a caveat rather than as the first step in project-specific selection of a UIDT.

In addition to re-assessing UIDT evaluation, Valaer and Babb present a clear and concise categorisation of UIDTs. Their categorisation has broad coverage - broader than other approaches (see section 2.3) and in fact broader than they themselves claim. Although they suggest that tools for HTML web page construction, Java™ programming, and development of virtual reality environments are beyond the scope of their categorisation this is not necessarily the case. Their definition of 'Application Development Environment' - software that provides capabilities beyond user interface design and development where the additional functionality may be useful in developing the rest of an application - describes each of the tool types they suggest are excluded. Since web pages and virtual reality environments are essentially specialised user interface styles that are developed using specialised implementation techniques and/or toolkits, Valaer and Babbs' UIDT categorisation does in fact cover each of the highlighted tool types.

The use of 'jargon' often makes it difficult to determine the rôle of individual UIDTs. This emphasises the need to provide developers with a clear and concise UIDT classification scheme - not only for choosing tools, but also for determining the applicability of UIDT evaluation methodologies[27]. Although the Valaer and Babb classification has been presented as a means to determine categories of UIDT for evaluation, it could also be used to determine the applicability of UIDT evaluation facilities - given the need to select a particular UIDT type an evaluator could use the classification definitions to determine which of the available UIDT evaluation facilities is most applicable to his evaluation needs.

It could be argued that some UIDTs may not sit comfortably in any single category. In terms of Valaer and Babbs' evaluation process this is insignificant - simply an initial means to restrict the decision space, category choice can be changed and/or tools from other

[27] As discussed in section 2.4, some evaluation methodologies focus on particular UIDT sub-types.

categories included. However, the overlapping nature of their categories may complicate the use of the classification to determine the applicability of evaluation facilities. For example, specialised widget sets - toolkits - are likely to be integrated into 3D visualisation tools; a UIMS may support the development of cross-platform user interfaces. This suggests that rather than being mutually exclusive, Valaer and Babbs' categories have a complex relationship structure (similar to that discussed in sections 2.2.2 and 2.3).

Assuming their definitions, the complexity of Valaer and Babbs' categories can be exploited. First consider the categories: graphical user environments; specialised widgets; plotting and analysis tools; 3D visualisation tools; and GUI porting tools. As defined by Valaer and Babb (see Table 2.2), these tool types would not be able to independently support the *complete* design and development of a user interface. Accordingly, they are typically subsumed by more comprehensive UIDTs (for example, specialised widgets are often included in UIMSs - see section 2.3). Available UIDT evaluation facilities focus on the more comprehensive UIDTs[28] (see section 2.4). To fully exploit the tool classification as a measure of UIDT applicability, the subset of categories listed above should therefore be considered fundamental component parts of the remaining categories (see Table 2.2).

Now consider the remaining, more comprehensive, categories: application development environments; GUI builders; UIMSs; and cross-platform development tools. In principle, Valaer and Babbs' definitions of application development environments, GUI builders, and UIMSs agree with those discussed in section 2.3. GUI builders (or Interface Development Tools) enable a developer to compose a user interface out of a set of predefined components, to define new components, and to generate code from the design. UIMS defines a category of tools that, at a minimum, expand on GUI builders to additionally support definition and execution of the functionality of the interface - often via prototypes. Although application development environments are generally considered to be a specialisation of UIMS (see section 2.3), they extend the minimal functionality of UIMSs to include support for development of the remainder of an application, and may or may not include facility for cross-platform development. Valaer and Babb have isolated Application Development Environment on account of the extensive functionality of these tools. On the basis of these generally accepted definitions, one category can be considered an extension of another. They can therefore be hierarchically structured as shown in Figure 2.8. UIDTs for which there are available evaluation facilities fall into one of these four categories. Given the hierarchically inclusive nature of the categories, UIDT

[28] Although UIDT evaluation facilities are targeted at more comprehensive tools, subsections of them could be used to evaluate the components outlined here.

evaluation methods can therefore be said to apply to tools in their target category, and to tools in the sub-categories illustrated in Figure 2.8

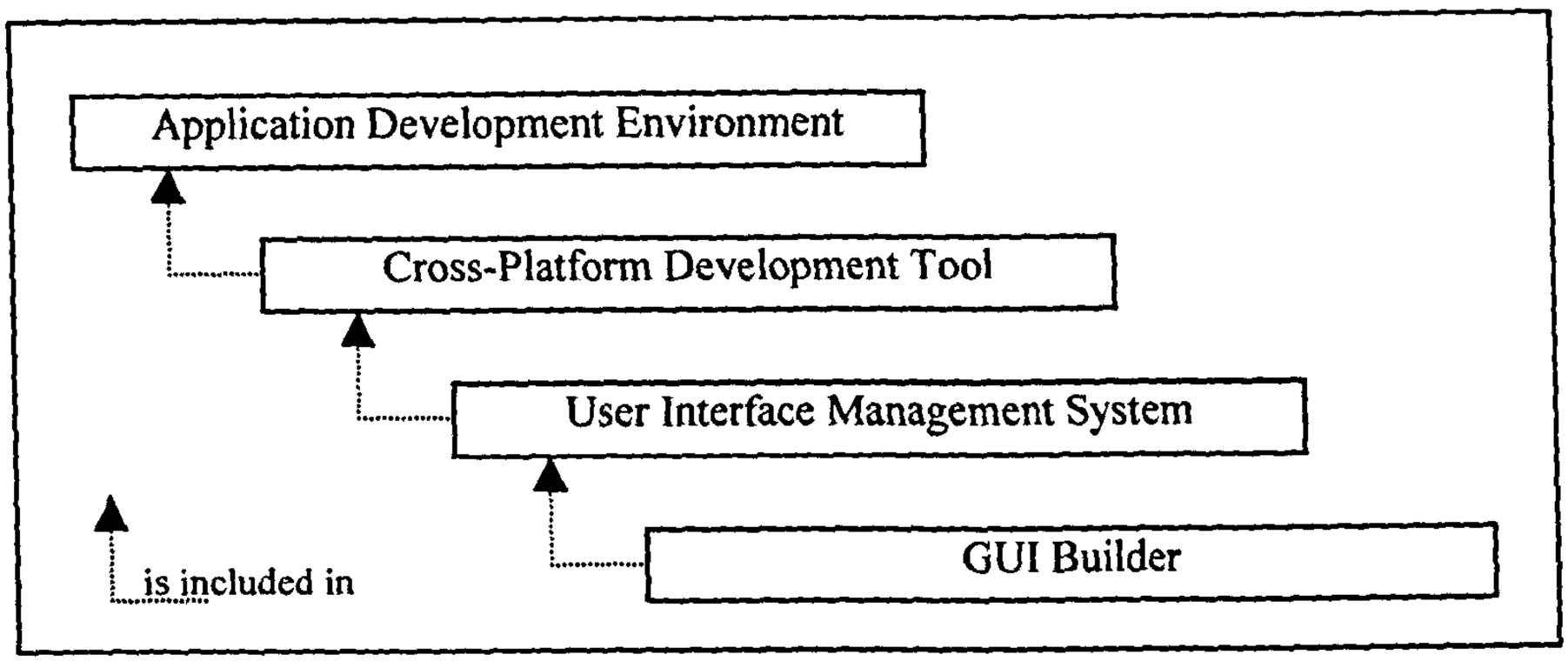


Figure 2.8 - *hierarchical relationships between GUI development tool categories*

2.6.1.2 UIDT Evaluation Based on Attributes of Quality

Differing from the others, Bass, Abowd, and Kazmans' evaluation facility considers quality attributes of the UIDT and of the target system (Bass et al., 1994). They heavily criticised the Hix *et al* checklists for focusing on functionality. Although it is important to measure more than just the functionality of UIDTs, Mosley's observation of its importance (Mosley, 1992) suggests that this focus is not as disadvantageous as Bass, Abowd, and Kazman imply. This highlights a potential conflict between the larger body of UIDT evaluation research and the method proposed by Bass, Abowd, and Kazman. Although, in this section much is argued against the approach taken by Bass, Abowd, and Kazman, later discussion in this chapter, and in chapter 4, suggests that it might be possible to resolve this conflict by adopting some of their principles but applying and/or measuring them in a different way.

The quality attributes listed by Bass, Abowd, and Kazman are biased towards the target system – they are essentially suggesting that good output equates to a good tool (and vice versa). Although it can be argued that the provision of certain functionality in UIDTs is more likely to result in better quality user interfaces[29], designer differences may lead to potentially very different output in terms of its quality. Their putative correlation will therefore only work if there was some investigation to rule out the effect of designer differences. They do not suggest that such investigation has been carried out, but do make some effort to compensate for evaluator differences when evaluating these tools and their output.

[29] For example, the use of certain user interface styles is often held to correspond to more usable user interfaces - if a UIDT supports such a style, there is greater potential that its use will lead to the construction of usable user interfaces.

Bass, Abowd, and Kazman identify four different evaluation techniques to be used according to the quality attribute being measured. These are: feature inspection; architectural inspection; direct developer input; and benchmarking. They claim that the first two are objective measures, and that the latter two are subjective. Relying most heavily on feature inspection and architectural inspection, Bass, Abowd, and Kazman have attempted to remove evaluator bias as far as possible. Nevertheless, the results obtained using these techniques are subject to issues of trust based on an evaluator's ability to identify features in unfamiliar tools. This issue of trust and the manner in which it can be tackled is discussed further in chapter 8. Furthermore, feature inspections do not directly reflect poor design of the UIDTs' user interfaces. Bass, Abowd, and Kazman use benchmarks to assess construction efficiency of the tool and modifiability of the target code. Acknowledging evaluator differences, they suggest that several representative evaluators perform the same series of tasks to allow the results to be generalised. Unlike other methods using benchmarks, Bass, Abowd, and Kazman suggest that the benchmarks be devised such that they are representative of the actual tasks that will be demanded of the UIDT, and that they be performed by a subset of the developers who will be using the selected UIDT. In so doing, Bass, Abowd, and Kazman have introduced a degree of context sensitivity to their evaluation process which is absent from the use of benchmarks in the others. However, requiring that several developers all perform several tasks increases the impracticality of UIDT evaluation.

Bass, Abowd and Kazman include a rating system for each of their listed quality attributes, in which individual ratings are weighted to achieve an overall rating. Although this demonstrates an attempt to allow consideration of the relative importance of quality attributes within a given context, it promotes the 'more-is-better' approach to UIDT evaluation and can be subject to the prohibitive costs associated with ratings as discovered by Mosley (Mosley, 1992, Mosley, 1995).

Bass, Abowd, and Kazman correlate the usability of the target system with the interaction styles provided by the UIDT, which they then inspect. Despite the identified importance of the usability of a UIDT (Mosley, 1992) and its influence on UIDT acceptance (Kemerer, 1992, Sundaram and Ramamurthy, 1996), they do not measure the usability or learnability of the UIDT itself. They therefore run the risk that up to 70% of a tool's actual functionality may not be realised (Kemerer, 1992), including the interaction styles required to produce good target-user interfaces.

Bass, Abowd and Kazman suggest that compatibility should be measured in terms of the compatibility of the target-user interface within the complete target application, and with respect to the UIDT within the development environment. Both are assessed using feature

inspection: the first according to language and platform compatibility; the second according to the other software support in the development environment. In the case of environmental compatibility, Bass, Abowd, and Kazman overlook the influence of working practice, development life cycle, and the actual project team members who will ultimately determine the UIDTs acceptance.

Bass, Abowd, and Kazmans' method supports the evaluation of interface builders, and focuses on quality attributes of the software architecture of the tool and of the user interfaces produced using the tool. Although it initially seems that their method is therefore restricted, they indirectly consider functionality that would be provided in more comprehensive UIDTs . For example, when measuring UIDT compatibility, they consider how it might integrate with the likes of compilers. In essence, therefore, they treat more comprehensive UIDT functionality as aspects of external software.

Bass, Abowd and Kazman make a number of fundamental assumptions in support of their method (Bass et al., 1994). They claim that the set of attributes is universal and fixed. The attributes are indeed common in software engineering, but it can be argued that they are neither universal, nor is their importance universal. Although all six attributes *can* be applied universally, their applicability may not be the same in every scenario – for example, when faced with tight schedules, construction efficiency may be more important than reusability; when prototyping, modifiability may be more important than response time. Bass, Abowd and Kazman assign their quality attributes to either the UIDT or the target-user interface. They appear to have done this somewhat arbitrarily; in particular, it can be argued that attributes assigned to the target-user interface are also relevant to the UIDT. For example, Bass, Abowd and Kazman focus on modifiability of the target-user interface – its extension and adaptation. Issues of modifiability are equally important with respect to the UIDT itself – for example, can a developer modify its user interface or functionality? If their quality attributes were applied to both the UIDT and target-user interface, the restrictive nature of their method (see above) would be lessened, making their method more compatible with existing UIDT evaluation facilities.

2.6.1.3 Wish-List Based or Abstract UIDT Evaluation

Informally, Myers suggests goals that might be expected of a UIDT - a 'wish-list' - and how tools might satisfy these goals (Myers, 1996). Although its lack of structure avoids the 'more-is-better' perspective (and associated numerical ratings), it nevertheless has a number of potential drawbacks. Several of the suggested goals are so abstract that individual evaluators could potentially interpret how they might be met in different ways. For example, the listed goal *'facility for a designer to rapidly investigate different designs'* leaves open to question what constitutes 'rapid'?, what is meant by 'investigate' - is it just

the designer visualising different ideas or is it a more structured end-user-based evaluation of design solutions?, and at what level are the 'designs' - just interface mock ups or more comprehensive prototypes? A further example - *'ability to create easy-to-use interfaces'* - is not only open to intra-evaluator interpretation (what constitutes 'easy-to-use'?) but also illustrates potential for evaluator bias[30] which will be reflected in their opinions of different tools.

Lacking structure, appearing *ad hoc*, and being incomplete, Myers' component list is perhaps appropriate for initial discussion of UIDTs. It is, however, inadequate to support and inform systematic comparison of detailed UIDT data with respect to a project-specific context. Myers' goals are therefore best regarded as triggers or guides for the use of alternative, structured UIDT evaluation facilities.

2.6.2 Examining CASE Tool Evaluation Methods

Although the preceding discussion has outlined the shortcomings of existing UIDT evaluation facilities and thereby highlighted scope for the development of a new facility, one issue remains to be clarified: why could evaluators not just use a CASE tool evaluation facility to evaluate UIDTs? By closer examination of available CASE tool evaluation facilities, this section addresses this question.

Defined as software tools which aid software engineering activities - including requirements analysis/capture, software design, code production, testing, document generation, quality assurance, configuration management, and project management (Jorgensen et al., 1995) - CASE tools can be considered to include UIDTs. To support the inherent complexity of user interface development (see chapter 3) UIDTs are themselves very complex and varied (see section 2.3). To accurately evaluate UIDTs - especially for use in a specific project - it is therefore essential to consider the detail of the tools. Existing CASE tool evaluation facilities (see section 2.5.2) - some of which focus on specific CASE tool types (see section 2.5.5) - do not support examination of UIDTs at the necessary level of detail[31]. Therefore, although UIDTs could in principle be evaluated using the appropriate parts of a CASE tool evaluation facility, an evaluation of this nature would be an insufficient basis for a detailed and meaningful UIDT comparison. Alternatively, this argument can be approached from a different perspective: if UIDTs are considered a specialised CASE tool, UIDT evaluation techniques could be considered specialised CASE tool evaluation facilities - similar to Poston and Sextons' specialised

[30] The skill level and measure of learnability will differ for each evaluator, thus influencing their ability to use each tool to produce 'easy-to-use' user interfaces.

[31] As listed in the Hix *et al* checklists for example.

testing-tool evaluation method (see section 2.5.5). In either case, the need to use dedicated UIDT evaluation facilities for UIDT-specific evaluation is clear. Even ignoring the suggested extrapolation of UIDTs, the principles embedded in existing CASE tool evaluation facilities do not meet the needs of UIDT evaluation, as discussed in the remainder of this section.

Using the models and associated question-based checklists developed by Vessey, Jarvenpaa, and Tractinsky, an evaluator can assess CASE tools in terms of their adopted product control and/or validity procedures and their support for collaborative systems analysis and design activities (see section 2.5.3). However, even in combination, their evaluation facilities fail to provide adequate support for accurate and effective CASE tool selection. As mechanisms by which to conduct *generic* analyses of the product control/checking and collaborative facilities in CASE tools, their models and checklists represent significant progress. However, given that Vessey, Jarvenpaa, and Tractinsky appear to have intended that their checklists be used to compare CASE tools for the purpose of categorisation as opposed to selection, it is possible to identify a number of shortcomings regarding their applicability as *selection* mechanisms for CASE tools. The questions listed by Vessey, Jarvenpaa, and Tractinsky focus on the functionality of CASE tools as far as that which is embedded in the control/checking mechanisms and collaborative facilities the tools support. In so doing, their checklists adopt an intentionally narrow perspective of CASE tool functionality which precludes them from supporting comprehensive evaluation of CASE tool functionality.

The checklists do not explicitly support tailorability, nor do Vessey, Jarvenpaa, and Tractinsky promote contextualisation of the evaluation process. As a result, their checklists do not support project-specific CASE tool evaluation and are remote from consideration of the environmental context in which a CASE tool is to be deployed; the strength of their checklists is therefore restricted to *generic* CASE tool comparison. The questions listed by Vessey, Jarvenpaa, and Tractinsky are abstract to the extent that they consider functionality at the level of *whether* it is possible rather than *how* it is achieved. The resulting perspective of a CASE tool evaluated using their checklists is therefore lacking this level of detail which, as is discussed in detail in chapter 4, is relevant to context-sensitive tool evaluation. Given the high-level or abstract nature of the checklists, their questions are open to interpretation. For example, consider the question *'can the tool automatically generate a first cut structure chart from the data flow diagrams?'*: what constitutes automatic generation?; what is acceptable as a 'first cut' structure chart? The models and checklists defined by Vessey, Jarvenpaa, and Tractinsky are specifically aimed at CASE tools which support structured systems analysis and design; they are therefore inappropriate for other specialised CASE tools such as testing tools. Finally, with respect

to their evaluation of collaborative support in systems analysis and design CASE tools, Vessey, Jarvenpaa, and Tractinsky suggest that, given absent functionality, it is often necessary for developers to integrate use of third party software (for example, email tools) with their use of these CASE tools. Despite this observation, however, their evaluation mechanism makes no provision for capturing this element of successful CASE tool use.

Using Mosley's five step CASE tool evaluation methodology an evaluator assesses tools on the basis of a template tailored to the appropriate CASE tool type (see section 2.5.4). This method was the first of its kind to emphasise tailorability and to seriously consider the way in which a tool must support the software life cycle and development strategies within an organisation. In spite of this, however, it presents a number of drawbacks for both CASE tool and UIDT evaluation. Consider the six categories of questions: ease-of-use; power; robustness; functionality; ease of insertion; and quality of support. Although they permit increased scope and flexibility in tool evaluation, their associated questions are potentially open to differences in interpretation. The effects of evaluator variance are further exacerbated by the lack of a consistent structure for recording evaluation data.

Despite considering the project-specific development life cycle into which the selected tool is to be integrated, Mosley does not consider two of the fundamental factors that contribute to, and influence, its integration - learnability and project-specific requirements[32] (Kemerer, 1992, Sundaram and Ramamurthy, 1996).

Highlighting the need to allow evaluators control over the priority of evaluated components, Mosley implemented a very detailed scoring criteria and weighting facility which evaluators could apply to each question. Too time consuming and complex, these were considered impractical. They did, however, illustrate that evaluators assign greatest priority to issues of functionality and usability when comparing CASE tools. Although other tool evaluation facilities include functionality and usability, Mosley's study provided the evidence necessary to justify their inclusion.

Designed to support the evaluation of a wide range of CASE tool types, Mosley's questions are necessarily abstract. Despite having identified the significance of functionality data, the questions do not permit an evaluator to compare the low-level functional differences between tools. Fundamental points upon which a selection decision is likely to be based may therefore potentially be overlooked. However, if the goal of an evaluation is to complete an initial generic comparison of tools - according to type - on the basis of a series of high-level responses, Mosleys' abstract questions are more than adequate.

[32] For fuller discussion see chapters 1 and 4.

Essentially a third-party evaluation facility, the STEC CASE tool evaluation methodology is accessed via their 'try-before-you-buy' lab. Until investigated, unresolved issues regarding trust in third-party evaluation leave the general effectiveness of such methods open to question (see section 2.6.1.1 and discussion chapter 8).

By studying organisations' selection procedures for testing-tools, Poston and Sexton determined that some were more effective than others. After noticing that the organisations with better success rates used checklists to structure their selection procedures, Poston and Sexton concluded that CASE tool evaluation needed to be structured. Having spent time observing and assessing CASE tool evaluation, Poston and Sexton were in a better position than most to determine what would be needed in a new CASE tool evaluation methodology. However, they based their method on some overly simplified assumptions. Based on their observed correlation between the success of tool selection and the use of checklists, they assumed that the success of an evaluation - in terms of the degree to which the selected tool met identified requirements - was due to the evaluation mechanism. Although their observation strengthens support for checklist-based evaluation techniques, their assumption overlooks the possibility that companies seen to be successful in terms of tool selection may have reviewed candidate tools relative to the context in which the selected tool was to be used. It is therefore possible that it is this facet of their process - irrespective of the use of checklists - that is the deciding factor in terms of their success. However, given that evaluators have been shown to focus on functionality and usability and that checklist-based systems permit greater and more structured attention to detail, the true cause of the observed success is likely to be a complex combination of both the means of, and underlying approach to, tool evaluation.

Poston and Sexton sub-divide selection criteria into four groups; each group acts as a filter to the subsequent group and each criterion is assigned a tolerance or bounding value[33] and a weight. One of these groups is concerned with Tool Dependent Non-Functional issues - for example, the time taken to learn to use a tool. Although potentially relevant, the usefulness of the factors in this group may be restricted if their values are set without reference to the expected context of use. By failing to promote or support such context-sensitive consideration, Poston and Sexton consign the strength of their method to generic tool comparison. Using this method, each examined criterion is rated and awarded a score - a product of the evaluator-assigned rating and the weight. The scores are then totalled - the tool with the highest total score is ultimately recommended. Poston and Sexton therefore promote the 'more-is-better' approach, arguments against which have already been presented.

[33] For example, the maximum cost that can be outlayed for hardware.

Placing strong emphasis on cost, the majority of the features in this method are oriented towards the financial implications of the evaluated tools. Although it is important, cost is not the principal consideration when *evaluating* tools - it is an assumption which limits the decision space (see chapter 1). If financial concerns are allowed to dominate tool evaluation there is a significant risk that a tool may be selected that is perhaps least costly, but is also least appropriate with respect to its intended use.

Poston and Sextons' use of checklists is somewhat inconsistent: they support detailed examination of some features whereas they treat others more abstractly, bordering on 'wish-lists'. Although there are advantages to being able to view data at different levels of abstraction, the evaluator should be allowed to make that choice to reflect the context of his evaluation.

By forcing evaluators to rely heavily on vendor-supplied information about tools, this method introduces considerable scope for evaluator bias: evaluators will be better able to validate vendor-supplied data for tools with which they are familiar than for tools they have not experienced. That said, the variable ability of evaluators to validate tool data is a factor that is likely to influence all tool evaluation facilities. Given individuality of evaluators, it may not be possible to eliminate this issue of bias - evaluators are always going to be able to validate data for tools they know better than for tools they don't know. However, if the tool data used during evaluations was to represent a consensus of expert opinion, it may be possible to increase evaluators' trust in data for tools with which they are unfamiliar. Similarly, by improving the quality and supporting documentation of selection decisions, it may be possible to increase the trustworthiness of decisions themselves. This thesis develops an evaluation method that establishes the potential to tackle the issue of trust - chapter 8 discusses the way in which this potential can be exploited to implement the solutions suggested here.

Devised specifically for the evaluation of testing-tools, the component lists in the Poston and Sexton checklists can be extended to include additional factors relevant to testing-tool evaluation. Although certain of their criteria are applicable to other CASE tool types, their method would, in general, appear to be restricted to evaluation of testing-tools.

On the basis of identified problems with CASE tool evaluation which relied on analytical reports, product demonstrations, and the opinions of tool users, Jorgensen, Bygdas and Lunde developed the first CASE tool evaluation method to focus on the actual situation for which a tool is being selected - see section 2.5.6 (Jorgensen et al., 1995). Suggesting that their method evaluates the impact of the CASE tools and not the skill of the evaluator - noted to be a serious drawback of other evaluation methods - they claim that using their method provides important information for cost/benefit analysis regarding CASE tool

investment. A significant advancement in CASE tool evaluation, the validity of their method is supported by the fact that it is based on observation of the failings in other methods[34]. However, by focussing on the efficiency impact of CASE tools to the exclusion of other important issues involved in the adoption of a CASE tool, their method is somewhat restricted.

2.7 Conclusions

Exploring the nature and importance of UIDTs, the initial sections of this chapter emphasised the number and variety of such tools and highlighted the importance of effective, structured evaluation when choosing between them. All the existing evaluation facilities for UIDTs, and some which exist for CASE tools - were then presented and analysed. The strengths and weaknesses of the various evaluation facilities were highlighted to emphasise the grounds for the development of a further UIDT evaluation methodology – and hence the work in this thesis. Drawing together the main points that have arisen during the central discussions, this chapter will conclude by identifying an informal specification of the requirements of a new UIDT evaluation methodology.

2.7.1 The Mechanism

Many of the evaluation facilities outlined in this chapter made a clear distinction between the *mechanism* for data-collection and the evaluation *methodology* (for example, Bass et al., 1994). The importance of structured data gathering was identified by Poston and Sexton (Poston and Sexton, 1992), and was stressed by Valaer and Babb when they found the absence of structure hindered their ability to record evaluation data and therefore effectively evaluate tools (Valaer and Babb, 1997). Observation of testing-tool evaluation led Poston and Sexton to suggest checklists as the most effective means by which to gather evaluation data (Poston and Sexton, 1992); in varying forms, checklists were the mechanisms most widely adopted by the evaluation methods discussed in this chapter (Bass et al., 1994, Hix and Ryan, 1992, Hix and Schulman, 1991, Hix et al., 1989, Mosley, 1992, Poston and Sexton, 1992, Sundaram and Ramamurthy, 1996, Valaer and Babb, 1997).

Of the checklists mentioned, those devised by Hix *et al* establish the most comprehensive categorisation of components relative to user interfaces and their development (Hix and Ryan, 1992, Hix and Schulman, 1991, Hix et al., 1989). Due to the speed of progress in the software industry, the lack of extensibility within the Hix *et al* checklists rendered them

[34] They do, however, use benchmark tests - arguments against which have already been presented in earlier sections of this chapter.

dated soon after their completion. Despite this, the composition of the Hix *et al* checklists and the published support for checklist-based data gathering techniques suggest that (at least initially) checklists are the most appropriate data gathering mechanism for use in a future UIDT evaluation methodology - assuming that their extensibility is assured as a matter of priority. Since the issue of data gathering is not the principal concern of this research, checklists are not however being advocated as the most effective means of data-collection. Checklists are used in this research on the grounds of observation and because the evaluation facility developed in this thesis uses the structure of the Hix *et al* component categorisation as a starting point (see chapter 4). The issue of effective data-collection techniques is discussed in chapter 8.

2.7.2 The Criteria and Measures

The evaluation facilities outlined in this chapter represent different subsets of criteria for consideration during tool selection (Bass et al., 1994, Hix and Ryan, 1992, Hix and Schulman, 1991, Hix et al., 1989, Mosley, 1992, Poston and Sexton, 1992, Sundaram and Ramamurthy, 1996, Valaer and Babb, 1997). These range from basic functionality and usability, to concerns about attributes of quality. They concern the developer and the end-user of the target-user interface, and relate to both the UIDT itself and the target-user interfaces produced using the UIDT. Where different facilities agree on the criteria, they often disagree on the means by which the criteria should be measured (for example, Bass et al., 1994 and Sundaram and Ramamurthy, 1996). Given this diversity, it could be argued that it is difficult to determine which criteria and dimensions of measurement should be included in a future UIDT evaluation methodology. However, highlighting the problems associated with each facility in terms of their selected criteria and approaches to measurement, the discussion in section 2.6 provides a basis for deciding which criteria and dimensions of measurement to adopt in the UIDT evaluation facility central to this thesis.

Considered most important (Mosley, 1992), criteria relating to functionality and usability were common to most evaluation facilities. Access to functionality is the principal reason for selecting UIDTs, and is consistently measured according to whether a tool supports listed features. The proposed UIDT evaluation methodology must therefore list criteria concerning the functionality provided by a UIDT; to accommodate advances in user interface technology this list must be sufficiently extensible. To have broad applicability and to remain viable, the new UIDT evaluation methodology should include functionality which spans the hierarchy of UIDT sub-types (see section 2.6.1.1). Although they at times differed, the listed features for UIDTs normally originated in the Hix *et al* checklists, or indeed made direct reference to them (for example, Bass et al., 1994). Given their comprehensiveness (see section 2.6.1.1) the Hix *et al* checklists represent a valid starting

point for the development of component lists within the proposed UIDT evaluation methodology.

Although ubiquitous, usability was used inconsistently in terms of its scope and means of measurement. For example, Bass *et al* consider usability as a measure of the target-user interface and measure it in terms of the interaction styles provided by the UIDT; Hix *et al* focus on the usability of the UIDT itself and measure it according to the ease with which individual components can be accessed (Bass et al., 1992, Hix and Ryan, 1992, Hix and Schulman, 1991, Hix et al., 1989). This difference highlights a number of important related issues which need clarification for the new UIDT evaluation methodology. Firstly, there is the issue of *which system should be investigated.* Much of section 2.6 was concerned with the question of whether the target-user interface should be measured during evaluation of UIDTs. Due to the potential impact of developer bias, it was concluded that only issues concerned directly with the UIDTs themselves should be evaluated. Secondly, there is the issue of *what constitutes usability.* As discussed in detail in chapter 4, usability is a complex measure of many constituent factors including learnability - a measure of (amongst other factors) the familiarity and generalisability of a user interface. Since user interface styles contribute to individual users' assessments of familiarity and generalisability, a measure of interface style - focussing on the UIDT itself - should perhaps be considered in the new evaluation methodology. Finally, there is the issue of *how usability should be measured.* Several methods use an evaluator-assigned ease-of-use rating to measure usability. Section 2.6.1.1 argued against this approach on the grounds of evaluator bias.

As indicated above usability is closely linked to learnability. Dependent on the person, task, and environment, if it is to be meaningful the learnability of a UIDT must be measured with respect to the context in which the tool is to be used (see chapter 4). Focussing on the learning curve[35], assessment of learnability in existing evaluation methodologies relies heavily on the skills of the evaluator to perform potentially unrepresentative benchmark tasks (Sundaram and Ramamurthy, 1996). Such measures therefore have little meaning. The importance of learnability with respect to UIDT acceptance (Kemerer, 1992) suggests that it is essential that learnability be included in future evaluation methods. Given their associations, learnability and usability are perhaps best measured together. The new evaluation methodology must therefore consider joint measures of usability and learnability which focus on their situated nature and their relationship with UIDT acceptance (see chapter 4).

[35] Only one aspect of learnability - see section 2.6.1.1.

The remaining UIDT evaluation criteria are – perhaps undeservedly – restricted to the target-user interface produced with the UIDT (see section 2.6.1.2). It has been suggested that if these attributes were also focussed on the UIDT it would be possible to comprehensively consider any UIDT type during evaluation. How best to introduce their strengths into the new UIDT evaluation methodology is therefore tackled in chapter 4.

In the majority of evaluation methods, the evaluator is confined to recording only data which is prescribed by the method - important data is therefore often omitted (see chapter 5). The variety of circumstances of independent evaluations highlights a very complex problem regarding the creation of a fully adequate descriptive model. Consider briefly, the options. Although unlikely, it might be possible to develop an exhaustive framework of all components and criteria that might ever be relevant to future UIDT evaluation. If feasible, the potential size of such a framework would inevitably be so impracticable as to render it useless[36]. Somewhere between this and the existing restricted evaluation structures is the notion of a flexible framework for UIDT evaluation. Encompassing an initial defined set of components and criteria, such frameworks might support flexibility and tailorability of component lists, or they might provide the facility to record 'miscellaneous' or un-categorised data - they might even support both. Given the suggested versatility of flexible frameworks, they should be able to cater to unusual circumstances and therefore provide the structure for a more adequate descriptive model. Flexibility should therefore be considered for the new UIDT evaluation methodology.

2.7.3 Assigning Importance to Criteria

The discussion in this chapter suggests that the key to successful, meaningful, UIDT evaluation is *tailorability*. In terms of both learnability and functionality, the ability to tailor the associated criteria to factors of concern within a given context is paramount if the results are to be useful during the selection process. This therefore raises the question of *how should tailorability be achieved.*

Several evaluation methods use weightings to prioritise evaluation criteria (Bass et al., 1994, Hix and Ryan, 1992, Hix and Schulman, 1991, Hix et al., 1989, Mosley, 1992, Poston and Sexton, 1992, Sundaram and Ramamurthy, 1996). Although their intention is to introduce tailorability, weightings usually evoke a 'more-is-better' approach which is exacerbated when the methods promote the generation of numerical ratings (for example, Hix and Ryan, 1992, Hix and Schulman, 1991, Hix et al., 1989). Although they have value in the case of generic tool evaluations, numerical ratings are of little value when it is

[36] Keeping it up to date would also be potentially impracticable.

necessary to consider project-specific requirements - in such cases it is the set of relevant components and their associated learnability that determine how well a specific UIDT meets project-specific needs. Project-specific issues aside, several of the evaluation methods identified problems with using weights - thus reducing their usefulness as tailoring mechanisms. For any given project, tailorability must therefore focus on identifying the set of relevant functional components and observing contextual issues regarding the anticipated tool users. On this basis it is more likely that a UIDT will be selected that meets the users' needs and is therefore accepted.

UIDT selection is a complex process governed by numerous, often unpredictable factors. It is therefore not possible to pre-judge precisely how evaluators want, or need, to compare their evaluation results. A UIDT evaluation methodology should therefore enable an evaluator to identify those issues which are of importance, and to compare them with reference to the context from which they acquired their importance. Rather than generate a recommendation (or numerical rating) which *cannot* cater for all potential circumstances and which hides much useful information, UIDT evaluation facilities should simply record and present evaluation data. If possible, they should also support data comparison and analysis so that an evaluator may make an informed, and contextually aware, tool selection.

Existing evaluation methods adopt different levels of abstraction when considering evaluation criteria – for example, the Hix *et al* checklists consider low-level details whereas Valaer and Babb adopted a higher level perspective (Hix and Ryan, 1992, Hix and Schulman, 1991, Hix et al., 1989, Valaer and Babb, 1997). Given that the level of abstraction at which the evaluator can, or wants, to work is not pre-determinable, the new UIDT evaluation methodology should not prescribe the level at which evaluation criteria must be considered; it should leave the decision of abstraction to the developer, and thus must be structured accordingly.

2.7.4 Resourcing an Evaluation

When deciding to perform a UIDT evaluation, two resource-related elements should be considered. The first of these is *time and effort*. Given the complexity of UIDTs and the development processes they are designed to support, accurate and effective UIDT evaluation cannot be made a 'quick' procedure *per se*[37]. This is reflected in the existing evaluation facilities which, by their own admission, are time consuming and often complex. To rush an evaluation would inevitably lessen its validity, increase the chance

[37] This thesis does, however, present mechanisms for streamlining evaluations and reusing evaluation results in order to reduce the time taken to complete UIDT evaluation whilst maintaining evaluation quality.

of selecting a UIDT ill suited to the needs of a given project, and subsequently waste time and money. The decision to formally evaluate UIDTs should not be taken lightly; it needs to address the balance between the cost of performing the evaluation against its potential benefit. Although the quality and effectiveness of the UIDT selection decision contributes to cost-benefit analysis, this research is not concerned with this aspect of UIDT evaluation.

The second resource-related element of UIDT evaluation concerns the *source of evaluation information*. There are two aspects to information source: *who will be performing the evaluation*; and *where will the data about the tools come from*. First consider the person performing the evaluation. An evaluator can be an independent third-party, a designated member of the project team who may or may not be using the selected tool, or an individual selecting a tool for his own use. In the first two instances - especially where the evaluator is an independent body - there are unresolved issues regarding the relationship between trust in the UIDT recommendation and confidence in the evaluator's ability. Briefly discussed in this chapter, mechanisms to increase this issue of trust are discussed in detail in chapter 8. The basis of these mechanisms is already in place in the UIDT evaluation methodology presented in this thesis (see chapters 4 onwards).

The second aspect of information source concerns the actual tool data. It can be obtained from a number of sources including vendor information, and 'hands-on' use of the UIDTs themselves. Obviously, there is cost/benefit analysis[38] involved when deciding whether to trust third-party data or to invest the time and effort in direct examination of the actual tools. Despite potential mistrust of vendor-supplied data (Mosley, 1992), the time taken to learn to use a tool sufficiently to record associated data with confidence (Hix and Ryan, 1992, Hix and Schulman, 1991, Hix et al., 1989) may force evaluators using existing evaluation facilities to consider this option. As with third-party evaluations, there are issues of trust in the quality of UIDT evaluation data based on its origin (see section 2.6.2). Chapter 8 discusses proposals for collaborative UIDT data sources based on the structure and software support for the evaluation methodology presented in this thesis. By adopting models of peer-review used successfully in other fields, it is hoped that this will increase the trustworthiness of UIDT evaluation data.

2.8 Chapter Summary

After introducing the concept and categorisation of user interface development tools, this chapter discussed their potential significance, or impact, within the software development process. The various existing user UIDT evaluation methodologies were presented, and

[38] Again, beyond the scope of this thesis.

their strengths and weaknesses were analysed. This analysis highlighted the need for another UIDT evaluation methodology and, together with the discussion in the concluding sections, identified the requirements which must be met by a new UIDT evaluation methodology if it is to combat the failings of the existing methods, and provide software developers with an effective and useful evaluation facility. The remainder of this research presents a new, contextually sensitive, UIDT evaluation methodology and associated software support, designed to tackle the issues identified in this chapter.

CHAPTER 3 : INDUSTRIAL SURVEY

3.1 Introduction

Before developing a new UIDT evaluation methodology, it is important to examine *how* and *why* software developers currently choose their UIDTs. By observing (in context) current evaluative processes and the criteria and restraints imposed upon UIDT selection, it is possible to develop a picture of the context and manner in which UIDTs are selected and used. This data can then be used to inform the development of a new UIDT evaluation facility that permits the description of project-specific context of use including the functional requirements demanded of a UIDT.

This chapter presents the findings of an industrial survey which was performed primarily to investigate the selection and subsequent use of UIDTs. In order to provide insight into the developmental context of use into which selected UIDTs were placed – and hence, the criteria to which a UIDT must be sympathetic to better achieve acceptance – the survey also examined the user interface development processes adopted by the organisations using such tools.

This chapter begins by outlining the manner in which the survey was conducted. After presenting the detailed findings of the survey with respect to application characteristics and software engineering administration, it looks at the practical user interface development processes followed within industrial contexts. This is followed by a discussion of the findings regarding the selection and use of user interface development software. The chapter closes with a discussion of the apparent relationships between the observed factors, and conclusions which can be drawn from the findings with respect to the development of a new UIDT evaluation methodology.

3.2 Conducting the Survey

The principal objective of this survey was to investigate within an industrial context: *how* and *why* user interface developers currently choose their UIDTs; whether they are subsequently satisfied with their chosen software; and, having made real use of their development tools, the extent to which available functionality is used. An additional aim of the survey was to examine the user interface development processes adopted by the software developers in order to consider the developmental context into which developers

expect to integrate their selected tools. It was hypothesised that the software engineering methodology employed within an organisation would influence the company's choice of tools – that factors such as team structures and practical design/engineering processes, together with the particular needs of the application domain for which software was being developed, would form the requirements base for the selection of UIDTs. It was further hypothesised that the absence of appropriate tools that could synthesise with its existing development process may lead an organisation to alter its developmental strategy in order to accommodate effective UIDT integration and use. However, this final question was considered beyond the scope of this survey, and more appropriate as an issue for future investigation. This section will explain how the survey was performed.

3.2.1 The Mechanism

There are two generally accepted mechanisms by which to perform a survey: use of interviews or questionnaires (Lindgaard, 1994, Faulkner, 2000, Dix et al., 1993, Hix and Hartson, 1993, Newman and Lamming, 1995, Preece et al., 1994). On the basis of their applicability and suitability relative to this survey, and despite the disadvantage that their results have to be taken at face value since the researcher cannot (beyond the use of pilot studies and question validation) usually confirm that individual respondents interpreted the questions as they were intended, questionnaires were selected as the most appropriate mechanism for this survey (Lindgaard, 1994, Faulkner, 2000, Dix et al., 1993, Hix and Hartson, 1993, Newman and Lamming, 1995, Preece et al., 1994). Given the outlined research agenda, the structure, style, and number of questions had to be determined so that the questionnaire was best suited to systematically collect the required data.

The format and content of the questions

The questions were designed to be as brief, clear, and relevant as possible (Lindgaard, 1994, Preece et al., 1994). Although experts suggest that the wording of questions be kept relatively simple (Lindgaard, 1994, Preece et al., 1994), the technical nature of this survey, together with the fact that the subjects were experts in the associated field, meant that the language was not a major concern. Where possible, the questions were phrased in a positive manner and in the present tense to most effectively avoid misinterpretations (Lindgaard, 1994). Although it is generally considered preferable to adhere to one out of the many styles of question throughout a questionnaire (Lindgaard, 1994), this was not possible given the complexity and varying nature of the data being investigated. To compensate, the different question formats were given a different visual appearance, and instructions clearly indicating the manner in which the questions should be answered were provided in each instance (Lindgaard, 1994). Where magnitude estimates were required, a

five point scale was adopted wherever possible to increase flexibility of response and reduce the number of opt-out-based selections of mid- or neutral- points (Lindgaard, 1994, Dix et al., 1993).

The layout and structure of the questionnaire

The questionnaire dealt with several issues and so was sub-divided accordingly (see Appendix A). Each section or subsection included a brief introduction to make the respondent aware of the nature of, and reason for, the associated questions (Lindgaard, 1994). The questionnaire began by asking the respondents to give general contact details, and briefly describe their qualifications and position within their organisation (Dix et al., 1993). In accordance with Lindgaard's advice that the more delicate or tricky questions should be located approximately one third of the way through a questionnaire – a point at which the respondent will be 'warmed-up' but will not yet be tired or bored – the more complex, and perhaps sensitive, questions regarding work practice were included after the general examination of project details (Lindgaard, 1994).

The number of questions was restricted to a maximum of 60 (Lindgaard, 1994), and based on the discussed structure, the included questions were designed to uncover (for each individual organisation):

1. the manner in which UIDTs had been chosen;
2. the nature of the tools used and the subsequent levels of satisfaction expressed by the people using the various tools;
3. and the software engineering methodology employed.

Questionnaires can only elicit the *respondent's* account of the topic under investigation, and so to reduce respondents' generalisation (or idealisation) of their work situation, they were asked to focus on a single application domain when answering the questions regarding project structure and work practice.

3.2.2 The Process

One of the most significant drawbacks to the use of questionnaires is the fact that respondents do not always interpret, and therefore answer, questions as they were intended. To reduce interpretative discrepancies trial completions of the questionnaires were conducted. The respondents (some of whom were academically acknowledged HCI experts) gave critiques which were fed back into the development of the final questionnaire. In addition, two interviews – structured according to the questionnaire – were conducted to gauge whether the correct questions were being asked and/or whether

their phrasing led to misinterpretation or ambiguity. The interviewees were software developers working on relatively large-scale applications for other university departments. The results of the interviews were used to inform the completion of the final questionnaire, a copy of which is included in Appendix A.

Thirty copies of the questionnaire went sent by standard post to a selection of software development companies within the Glasgow area[39]. The companies were identified by means of the *Scottish Software Association: Directory of Member Companies 1995* and targeted according to size and area of specialisation to obtain a cross-section of both – it being hypothesised that these attributes might play an important rôle in the choice of UIDTs (Scottish Software Federation, 1995). An encouraging 50% response rate was obtained – 14 questionnaires out of the 30 issued were returned completed, and one questionnaire was returned uncompleted[40]. The latter is excluded from the results discussed in the remainder of this chapter. A number of the returned questionnaires were accompanied by additional documentation pertaining to the working practice employed within the organisation.

Since the companies were all Glasgow-based, it is possible that geographical bias may have influenced the results of this survey. Little is yet known regarding the influence of geography on approaches to software development, and it is considered beyond the scope of this thesis to investigate the matter further. Such investigation would, however, be an interesting avenue for future work, and is therefore discussed in chapter 8.

The remainder of this chapter details the findings of the survey. Unless otherwise indicated, the figures in each table/chart show the absolute number of companies within each of the identified domains. Where incomplete data was returned (usually due to the failure of a respondent to answer part or all of a question) figures often do not reach the expected total (14 in the case of general results; and the domain-specific subtotals elsewhere). In such circumstances, the results are flagged with a †.

3.3 Structure of Presentation of Survey Results

Respondents were asked to identify an application domain for which they would answer questions regarding the nature of projects and work practice (see question 2.5[41]). The

[39] This geographical location was selected for ease of accessibility should further interviews etc. be considered useful or necessary (the questionnaires were not anonymous).

[40] With a covering letter to explain that the respondent did not feel he was in a position to assist in the research.

[41] This, and all subsequent question numbers refer to the questionnaire included in Appendix A.

domains identified by the respondents fell into six different categories, as follows (the letters in brackets indicating the abbreviations to be used throughout this chapter):

1. Database Applications **(D)** – 5 companies;
2. Manufacturing Control **(M)** – 1 company;
3. Software Tool Development **(S)** – 2 companies;
4. Educational Systems **(E)** – 3 companies;
5. Information Systems **(I)** – 2 companies;
6. Real-Time Systems **(R)** – 1 company.

The results are presented collectively – in the general **(G)** sense – and then, where appropriate, for each domain. Per domain trends are discussed and conclusions drawn where possible.

It was hypothesised that the size of an organisation might influence its choice of UIDT. Although the results obtained did not noticeably support this hypothesis, there were a few instances where the size of the organisation appeared to have had some impact on UIDT selection, beyond determining the available budget for UIDT purchasing. In the following presentation of the results, attention will only be drawn to company size where it is considered to be of potential influence or importance.

3.4 The Survey Results

This section presents the observed results. The data is interspersed with comments and some initial discussion for the purpose of explanation. A more cohesive discussion of the results as a whole is included in section 3.5. The small number of responses mean that these results cannot be generalised beyond the scope of the survey and therefore no statistical significance will be attributed to them. However, given the use to which the results will be put, what is important is that associations have been observed. The results should therefore be considered a qualitative observation of the important issues informing the development of a new UIDT evaluation methodology.

3.4.1 Application Characteristics

Section 2B of the questionnaire focused on application characteristics (relative to the identified domains - see section 3.3). The questions covered: the typical size of the applications; their safety-criticality; the locus of control associated with the applications;

and the data types manipulated by the applications. The responses to each of the observed issues of application characteristics are presented below.

3.4.1.1 Application Sizes

Figure 3.1 shows the size of typical applications (measured in number of lines of code) across each identified domain. For the companies questioned, the mode is middle-sized applications of between 10,000 and 100,000 lines of code. Given that few of the respondent organisations comprise more than 50 software developers, it could be surmised that the organisations are more suited to coping with this scale of development. Alternatively, there may be greater (or more frequent) demand for applications of this size than for larger applications.

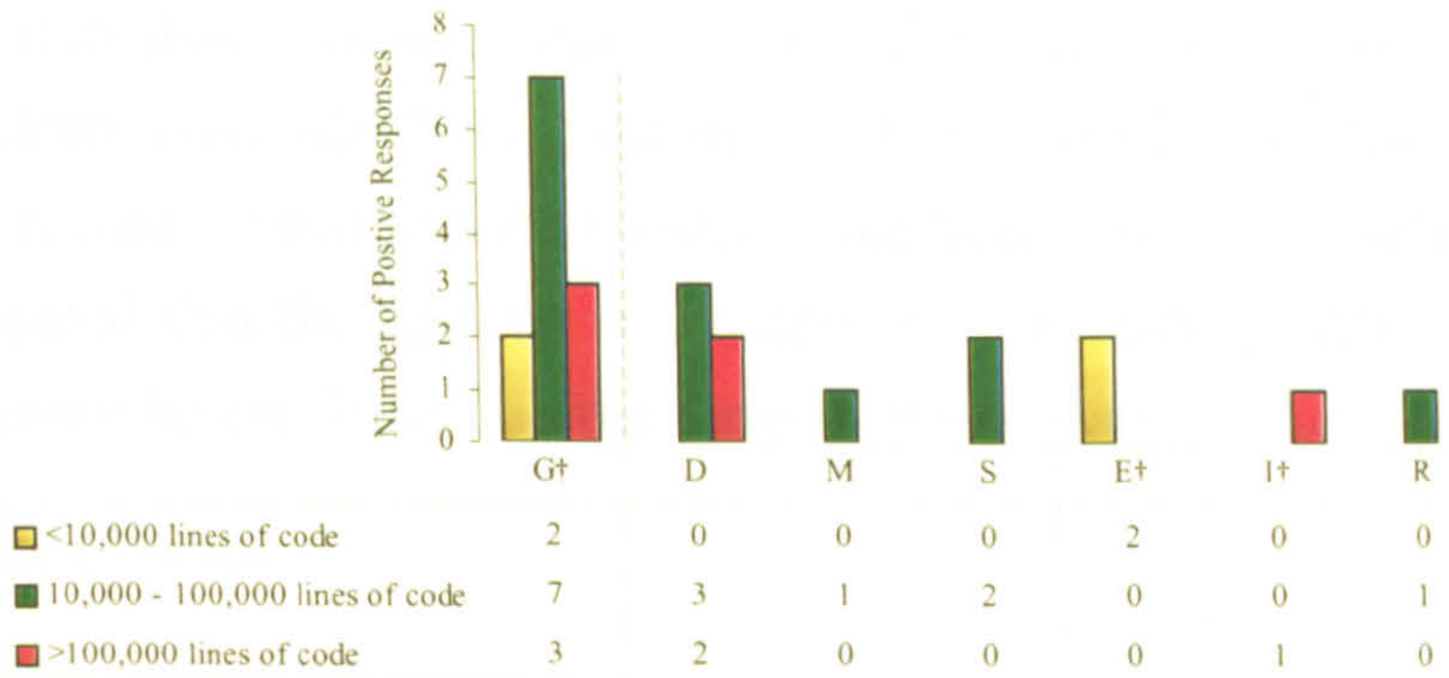


Figure 3.1 – *Domain application sizes*

The domains of D, M, and R[42] reflect this modal application size whereas E tends towards smaller scale applications. Given the size and complexity of most commercial software development tools, it was anticipated that S would reflect large-scale application development. It too, however, is dominated by medium sized application - the reason for this being unclear.

3.4.1.2 Safety Criticality of Systems

Table 3.1 shows that 4 out of 14 respondents claim that they produce safety-critical applications and that these are concentrated in three of the domains – D, I and R. Unsurprisingly, all applications in R are considered safety-critical. It was interesting, however, that half of the applications in I, and 2 out of 5 in D, were deemed to be such.

[42] For full identification of the domains denoted by these letters - in this and all following discussion - refer to section 3.3.

Domain	N° of domain-specific responses for which applications were rated 'safety-critical'
G	4
D	2
M	0
S	0
E	0
I	1
R	1

Table 3.1 – *Safety-criticality of systems*

Applications within these domains may be considered safety-critical on the grounds that they provide information which is used in a safety-critical field (for example, database systems which record sensitive safety-critical medical research results/data). It is not, however, anticipated that the applications themselves are safety-critical in the same sense as R, but this cannot be confirmed on the basis of this survey.

3.4.1.3 Locus of Control

'Locus of control' was introduced by Rasmussen, and refers to the freedom of action available to the application user (Rasmussen, 1992). He identifies three categories of locus of control, namely:

1. *autonomous user served by system* – users use the software to achieve their own pre-determined goals and are not constrained in the manner in which they use the system;
2. *autonomous user with constraints* – the user is still in charge of the system, but has to use the application in accordance with some pre-determined rules or policies - e.g. administrative case handling systems;
3. and *operator serving the system* – the user is paced and constrained by the laws of physics – e.g. process plants.

Respondents were asked to indicate the locus of control for typical applications within their identified domain (see question 2.8). The responses are shown in Figure 3.2.

For the observed domains, the modal locus of control is autonomous users served by the system and is reflected in the domains M, S, and E. Despite being expected in the case of S and E, it is perhaps more unusual in the case of M. Although it is anticipated that this is due to the particular manufacturing process for which the observed application was being developed, further investigation would be required to clarify the situation.

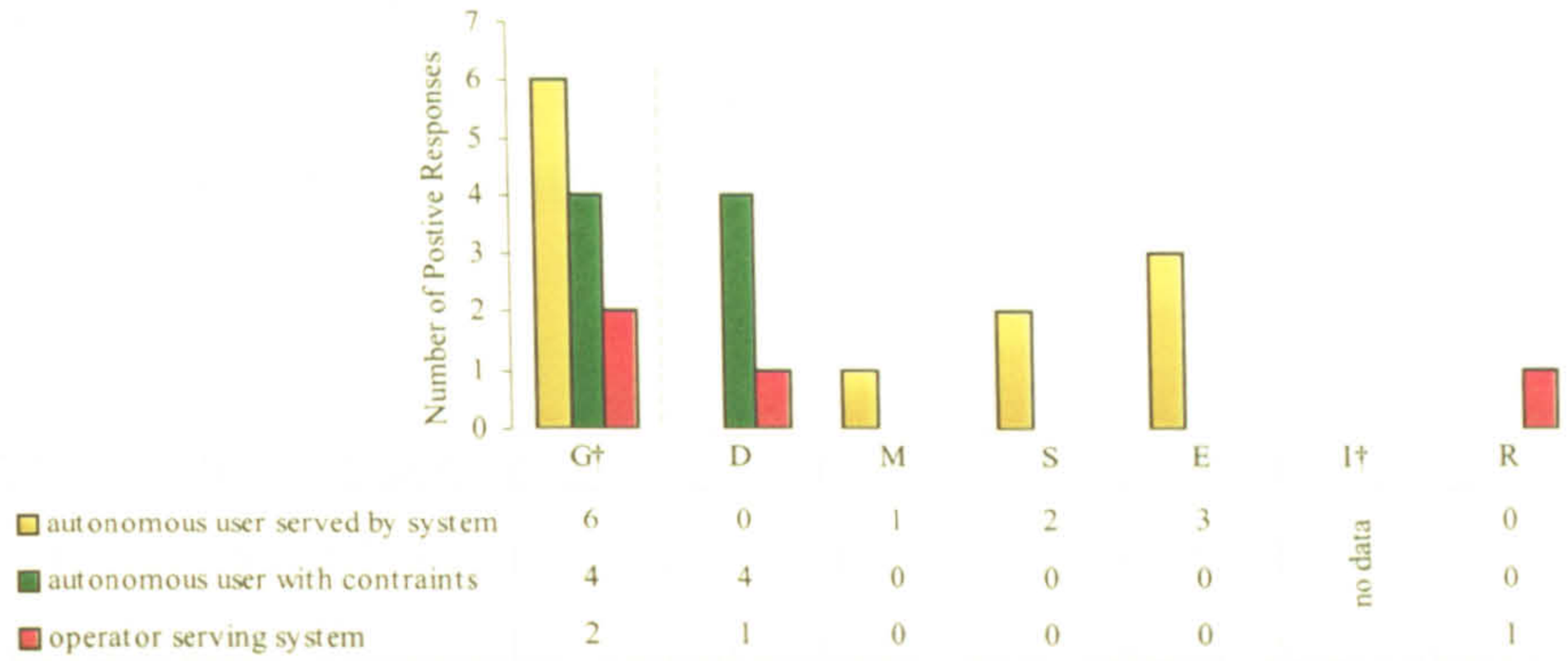


Figure 3.2 – *Locus of control*

With the majority (4 out of 5) of its applications considered to impose constraints on autonomous use, D differed from the mode. As above, this is likely to be due to the specific purpose for which these applications were being developed – for example, applications such as GPass (a G.P. patient record management application) that manipulate medical records normally impose data entry/access constraints on their users. Contrary to the mode, the application in R was developed to establish its user as an operator controlled by the system.

3.4.1.4 Platform

Some UIDTs are platform-specific and so it was hypothesised that the platform for which applications were being developed may (quite heavily) influence UIDT selection. Respondents were therefore asked to specify whether their applications were developed for single or multiple platforms. The results can be seen in Table 3.2.

Domain	Cross Platform	Single Platform	Comments
G†	6	6	NT, Unix, Windows95
D	2	3	Windows95
M	1	0	PCs
S	1	1	WindowsNT/95
E†	1	1	NT, Unix
I†	1	No data	Unix, NT
R	0	1	PCs

Table 3.2 – *Platform*

In general, approximately half of the companies studied develop applications for single platforms, and half develop for multiple platforms (see D, S, and E). At opposite poles, M applications are cross-platform, and R applications are designed for single platforms.

Given the potential complexity of single-platform real-time safety-critical system development, it is perhaps unsurprising that this organisation does not produce generic multiple platforms designs and, furthermore, focuses on a single platform type.

3.4.1.5 Domain Data Types

Relative to their identified domains, respondents were asked to highlight the data type(s) manipulated within typical applications. This information allows for identification of typical project-specific functional requirements that might realistically determine the suitability of a selected UIDT - for example, an application's need to handle complex graphics contributes to project-specific requirements that should ideally be supported in a selected UIDT. Such data also highlights the general complexity of the observed applications. The identified data types are shown in Figure 3.3.

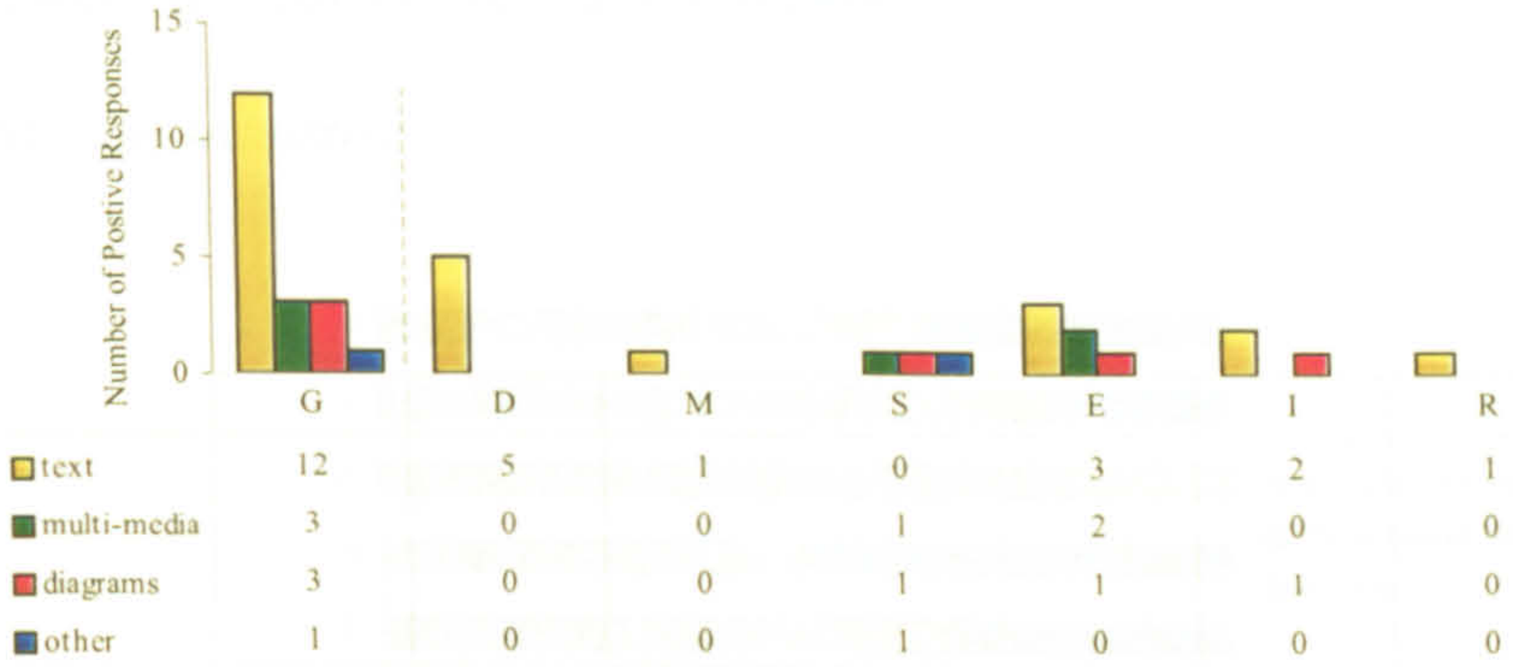


Figure 3.3 - *Domain data types*

Within the scope of this survey the predominant underlying data type for which applications are being developed was text (including numeric data) - as is evident in the domains of D, M, E, I, and R.

E applications make greater use of alternative data types. Proportionately, I applications made the most use of diagrams, which could be accounted for by the fact their efficiency and effectiveness as a means of data representation are widely recognised (Ahlberg and Shneiderman, 1994, Shneiderman, 1998, Rao and Card, 1994). It is unclear why applications in category S make no use of textual data although it is probably due to an error or misunderstanding during completion of the questionnaire.

3.4.1.6 Data Intensity

Question 2.11(a) of the questionnaire asked respondents to indicate whether or not they considered their applications to be data intensive. The results are shown in Figure 3.4.

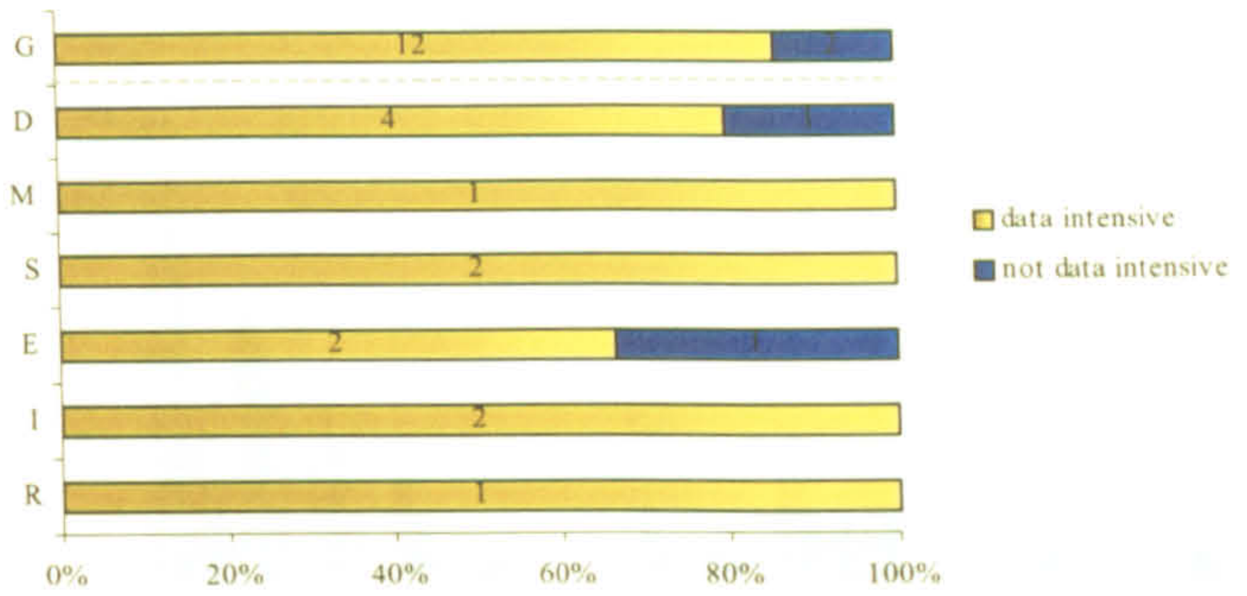


Figure 3.4 - *Data intensity*

There is a general trend towards data intensity in applications. Given that computer applications are often developed for the management of data (especially in large quantities), this is perhaps unsurprising. The main counterexample for this trend is E, a third of which were not considered data intensive.

3.4.1.7 Interaction Intensity

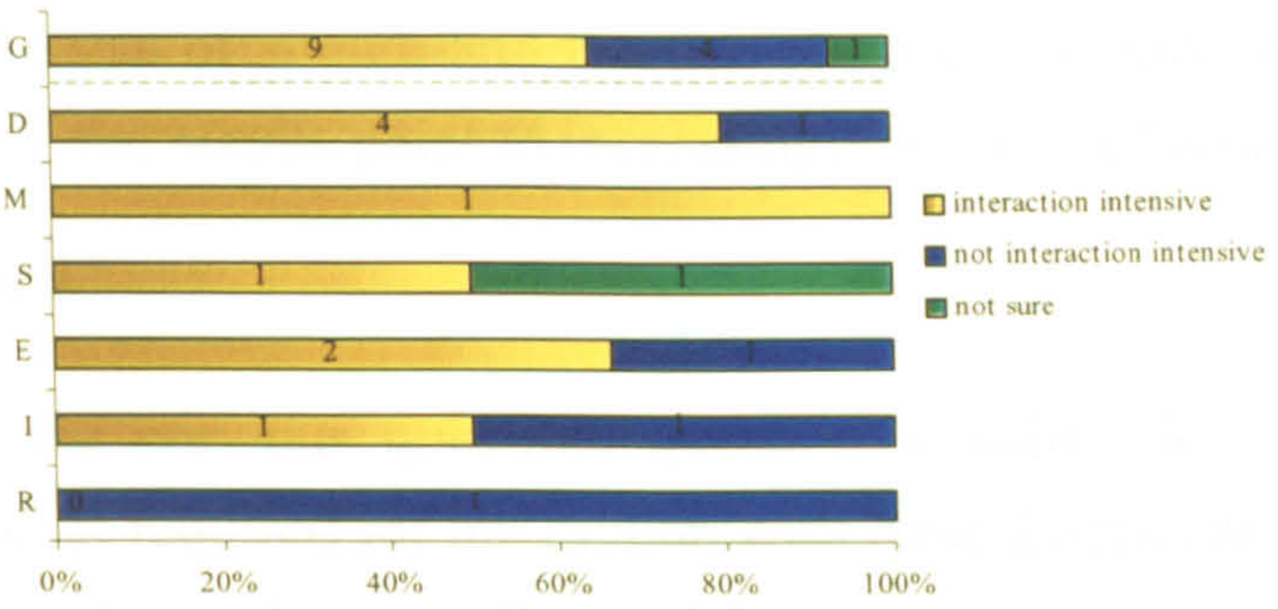
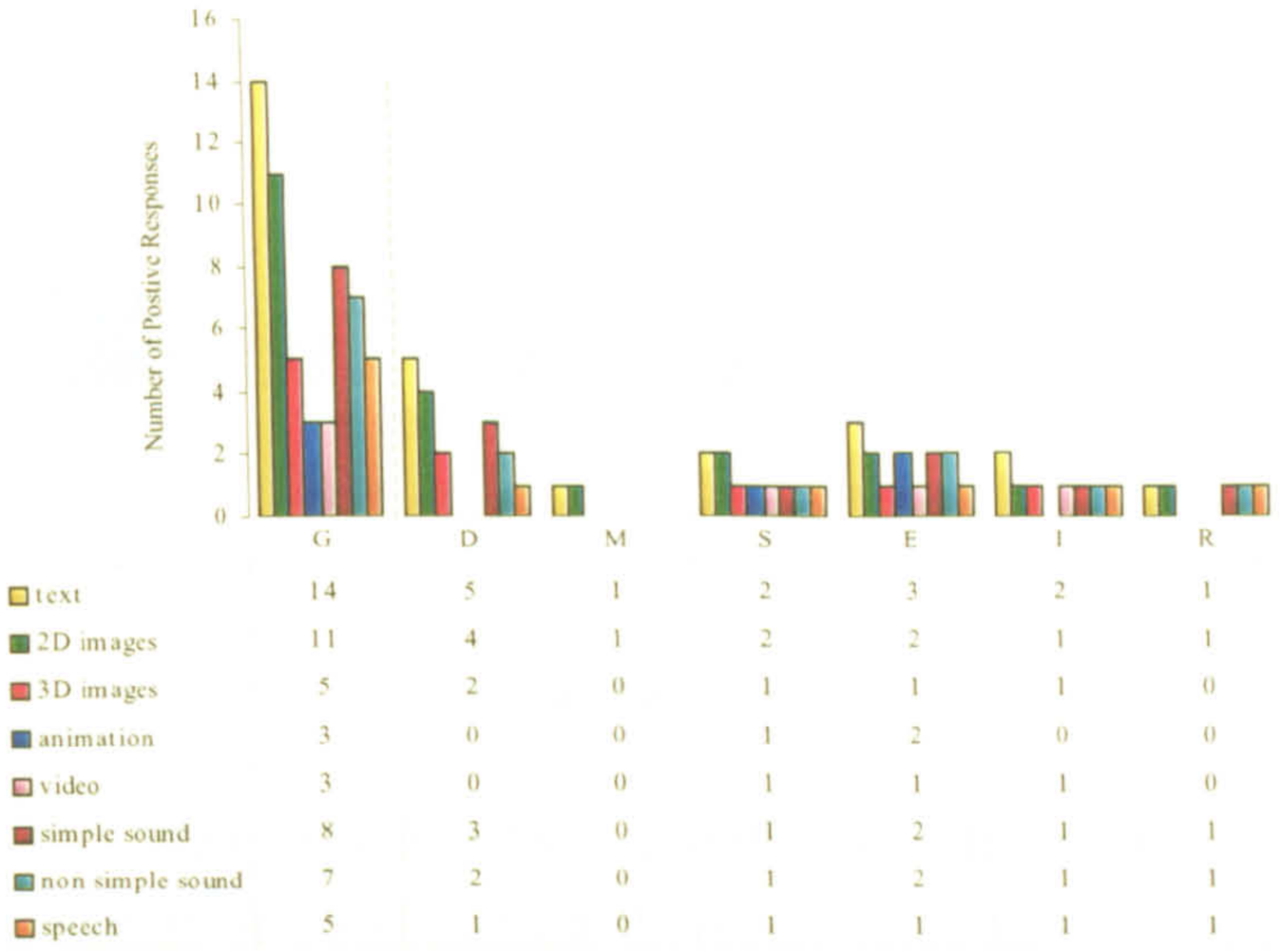


Figure 3.5 – *Interaction intensity*

As with data intensity, respondents were asked to consider the interaction intensity of their applications - their responses are shown in Figure 3.5. In general, two thirds of applications were considered to be interaction intensive – a trend which is matched almost exactly by domain E. Domains D and M rate an even higher proportion of applications as interaction intensive in contrast to applications in R which are not considered interaction intensive. I shows a more even distribution which would seem to be influenced by the specifics of the type of information the system is supporting.

3.4.1.8 Output Media Used

To obtain an overview of the nature and complexity of the user interfaces to their applications, question 3.14(a) asked respondents to indicate the output media used for typical applications within their identified domain. Figure 3.6 shows the number of companies (per domain) which make use of the various media types.



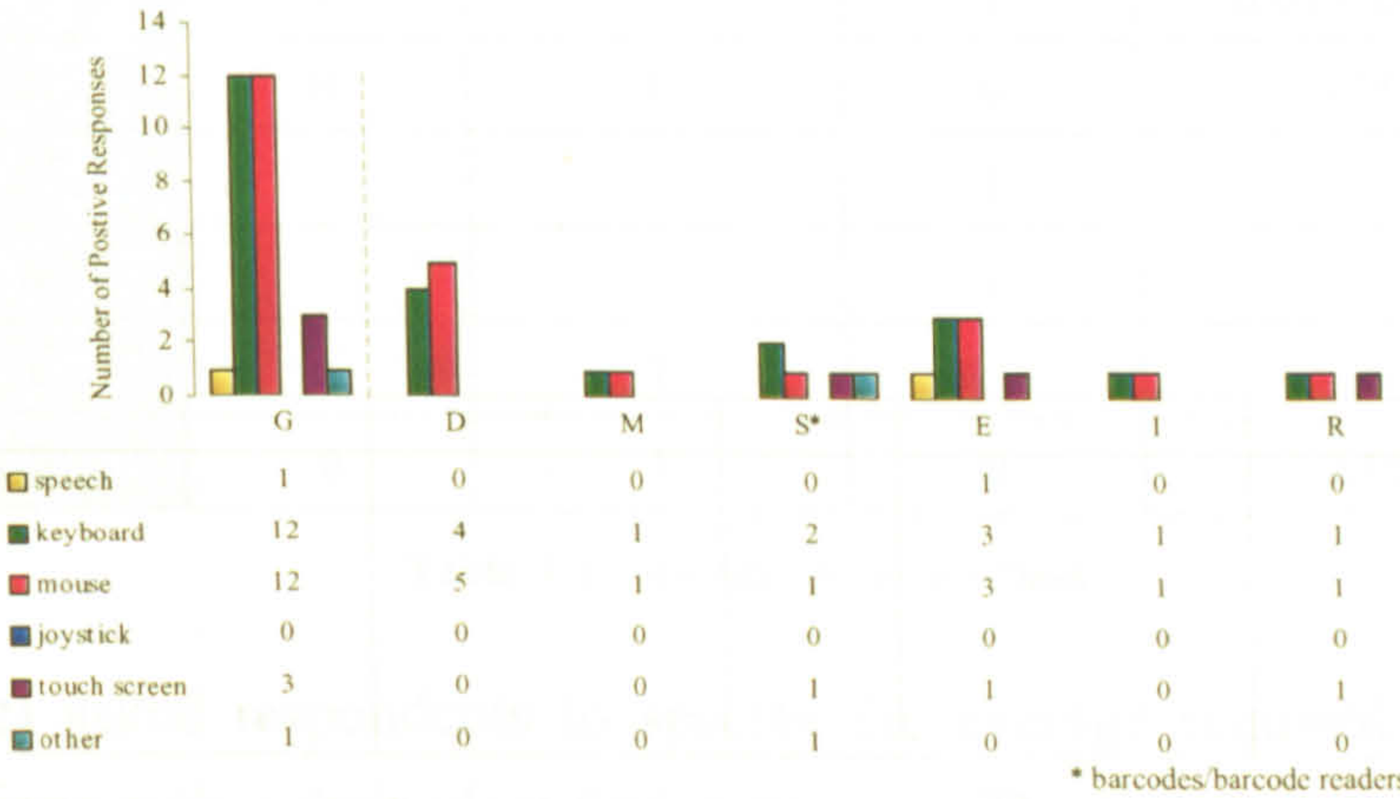
	G	D	M	S	E	I	R
text	14	5	1	2	3	2	1
2D images	11	4	1	2	2	1	1
3D images	5	2	0	1	1	1	0
animation	3	0	0	1	2	0	0
video	3	0	0	1	1	1	0
simple sound	8	3	0	1	2	1	1
non simple sound	7	2	0	1	2	1	1
speech	5	1	0	1	1	1	1

Figure 3.6 – *Output media used*

Text remains the most commonly used output media although use of 2D images is also popular. Applications in E and I cover the widest range of output media. More than half of respondents reported the use of sound, and the use of speech sounds was higher than might be expected. Overall, the results point to increasingly multi-media interfaces.

3.4.1.9 Input Devices Used

Sharing motivation with the previous question, question 3.14(b) asked respondents to comment on the input devices required for interaction with typical applications within their identified domains - the results are shown in Figure 3.7.



	G	D	M	S*	E	I	R
speech	1	0	0	0	1	0	0
keyboard	12	4	1	2	3	1	1
mouse	12	5	1	1	3	1	1
joystick	0	0	0	0	0	0	0
touch screen	3	0	0	1	1	0	1
other	1	0	0	1	0	0	0

* barcodes/barcode readers

Figure 3.7 – *Input devices used*

Unsurprisingly, the keyboard and mouse are the most widely used of the standard input devices. E was the only domain in which speech input was used. Although touch screens were used by almost 1 in 4 of respondents, their absence from I is noticeable.

3.4.1.10 User Type

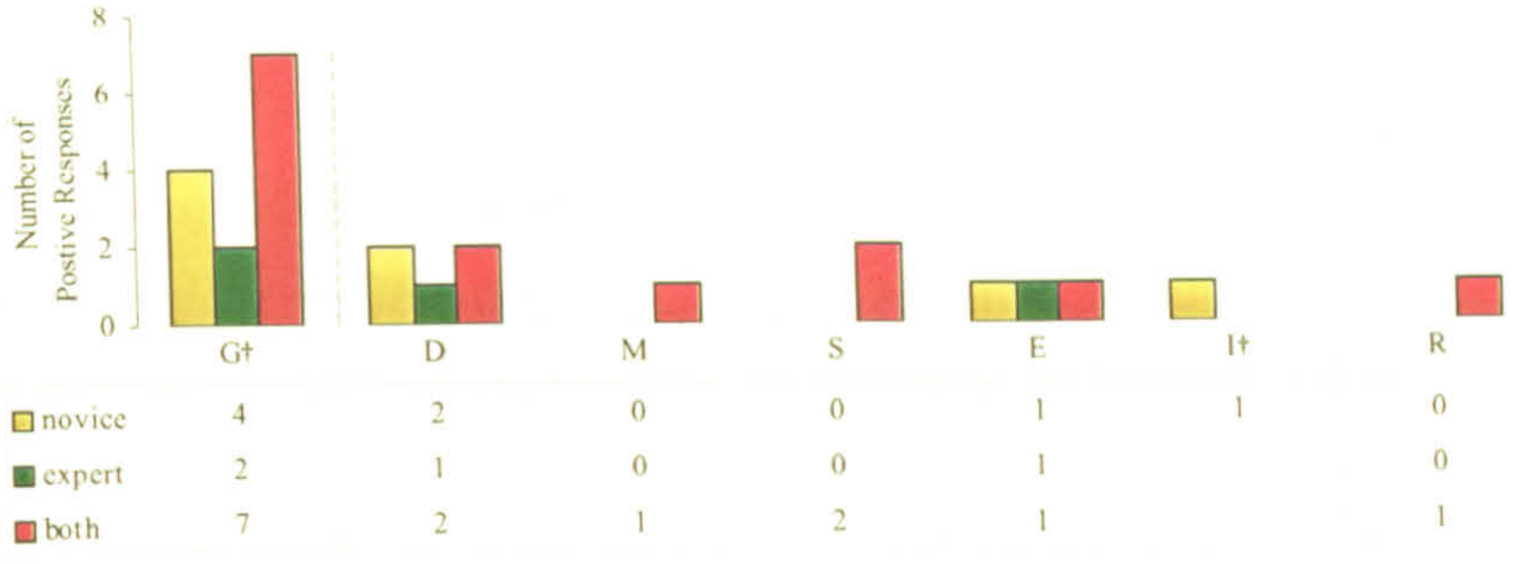


Figure 3.8 - *User type*

To determine whether their user interface design was influenced by the expertise of their end-users (novice, expert or combination thereof), companies were asked to indicate the characteristic expertise of the user base for typical applications in their identified domain. Figure 3.8 shows the reported user base characteristics.

In general, applications typically cater for both novice and expert users. Only domain I stresses support for novice users - although the information returned was incomplete for this domain.

3.4.1.11 Average Response Times

Domain	<0.5 secs	0.5 – 2.0 secs	>2.0 secs	Range (secs)
G†	1	4	5	0.01 - 20.00
D	1	1	3	0.01 - 20.00
M	0	1	0	1.00
S†			1	
E†			1	
I†		1		
R	0	1	0	2.00

Table 3.3 – *Average response times*

Question 2.11(c) asked respondents to specify the average required response times for typical applications within their identified domains. The times given are shown in Table 3.3, in which the range of times per domain is given only where all respondents within that domain indicated response times.

Required response times are generally in the region of 0.5 seconds or more. Domain D represents the greatest range in response times, and thereby introduces the minimum and maximum outliers in the data set.

3.4.1.12 Cognitive Load

Cognitive load refers to the mental effort required of users during interaction with a system. On a scale of 'low', 'medium', 'high', and 'don't know', respondents were asked to indicate their assessment of the cognitive load that was characteristic of typical applications in their identified domain. The results, which serve as an indicator of the complexity of their typical user interfaces, are shown in Figure 3.9.

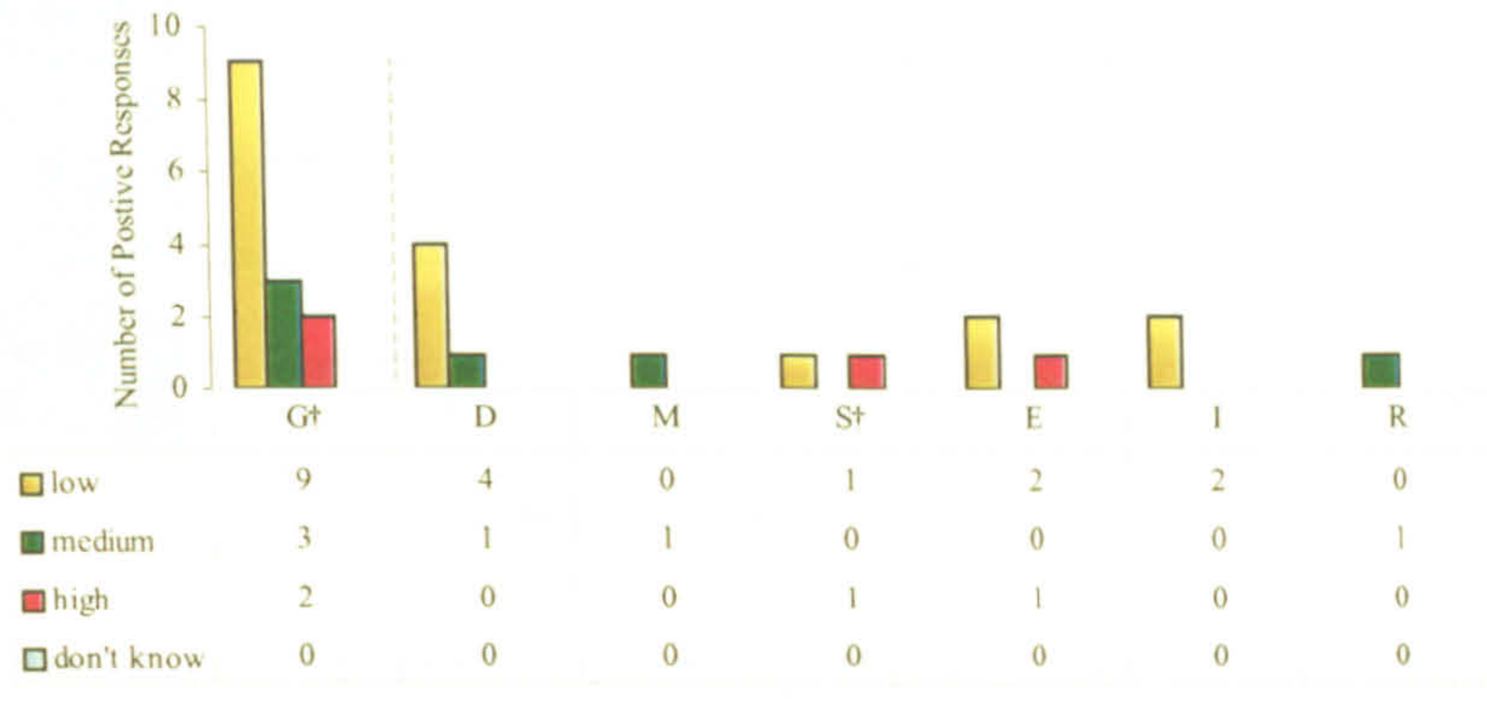


Figure 3.9 - *Cognitive load*

Although these results suggest that cognitive load is generally low across typical applications in the identified domains, the respondents' assessments should be considered with caution since it is likely they would be reticent to admit to placing high demands on their users[43]. In general, domains D, E, and I comply with the trend for low cognitive demand, whereas M and R reflect medium demand levels. Since applications in domain I are often used infrequently or irregularly by any given user, high cognitive demands would be detrimental to their acceptance and use. It is therefore unsurprising that typical applications in this domain receive a low cognitive demand rating. One third of respondents in domain E rated the cognitive demand for their typical applications as high. It is possible that these higher than (perhaps) expected cognitive demands are a consequence of increased visual processing demands placed on the users by the wide range of data types incorporated in such applications (see sections 3.4.1.5 - 3.4.1.9). Alternatively, they could be a factor of the nature of the educational topic. Further investigation would be necessary for clarification of the actual cause.

3.4.1.13 Average Critical Load

The average critical load of an application is the average number of users who could conceivably be using the system simultaneously. Table 3.4 shows the average critical load

[43] To assess whether the applications do in fact place high or low cognitive demands on their users would require extensive further investigation.

for typical applications within the identified domains. Ranges are only given for individual domains where all associated respondents indicated a critical load.

Domain	**1** (person)	**≥ 2 - ≤ 20** (people)	**> 20** (people)	**Range** (people)
G†	2	5	3	1 – 1000
D†	1	3		
M	0	0	1	24
S†			1	
E†	1	1		
I†			1	
R	0	1	0	4

Table 3.4 – *Average critical load*

No general statement can be made regarding the number of simultaneous users. The domains of R and D cater for a load of between 2 and 20 users, in contrast to domain E in which applications are often used by one person. Systems developed for domains M and I are often used by more than 20 people - examples of applications in domain I are often in public places and are remotely accessible from multiple sites at the same time.

3.4.1.14 Summary of Identified Application Characteristics

Relative to the scope of this survey, domain-specific application characteristics have been identified. Application sizes are similar across all domains, and less than one third of applications are considered safety-critical. In general the applications considered to be non-safety-critical serve autonomous users. Database applications which are considered safety-critical allow autonomous use within certain constraints. In contrast, the rôle of users in safety-critical real-time systems is that of operator.

With the exception of real-time safety-critical systems, the applications are being developed for multiple platforms. Text is the most common domain data type and most applications are considered to be data intensive. In general, interaction intensity is directly proportional to the level of user autonomy - the more interaction intensive the applications the greater the autonomy of the user, or in contrast, the lesser the autonomy of the user the lesser the intensity of the interaction (for example, real-time systems).

The range of reported output media points to increasingly multi-media user interfaces. Once again, interaction intensity is directly proportional to the range of output media employed - the greater the range of output media, the greater the intensity of interaction, or conversely, where the range of output media is more restricted the interaction intensity is reduced as, in turn, is the autonomy of the user.

The keyboard and mouse are the most commonly employed input devices. Second to these is the touch screen which is used in both interaction intensive non-safety-critical applications (for example, educational systems), and non-interaction intensive safety-critical applications (for example, real-time systems). In general, applications are developed for both novice and expert users. Only some systems within the domains of database applications and educational software are designed solely for expert users - a consequence of the rôle of the specific applications. The majority of applications report a low cognitive demand[44]. Where cognitive load is reported as medium (for example, real-time and manufacturing control systems) or high (for example, an educational system) it is likely to be a consequence of the specific nature of the individual applications.

Where response times are most restricted or specifically defined (for example, 0.5 - 2.0 seconds) applications are either safety-critical or novice-centric (for example, some information systems). Applications within the database and educational software domains have the longest acceptable response times - the average, however, is 0.5 seconds. Critical load varies considerably across the different domains.

These observations illustrate the complex connectivity between the output media, interaction intensity, user autonomy, safety-criticality, and required response times of applications. Thus, the functional requirements placed upon a UIDT will potentially be a complex, application-specific combination of issues driven by one more of the above application characteristics. A new UIDT evaluation methodology must therefore provide the facility to include and tailor functional requirements associated with the above application characteristics on a per-project basis.

3.4.2 Software Engineering Administration

The survey aimed to investigate the nature of the software engineering processes adopted within the respondents' organisations. The questionnaire was therefore designed to examine issues such as the size of individual project teams and the allocation of responsibilities among project team members. The principal intention of this observation was to identify the likely environmental context into which chosen UIDTs are placed in order that a new UIDT evaluation methodology could potentially tackle relevant environmental issues, and therefore be more contextually aware and hence effective. The results of this investigation are outlined in the following sections of this chapter.

---

[44] As mentioned in section 3.4.1.12, it is recognised that the credibility of the respondents' assessments of cognitive load is perhaps questionable and as such this particular finding should be treated with caution.

3.4.2.1 Size of Project Team

It was hypothesised that the size of a project team will influence not only the internal structure of the team, but also the demands placed on the software support required to assist the team. Respondents were therefore asked to state the average size of typical project teams within their identified domain (see question 3.1(a)). Team sizes are shown in Table 3.5, in which the range of team size per domain is given only where all respondents within that domain indicated team sizes.

Teams are predominantly of five or fewer people. Although this raises the question of whether this team size is predominant due to optimum manageability this was unfortunately indeterminable from within the structure of the survey. Domain S has the greatest range of application characteristics or complexity, and this is mirrored by the split in identified team sizes between the two extremes shown in Table 3.5. That said, the results for this particular domain are slightly skewed by the fact that it included the largest of the surveyed companies. This therefore suggests that the size of the organisation exerts influence on the size of project teams therein.

Domain	≤ 5 (people)	6 – 9 (people)	≥ 10 (people)	Range (people)
G†	10	2	1	1 – 50
D	4	1	0	3 – 6
M	1	0	0	5
S	1	0	1	5 – 50
E†	1	1		
I	2	0	0	1 – 5
R	1	0	0	4

Table 3.5 – *Size of project team*

3.4.2.2 Size of the HCI Team

To determine the breakdown of project teams into specialised sub-teams, respondents were asked to state the number of people within a typical project team who are involved with human-computer interaction (HCI). The answers given are shown in Table 3.6, and ranges are again given only for identified domains in which all respondents indicated HCI team size[45].

[45] Domain S has no entries since respondents either failed to answer the question or, in the case of the largest company discussed earlier, the entire team is devoted to HCI..

Approximately one third of identified HCI sub-teams comprise 2 to 3 people. Applications in domain D that have project teams of less than or equal to 5 people allocate HCI responsibility to between 2 and 3 of them. In contrast, all of the team members in M are involved with aspects of HCI. Applications in domain I are split between involving all, and only a proportion, of team members in HCI. Similar splits in other domains are far less pronounced.

Domain	**1** (person)	**> 1 – ≤ 3** (people)	**> 3** (people)	**Range** (people)
G†	2	5	3	1 – 6
D	1	4	0	1 – 3
M	0	0	1	5
S†	no data			
E†			1	
I	1	0	1	1 – 5
R	0	1	0	2

Table 3.6 – *Size of the HCI sub-team*

3.4.2.3 Team Members' Training

Domain	No Computing Science Training	Computing Science Training with No HCI Training	Graphic Design	Computing Science Based HCI Training	Non-Computing Science Based HCI Training
G	6	11	4	7	1
D	2	5	1	1	0
M	0	0	0	1	0
S	1	2	1	2	1
E	2	1	2	2	0
I	1	2	0	0	0
R	0	1	0	1	0

Table 3.7 – *Team members' training*

Placing emphasis on HCI, respondents were asked to categorise the training of team members within typical project teams for their identified domains (see question 3.2). Table 3.7 shows the number of companies for which typical project team members' training falls within the identified categories.

Predominantly, team members have no computing science training or have either computing science training with no HCI training, or computing science-based HCI training. Teams in the domains D, S, and E all include graphic designers; for E this is consistent with their noted high use of graphics and multi-media (see sections 3.4.1.8 and

3.4.1.9). A small number of people have non-computing science based HCI training (e.g. cognitive psychologists).

3.4.2.4 Team Involvement Throughout the Development Process

To gauge team member allocation of responsibilities, respondents were asked to indicate whether all project team members are typically involved at all stages during development. They were also asked, where appropriate, to describe any sub-teams operating within a single typical project team. Although a number of respondents did indicate sub-teams, with the exception of domain S, there are little or no similarities between the sub-team allocation either within one domain or across several domains. The specified sub-teams are therefore not listed here. Table 3.8 shows the number of cases where all team members are typically involved throughout the entire project - the domains for which sub-teams were specified have been marked (*).

Domain	Everyone involved throughout project
G	7
D	3*
M	1
S	0*
E	1*
I	1*
R	1

Table 3.8 – *Team involvement throughout the development process*

3.4.2.5 User Interface Designers' Responsibility Regarding Underlying Functionality

It was anticipated that if user interface developers were also required to develop the underlying application functionality this would influence the selection of software support. Respondents were therefore asked to indicate whether user interface designers typically develop the underlying functionality for applications within their identified domain. Figure 3.10 shows the number of companies in each identified domain for which the user interface designers also develop the underlying functionality.

In the majority of cases, the people developing the user interface also develop the underlying functionality. No explanation for this is given, but it may be due in part to the size of the companies - or more particularly the project teams - involved.

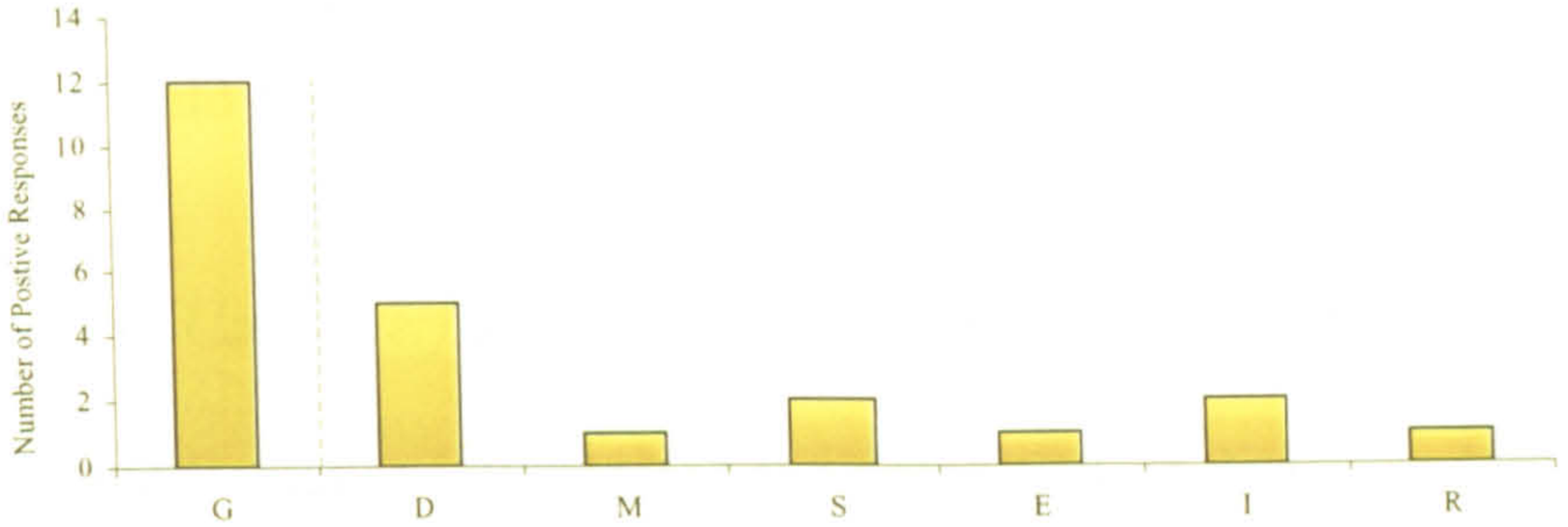
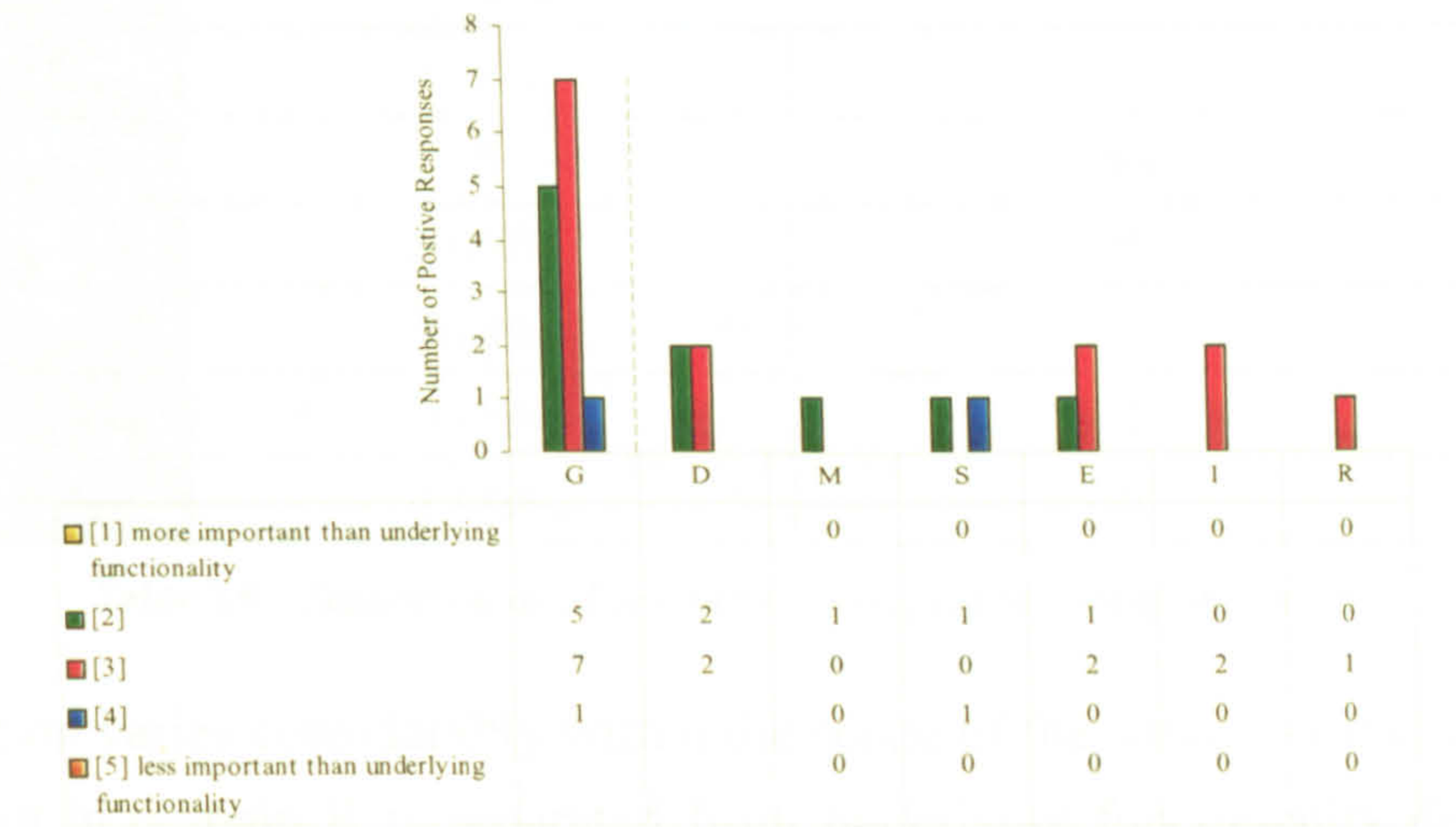


Figure 3.10 - *Projects in which user interface designers are also responsible for underlying functionality*

3.4.2.6 Rating of Importance of User Interface Construction

It was felt that the respondents' opinions regarding HCI would influence the manner in which they answered certain questions. Given a scale of 1 ('UI construction more important than that of the underlying functionality') to 5 ('UI construction less important than that of the underlying functionality'), respondents were therefore asked to rate the importance of user interface construction relative to that of underlying functionality development. Their reactions are shown in Figure 3.11.



	G	D	M	S	E	I	R
[1] more important than underlying functionality			0	0	0	0	0
[2]	5	2	1	1	1	0	0
[3]	7	2	0	0	2	2	1
[4]	1		0	1	0	0	0
[5] less important than underlying functionality			0	0	0	0	0

Figure 3.11 - *Subjective rating of importance of UI construction*

The majority of respondents rate UI importance equal to that of the underlying functionality. Since respondents faced with rating a statement regarding which they have no strong opinion normally settle for the mid-point on the given scale (Dix et al., 1993), these results could represent a balanced view held by half of the respondents or could equally represent their ambivalence - further investigation would be necessary to determine which.

Approximately one third of respondents rated the importance of UI construction higher than that of the underlying functionality. These respondents represent companies for which HCI teams are typically larger and comprise the most HCI training amongst team

members, and all team members are typically involved throughout the entire project. That said, these results should be treated with due caution given the potential influence of the Hawthorne Effect.

3.4.2.7 Separation of User Interface Design from Design of the Rest of the System

Section 3C of the questionnaire investigated the separation of design and implementation tasks within the practical software development processes adopted by the respondent organisations. Respondents were first asked to indicate whether the *design* of the user interface is typically separated from the *design* of the rest of the application and, where separation exists, to indicate whether there is additional separation of graphic design. The results are shown in Table 3.9 in which the first column indicates the number of companies which separate the design of the two aspects, and the second column shows the number of those (i.e. the companies separating design) which further separate graphic design.

Domain	UI Design Separated from Design of Rest of System	(Of which) Further Separation of Graphic Design
G	7 (50%)	3
D	1 (20%)	1
M	0	n/a
S	2 (100%)	0
E	2 (66%)	1
I	1 (50%)	1
R	1 (100%)	0

Table 3.9 – *Separation of UI design from design of the rest of the system*

Design separation varies considerably within the scope of the survey - for example, all user interface design in domain R is separated from underlying functionality design, whereas there is no such separation in domain M. Approximately half of the companies that separate UI and underlying functionality design make an additional separation of graphic design. This is especially noted in the domains of D and I which had graphic design expertise and made high use of graphics respectively. These results suggest that design separation is dependent on, and determined by, the nature of the application together with the expertise available within a project team.

3.4.2.8 Separation of User Interface Implementation from Implementation of the Rest of the System

Respondents were asked to indicate whether the *implementation* of the user interface is typically separated from *implementation* of the rest of the system. Figure 3.12 shows the

number of companies which typically separate the implementation of the two aspects of a software application.

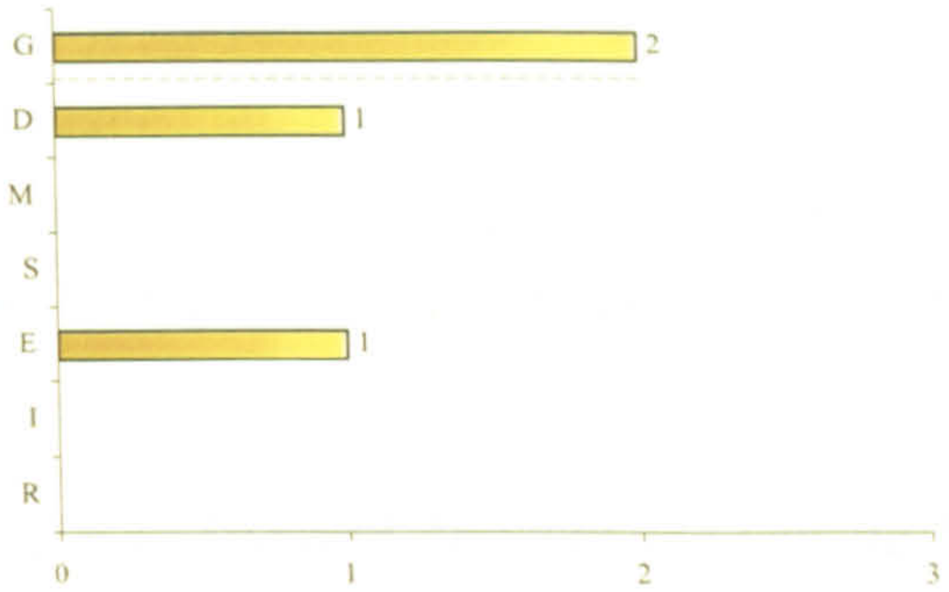


Figure 3.12 - *Number of companies which impose separation between the implementation of the user interface and the implementation of the rest of the system*

Very few of the companies separate the implementation of the user interface from that of the rest of the application. At the level of conjecture, the lack of separation may be due to the fact that achieving *true* separation of user interface code from the code for the underlying functionality can be very difficult (as discussed in Chapter 2).

3.4.2.9 Responsibilities for Design and Implementation

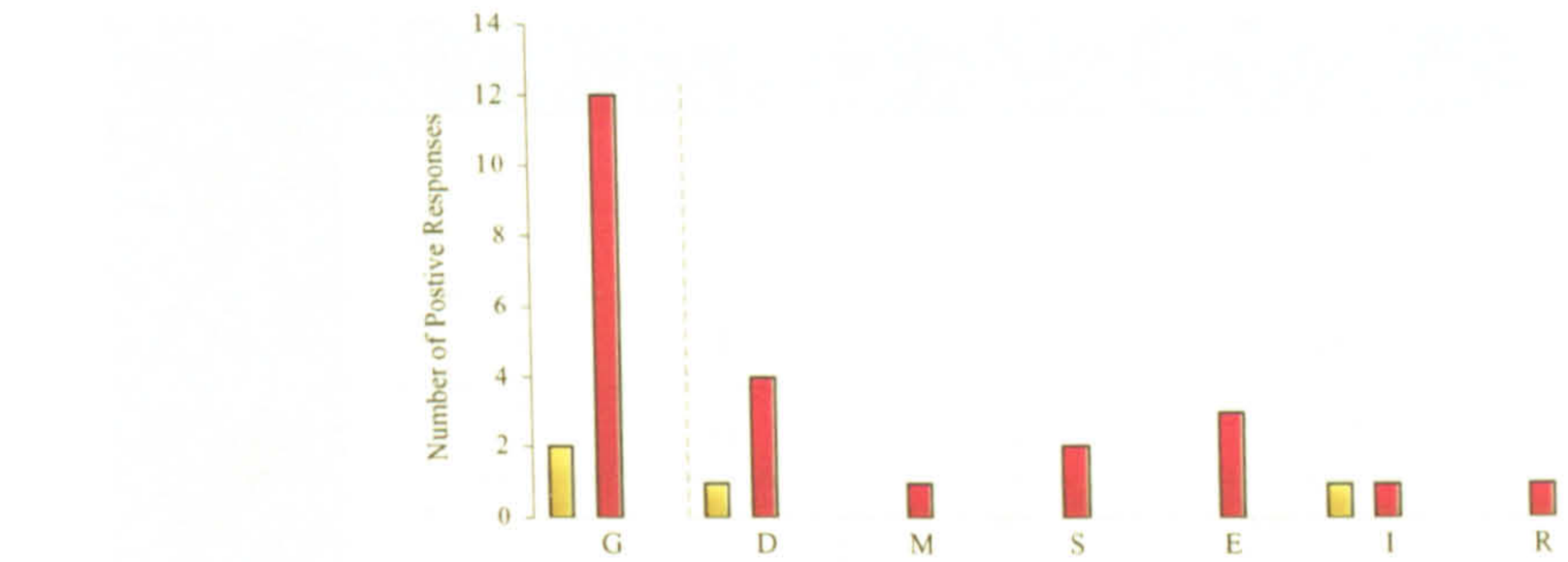
To investigate the allocation and administration of responsibility for design versus implementation, respondents were asked to indicate whether the same person is typically responsible for user interface design and implementation (columns 1 and 2 in Table 3.10) and, where responsibility is split, to indicate the typical method of communication of the user interface design from the designer to the implementor (columns 3, 4 and 5 in Table 3.10). Within Table 3.10, communication mechanisms are only specified where design and implementation responsibility is split. In some instances, more than one method of communication is used.

Domain	Same person responsible for both design & implementation	Different person responsible for design & for implementation	Oral communication of design	Written communication of design	Communication of design via prototype
G	10	4	3	1	3
D	3	2	1	0	1
M	1	0	n/a	n/a	n/a
S	1	1	1	1	1
E	2	1	1	0	1
I	2	0	n/a	n/a	n/a
R	1	0	n/a	n/a	n/a

Table 3.10 – *Responsibilities for user interface design and implementation & associated communication mechanisms*

The majority of companies typically allocate responsibility for both design and implementation of the user interface to the same person(s). Where this is not the case (domains D, S, and E) working prototypes and oral communication are the most popular means by which the design is communicated. To achieve a working prototype usually involves some kind of implementation/programming and so - depending on their style of prototyping - these cases may just be handing over the design at a later/different stage of implementation. Written communication of designs is the least employed means of information transferral.

3.4.2.10 Stages Where HCI Was Thought Appropriate During Project Development



	G	D	M	S	E	I	R
[1] after underlying application functionality specification	2	1	0	0	0	1	0
[2] after underlying application functionality implementation	0	0	0	0	0	0	0
[3] at all stages throughout the software development process	12	4	1	2	3	1	1
[4] not at all	0	0	0	0	0	0	0
[5] other	0	0	0	0	0	0	0

Figure 3.13 - *Stages where HCI was thought to be appropriate during project development*

Given a choice of 5 options - (1) After the underlying functionality has been specified, (2) After the underlying functionality has been implemented, (3) At all stages throughout the software development process, (4) Not at all, and (5) Other - respondents were asked to indicate the point during a project life cycle at which they thought HCI concerns were appropriate. This question was included to gauge respondents' opinions regarding HCI, and so consideration of the results should take into account the potential influence of the Hawthorne Effect. In addition to observing opinion, the question also aimed to ascertain at what (abstract) stages during design, factors of HCI are typically given consideration. The responses are shown in Figure 3.13

The majority of respondents consider HCI to be relevant at all stages during the development process. The only obvious exception is the respondent representing a

company developing applications in domain I who considers HCI appropriate only after the underlying functionality has been specified[46].

3.4.2.11 Percentage of Development Time Devoted to the User Interface

To determine the relative significance - in terms of development costs - of the user interface in relation to the rest of the system, respondents were asked to estimate the approximate percentage of development time (for a typical project) that is devoted to the user interface. This, in turn, indicates the proportion of time for which any UIDT could potentially be used during the course of a project. Table 3.11 shows the number of companies within each domain which devote *x*% of development time to the user interface.

Domain	≤ 10%	> 10 % & ≤ 50%	>50%	Range of time (%)
G	4	6	4	10 – 80
D	1	2	2	10 – 70
M	0	1	0	40
S	1	0	1	10 – 75
E	1	2	0	10 – 50
I	0	1	1	40 – 80
R	1	0	0	10

Table 3.11 – *Percentage of development time devoted to the UI*

The proportional allocation of development time is diverse. Development in domain I devoted the most time proportionally to the user interface, whereas R afforded it the smallest proportion of development time. The latter may be due, in part, to the safety-critical requirement on real-time applications to focus on, and get right, the functionality. Although this apparent lack of significance attributed to the user interface is perhaps unexpected given the potential for a poor user interface to prompt user error, section 3.4.1.3 reported that these applications place the locus of control with the system for which the user is merely an operator. It is therefore possible that either the user is not given scope to take autonomous action and subsequently make errors, or the user interface is not of a nature that requires substantial design effort[47]. The most common percentage of development time to be allocated to the user interface is between 10% and 50%. Agreeing with the general findings of Myers and Rosson (Myers and Rosson, 1992) - as discussed in chapter 2 - this illustrates the variation of scope for potential UIDT involvement.

---

[46] In light of the potential influence of the Hawthorne Effect, this is perhaps a more 'honest' answer.

[47] Further investigation would be necessary in order to clarify this.

3.4.2.12 Use of Step-Completion Methods

The survey investigated general administrative protocols that are followed during software development within the respondent organisations. To determine whether step-completion methods are used in industrial practice and, where adopted, whether they are determinative in the decision to purchase UIDTs, respondents were asked to describe their method of step-completion. In this survey 'step-completion' was taken to be the completion of a small part of a bigger component in the overall software engineering process – for example, the completion of one part of the overall design.

Figure 3.14 shows the number of companies that make use of some kind of structured step-completion method.

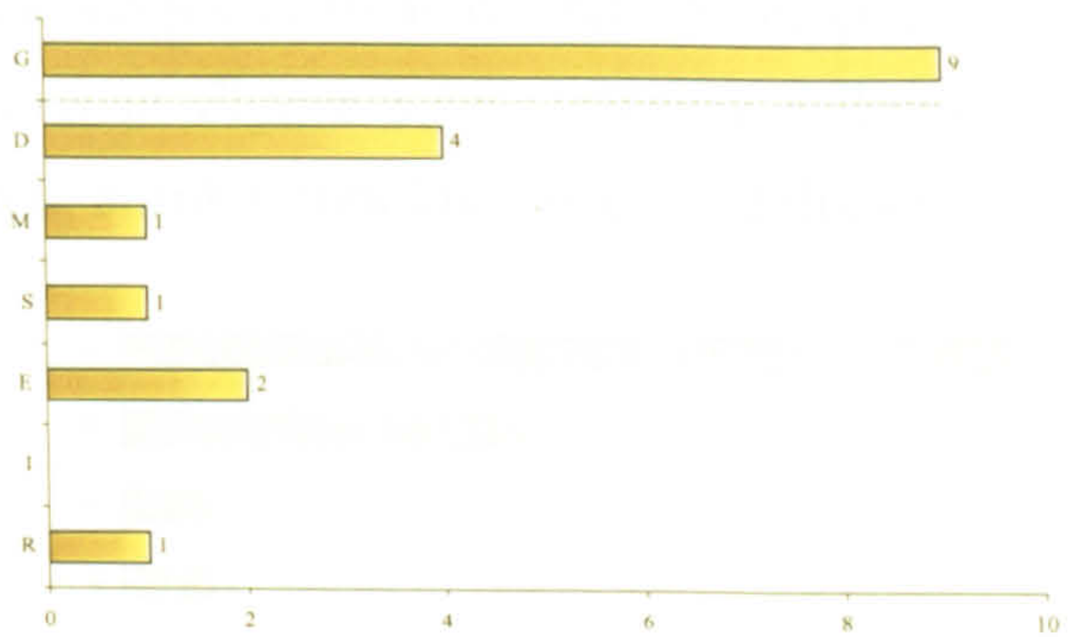


Figure 3.14 - *Use of step-completion methods*

Step-completion methods are used extensively. Furthermore, the more safety-critical the system the greater the use that is made of such methods. The extensive use of step-completion methods[48] suggests that provision of this type of support may factor in the decision to purchase a software support system.

3.4.2.13 Use of Versioning Mechanisms

Respondents were asked to state whether they made use of any form of versioning mechanisms during software development (see question 3.21). The responses are shown in Figure 3.15.

More than half of the companies within any given domain use versioning mechanisms. Once again, the more safety-critical the application domain, the greater the use that is made of versioning techniques. These figures suggest that the ability to support version control is a potentially important facet of a tool when being evaluated for purchase.

[48] Within the scope of this survey.

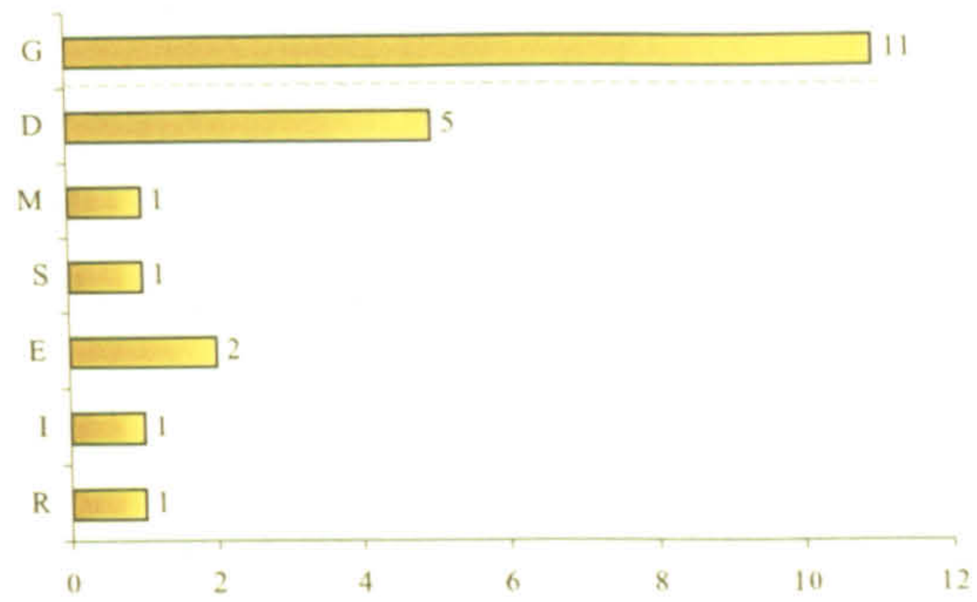


Figure 3.15 - *Use of versioning mechanisms*

3.4.2.14 Use of Signing-Off Procedures

Respondents were asked to state whether they made use of a signing-off procedure at the conclusion of particular stages in their overall development processes. The response to this question - as shown in Figure 3.16 - closely follows that of the use of versioning mechanisms and as such, similar conclusions can be drawn.

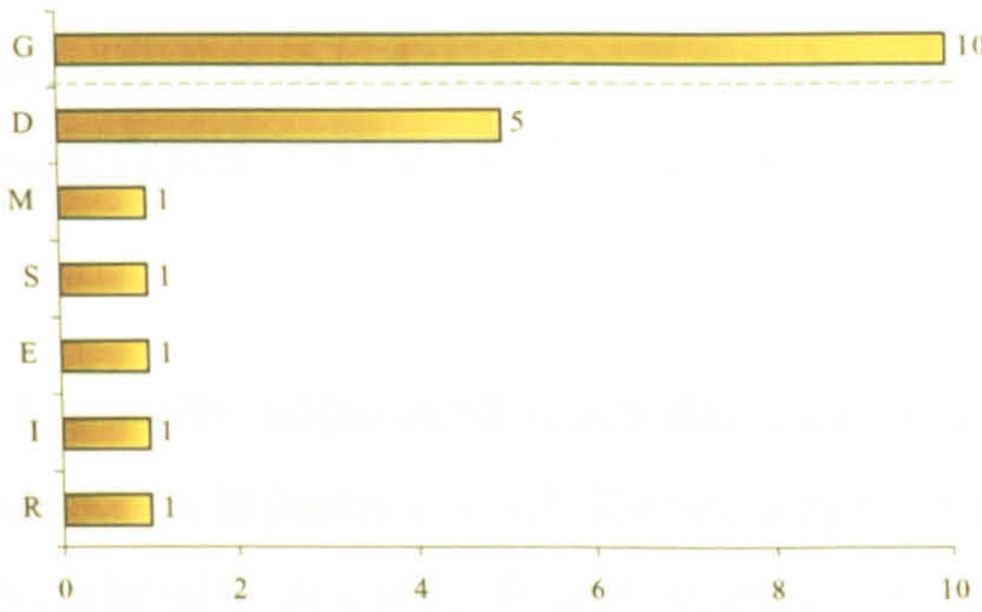


Figure 3.16 - *Use of signing-off procedures*

3.4.2.15 Summary of Software Engineering Administration

The above discussion illustrates the manner in which software engineering projects are administered and structured within the scope of the companies surveyed. Project teams are generally in the region of up to five people, but are influenced by the size of the organisation. Although in some instances all team members are involved with HCI issues, in general HCI sub-teams comprising 2 or 3 people are established.

The majority of project team members fall into one of three training categories: they either received no formal computing science training, received computing science training without HCI training, or received computing science training including HCI training. The projects which had the broadest training base among their team members also included the widest range of output media within their developed user interfaces. Graphic designers are used for applications which make considerable use of graphics and animation and which are interaction intensive. Approximately half of the respondents reported that

members of their project teams are not involved throughout the development process - such projects also reported the broadest training base. This suggests that the selection of a UIDT for use within any given project team needs to carefully consider who will be using the UIDT and for which developmental activities.

In general, the team members responsible for the user interface to an application are also responsible for the development of the underlying functionality. As such, a selected UIDT is likely to need to support a broader range of contextual, and potentially functional, requirements. The one exception to this general rule is educational software development in which the responsibility is split - such applications reported the greatest range in output media and underlying data types and were interaction intensive.

Respondents generally consider user interface construction to be equally important to the construction of the underlying functionality. Those who rate its importance slightly higher than that of the underlying functionality report the largest number of graphic designers or HCI specialists amongst their project teams, have larger HCI teams, and involve most personnel at all stages throughout the design and development process. In general, the same respondents also considered issues of HCI as important at all stages throughout the development process[49].

User interface design is typically separated from the design of the rest of the application within half of the surveyed companies - of these, approximately half further separate graphic design. Correspondingly, graphic design separation is established in project teams that include a graphic designer. Separation of user interface design from that of the rest of the application reflects the interaction intensity and range of output media and input devices - the more complex the user interface, the greater the likelihood that the user interface and underlying application design will be separated. Unlike design, the implementation of the user interface and that of the underlying application functionality is not typically split.

The same personnel are typically responsible for the design and implementation of the user interface. Where these activities are separated, the applications are interaction intensive with the greatest range of output media and input devices, and the design is generally communicated via means of working prototypes or verbal discourse (consistent with the more complex user interface development methods reported in the following sections).

[49] As mentioned in section 3.4.2.6 and 3.4.2.10, these results should be treated with caution given the potential for ambivalence and the Hawthorne Effect.

There is diversity in the reported estimates of the proportion of development time allocated to the user interface. Real-time systems devote the smallest proportion of development time to the user interface; information systems devote the largest - corresponding to the range in output media within the respective user interfaces. Commonly, between 10% and 50% of development time is allocated to the user interface.

Prototypes are used by development teams across all application domains, with evolutionary prototypes being the most popular (this is reflected in the development processes discussed in the following sections). Database application development typically makes use of the widest variety of prototypes - an observation which is mirrored in the percentage of development time devoted to the user interface.

Project development within each of the identified domains makes similar use of step-completion methods, versioning mechanisms, and signing-off procedures. Such mechanisms are especially popular for the development of safety-critical applications. The use of these mechanisms corresponds to projects within which the user interface developers are also responsible for the underlying functionality.

The above observations illustrate the manner in which software engineering projects are organised, and thus the typical environments into which a selected UIDT might be placed. They show the range of responsibilities and skills which typically exist within software development project teams, to which a selected UIDT might need to be sympathetic. Similarly, they identify some of the administrative-centric functionality that might be sought in a UIDT.

3.4.3 UI Design and Development Life Cycles

The relative merits of various software development processes are well documented (Boehm, 1988, Sommerville, 2000). Unfortunately, it is rare for these development processes to spare adequate thought for the complexities of designing and developing the user interface to an application. There is an implicit assumption that the development of the user interface follows the same lines as that of the rest of the application.

Aside from observing the characteristics of user interfaces and the UIDTs used to assist their development, this industrial survey aimed to identify the design and development processes *actually* adopted for user interface development. This section will outline the observed processes and compare them with the well documented development process of Hix and Hartson (Hix and Hartson, 1993).

3.4.3.1 The Model Development Process

The process outlined by Hix and Hartson was selected not only because it is well documented and concise in its coverage of the process of user interface development, but also because it is non-prescriptive (Hix and Hartson, 1993). The remainder of discussion in this section focuses on development processes specific to the user interface, and does not attempt to compare them with, or link them to, the development of the underlying functionality.

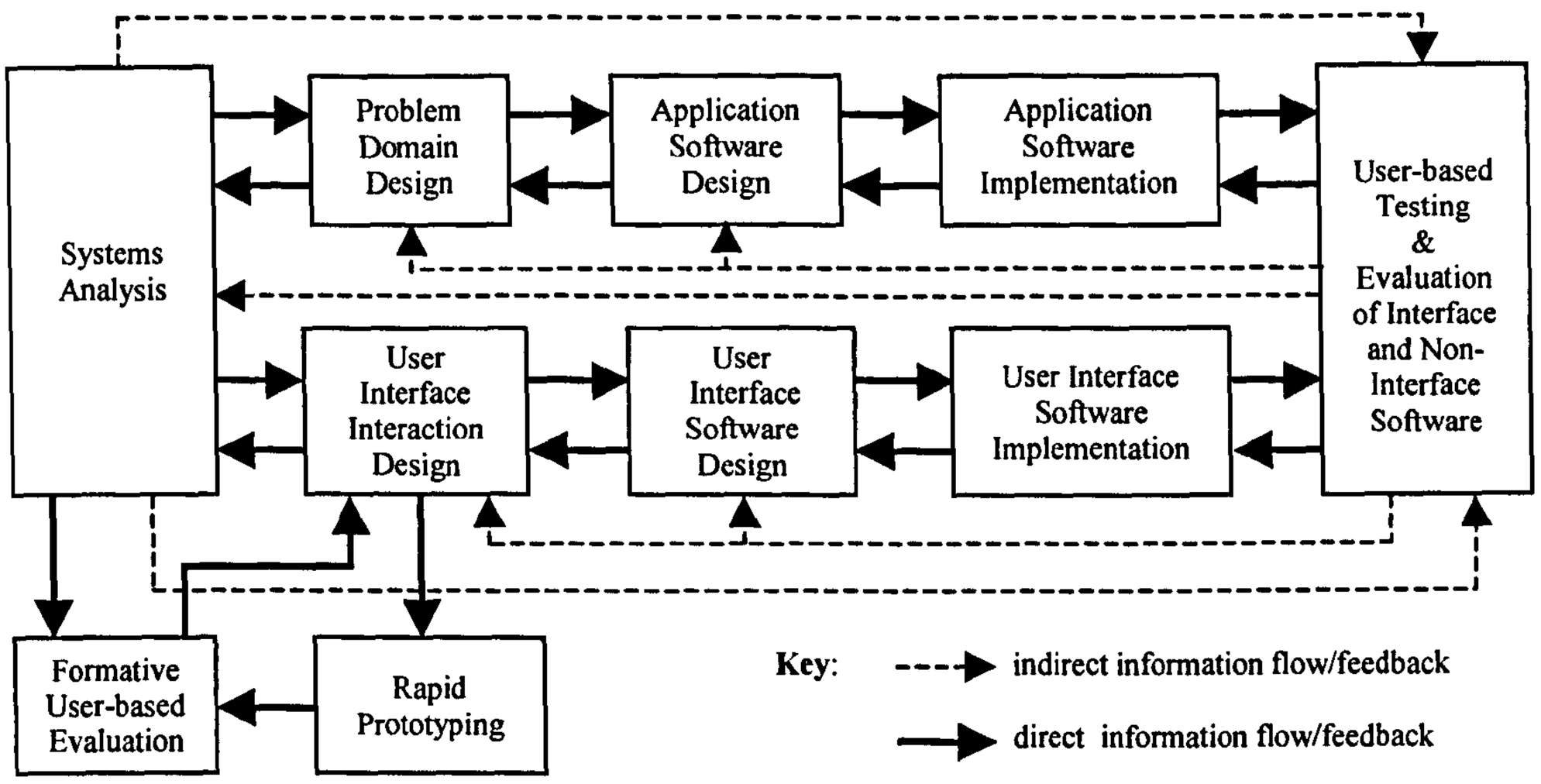


Figure 3.17 -*The Hix & Hartson software development process* (Hix and Hartson, 1993)

Figure 3.17 shows the complete software development process proposed by Hix and Hartson (Hix and Hartson, 1993). The boxes highlight the main steps involved in the development process and the arrows, the various links or communications between the steps. Hix and Hartson suggest that the development of the user interface and that of the underlying application exist as two parallel sub-components within the overall development process, linked by the systems analysis and the user-based testing phases. They make no claim as to who performs each of the steps, or indeed the relative timing of the sub-processes and their absolute sequencing. Instead they propose the process as a *possible* collection of steps to be performed in order to achieve the development of a software application.

Of the sub-processes listed, perhaps the least familiar is the one termed 'problem domain design'. This is the process of letting the designer(s) know what the software is supposed to do. It is abstract and independent of the software design and implementation, and allows the user interface designer(s) and developer(s) to work in tandem (but relative isolation if required) with the developers of the underlying functionality. The diagram highlights the need for integration at both the systems analysis step and the testing stages.

Hix and Hartson state that in reality, there is a distinct need for vertical channels of communication between the parallel processes of user interface and underlying functionality design/development (Hix and Hartson, 1993). However, they suggest that such channels of communication will most likely be *ad hoc*, and will develop as and when they are required during the course of application development. As such, the above process should not be considered to be lacking due to the absence of explicit vertical communication channels.

Hix and Hartson recognise that the 'clean' software distinction between the user interface software and that of the rest of the system may in reality not be as clean (see discussion in chapter 2), but stress that their model is not an attempt at a software engineering method *prescription.*

3.4.3.2 The Data Gathered

Isolating the user interface-specific components of the Hix and Hartson software development process generates the sub-process diagram as shown in Figure 3.18 below.

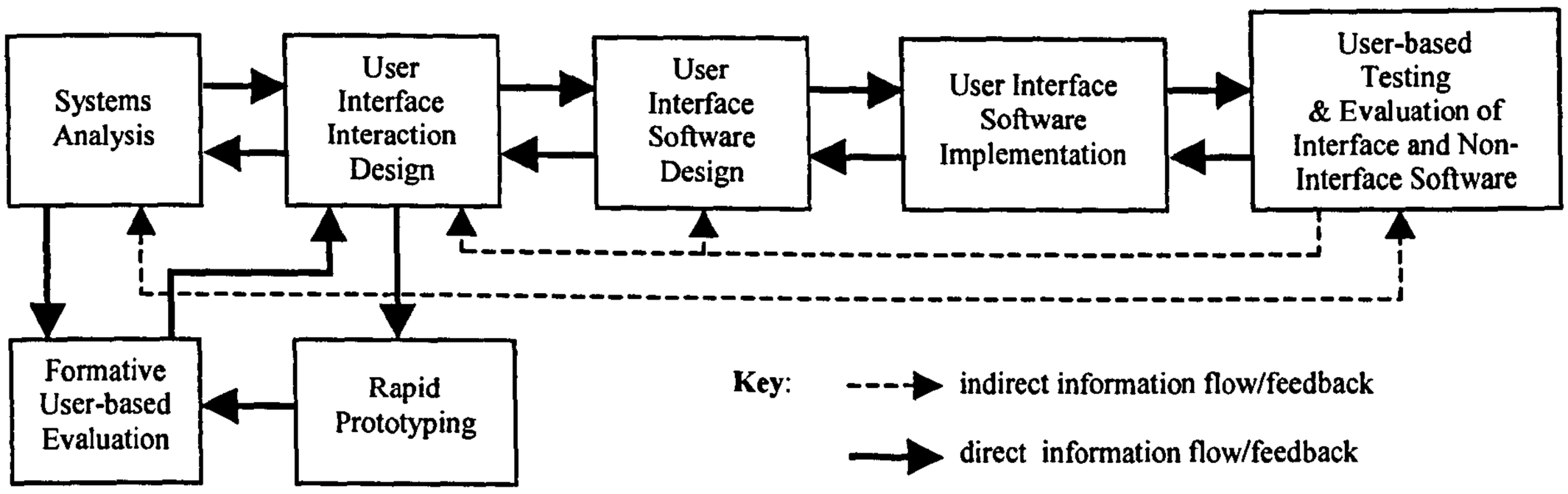


Figure 3.18 – *The user interface sub-process of the Hix & Hartson software development process* (Hix & Hartson, 1993)

Each of the development steps shown above was itemised within a non-ordered list of user interface life cycle stages in section 3F of the questionnaire (see Appendix A). The respondents were asked to highlight which of the steps are typically employed during user interface development within their own organisation, and to indicate (where appropriate) a sequence number for each step. The steps were listed in a random order to prevent indirectly influencing the responses to this question, and explanation was provided where it was considered necessary - as shown in the following list:

- *rapid prototyping;*
- *user-based evaluation of the interface:*
  - (assessment of the user interface is the responsibility of the user who is free to provide whatever feedback he/she deems appropriate – i.e. a loosely structured process);
- *formative user-based evaluation:*
  - (evaluation of the user interface *as it is being developed* – the results of the evaluation being fed back into the design at early stages);
- *systems analysis;*
- *user interface software design:*
  - (similar to design of any other software – system-centred);
- *user-based testing of the interface software:*
  - (the user is given specific structured tasks to do which are monitored by the developer to assess specific aspects of the interface design);
- *user interface software implementation;*
- *user interface interaction design:*
  - (design of the user actions, feedback, screen appearance, user tasks, functionality sequencing, content, information access, design of interface objects, screen layout and interaction styles – user-centred).

Respondents were free to annotate the listed steps and to introduce additional (non-listed) steps to suit the individual practice within their organisation. It was hypothesised that the characteristics of the application domain would influence the development process, and so the following results are discussed on a per domain basis. Unfortunately, a number of respondents failed to indicate sequence numbers for the steps they performed. These results will be discussed but no attempt will be made to model their overall development process. In general, the respondents did not indicate the nature of communication between the various components and therefore some assumptions had to be made in order to represent this aspect of the development process in the following models. It was not the aim of this investigation to conclusively identify the sequential ordering of, and communication mechanisms between, the sub-components of practical user interface development. Instead, the principal concern was to illustrate the sub-components that make up typical practical development processes in order to determine functional provision that may be demanded of a UIDT. Hence, issues of sub-component sequencing and communication mechanisms have not been scrutinised.

After the domain-based discussions, this section will conclude by highlighting similarities between the *real* use of development processes and that proposed by Hix and Hartson.

3.4.3.3 Database Application Domains

Within this domain, five respondents highlighted their development process. However, only two indicated the order in which the particular steps were carried out. These are modelled in Figure 3.19(a) and (b).

Both processes adopt the same development components, differing only with respect to the order in which the steps are performed. The first model suggests that iterative prototyping is used to evaluate the design while continuously moving towards a more 'concrete' final version. The second model indicates that prototyping has a more extensive rôle in the user interface development process - it is used to assist at every stage of design and development. Respondents indicated the use of all types of prototypes (see section 3.4.3.9) which is consistent with the pivotal rôle prototyping plays within these models (especially the second one).

Although neither of these models match the Hix and Hartson model, with the exception of the separate user interface software *design* stage, they both comprise the stages suggested by Hix and Hartson (Hix and Hartson, 1993). The absent stage could indicate that: either companies do not explicitly design the software architecture for the user interface (as they would for the underlying functionality) and instead evolve the software structure from the prototypes; or they do not consider this as a separate task, instead including it in the implementation stage. Further investigation would be required to clarify the situation.

Although the sequencing of stages in both models differs from the Hix and Hartson model, the latter was not intended as a sequence model and so this is of little significance (Hix and Hartson, 1993). The modelled processes do, however, loosely follow the same sequence – systems analysis followed by design and implementation and then testing. They indicate extra or alternative communication links between the various stages that are not present in the Hix and Hartson model – for example, links between the prototyping stage and all other stages rather than links just to formative evaluation and interaction design as modelled by Hix and Hartson (Hix and Hartson, 1993).

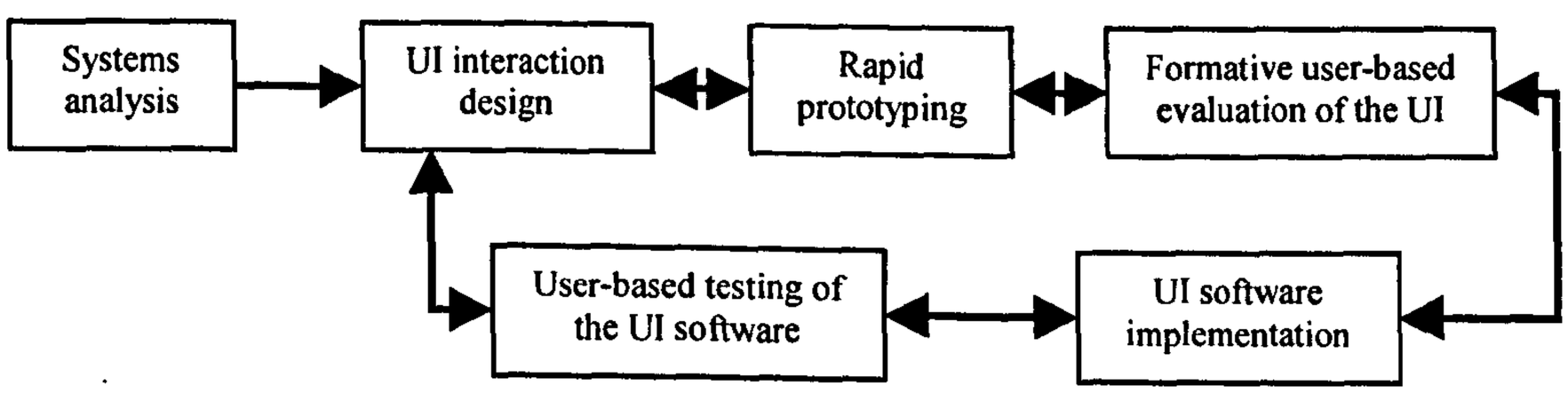


Figure 3.19(a) – *A development process for database application development*

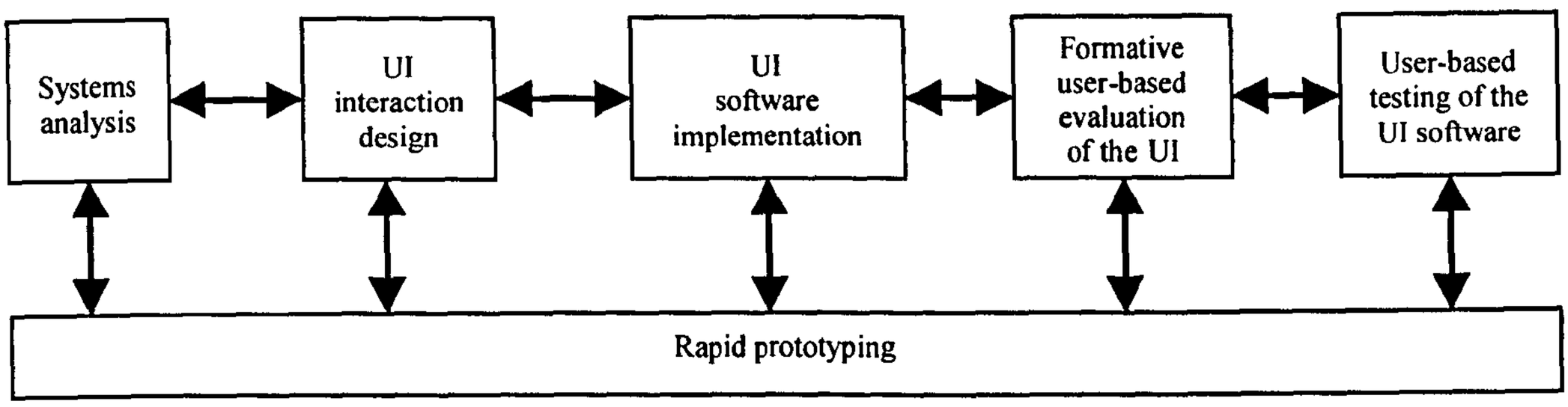


Figure 3.19(b) – *Another development process for database application development*

Although respondents for the remaining companies in this domain indicated that they perform all of the suggested stages, there is a possibility that the respondents answered according to what they thought they *should* be doing rather than what they *actually* do. This caveat applies to all further discussion regarding development processes.

3.4.3.4 Manufacturing Application Domains

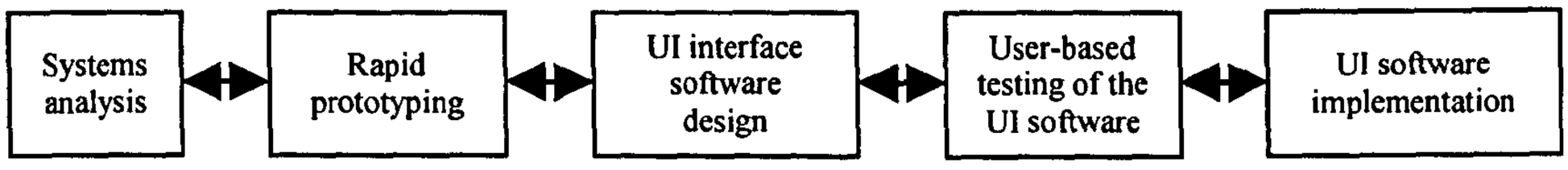


Figure 3.20 – *A process for manufacturing application development*

The user interface development process adopted by the respondent within the manufacturing control domain is shown in Figure 3.20. It demonstrates an unusual or unintuitive component sequencing that may be due to misinterpretation on the part of the respondent (regarding the meaning of the component tasks), or may be intentional. If, for example, the respondent typically centres user-based testing of the user interface software around the prototypes, then it is conceivable that the final version of the software may only be completely implemented after all testing is done and design decisions made. In essence, this model is geared towards postponing the most inflexible (or costly to change) aspects of the development to the end of the process. Prototypes are pivotal to this development process - as suggested by fact that they are used to facilitate the design of the user interface following systems analysis, and as confirmed by the fact that this organisation makes extensive use of evolutionary prototypes during application development (see section 3.4.3.9).

This model confirms the use of five of the steps identified by Hix and Hartson (Hix and Hartson, 1993) and illustrates that the nature of the prototypes used can influence step sequencing.

3.4.3.5 Software Tool Application Domains

Two respondents representing companies in this domain identified their development processes, as modelled in Figure 3.21(a) and (b).

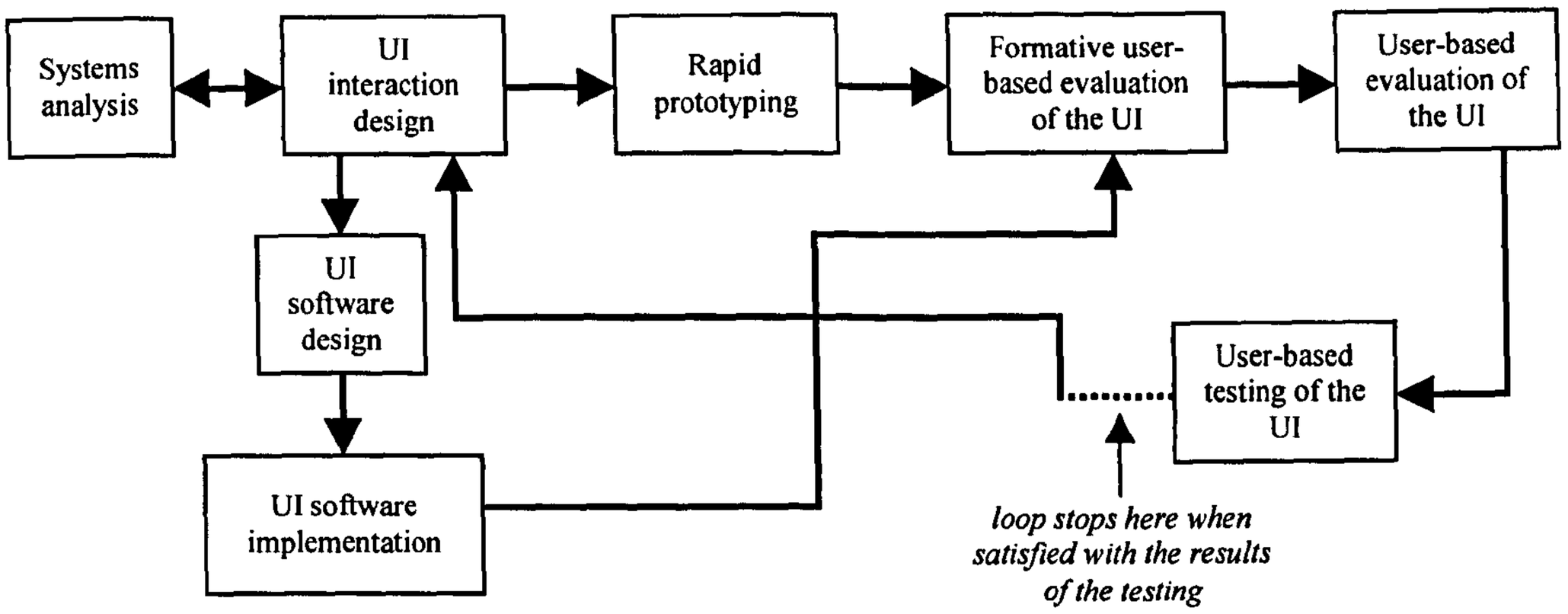


Figure 3.21(a) – *A process for software tool development*

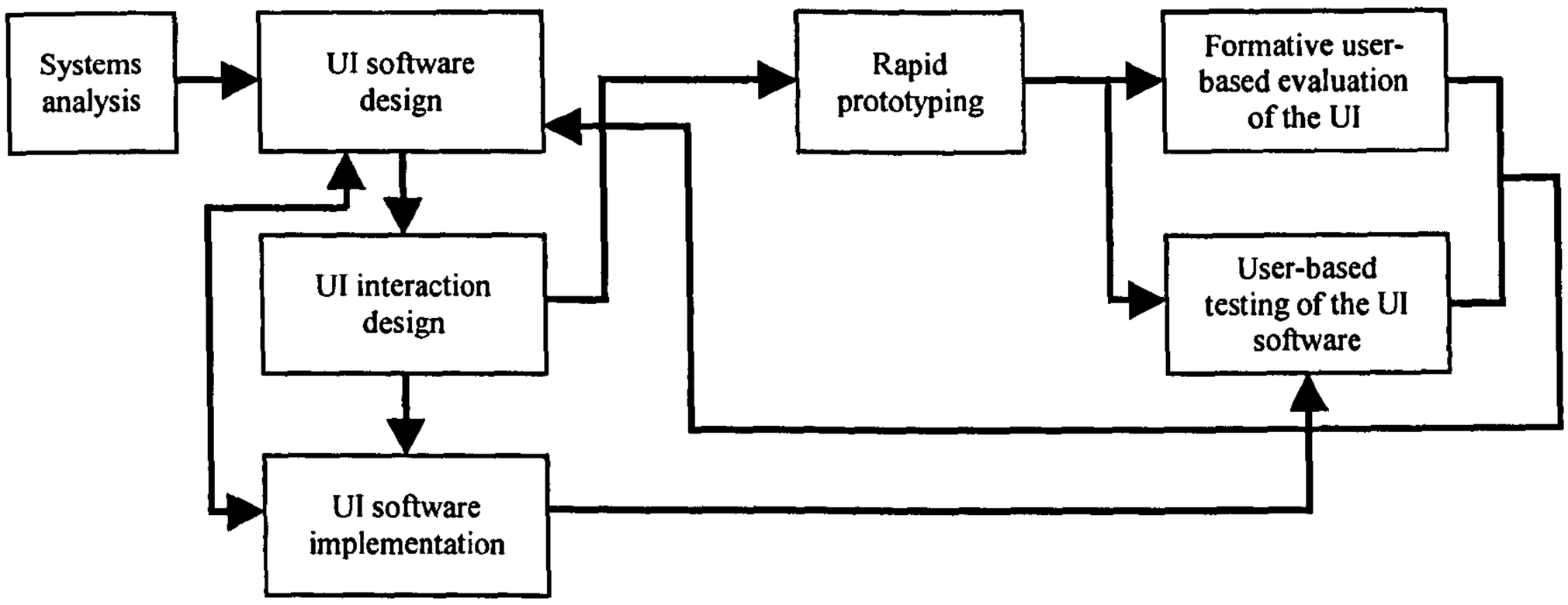


Figure 3.21(b) – *Another process for software tool application development*

With the exception of 'user-based evaluation of the user interface' in the second model, both use all the development steps proposed by Hix and Hartson. Although they are not identical to the Hix and Hartson model, tracing through the development paths in these models highlights the fact that, despite it not being the intention of Hix and Hartson to suggest a definitive sequence, the implicit sequence within the Hix and Hartson model is very close to the actual development sequence used by these respondents. Within the scope of this survey, the companies developing applications within this domain follow similar working practices that reflect the use of evolutionary prototypes, and are well suited to their stated use of global and local prototyping (see section 3.4.3.9).

3.4.3.6 Educational Application Domains

Consider now the development processes adopted for educational software. Of the respondents in this category, two explicitly indicated their working practice and a third just commented on his practice. Figure 3.22(a) and (b) shows the identified development processes.

Although both processes make use of all of the Hix and Hartson steps, they 'link' them differently (Hix and Hartson, 1993). The first of the models, which has a bi-directional

linear (or sequential) structure, places emphasis on the prototype for the user interface design and its subsequent evaluation and testing. In this model, like a number of the previous models, implementation of the user interface comes after all the evaluation and testing has been done on the prototype. Although not mentioned in the response of the company, it is reasonable to assume that further iterations of the development process (in whole or part) are performed until the user interface is completely implemented and tested.

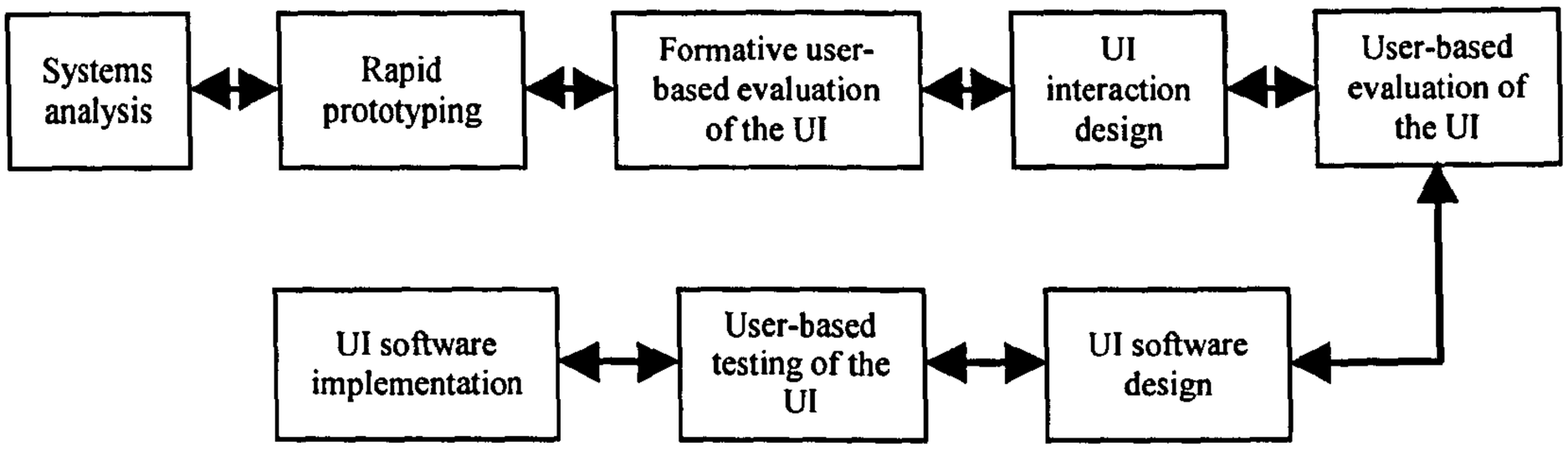


Figure 3.22(a) – *A process for educational software development*

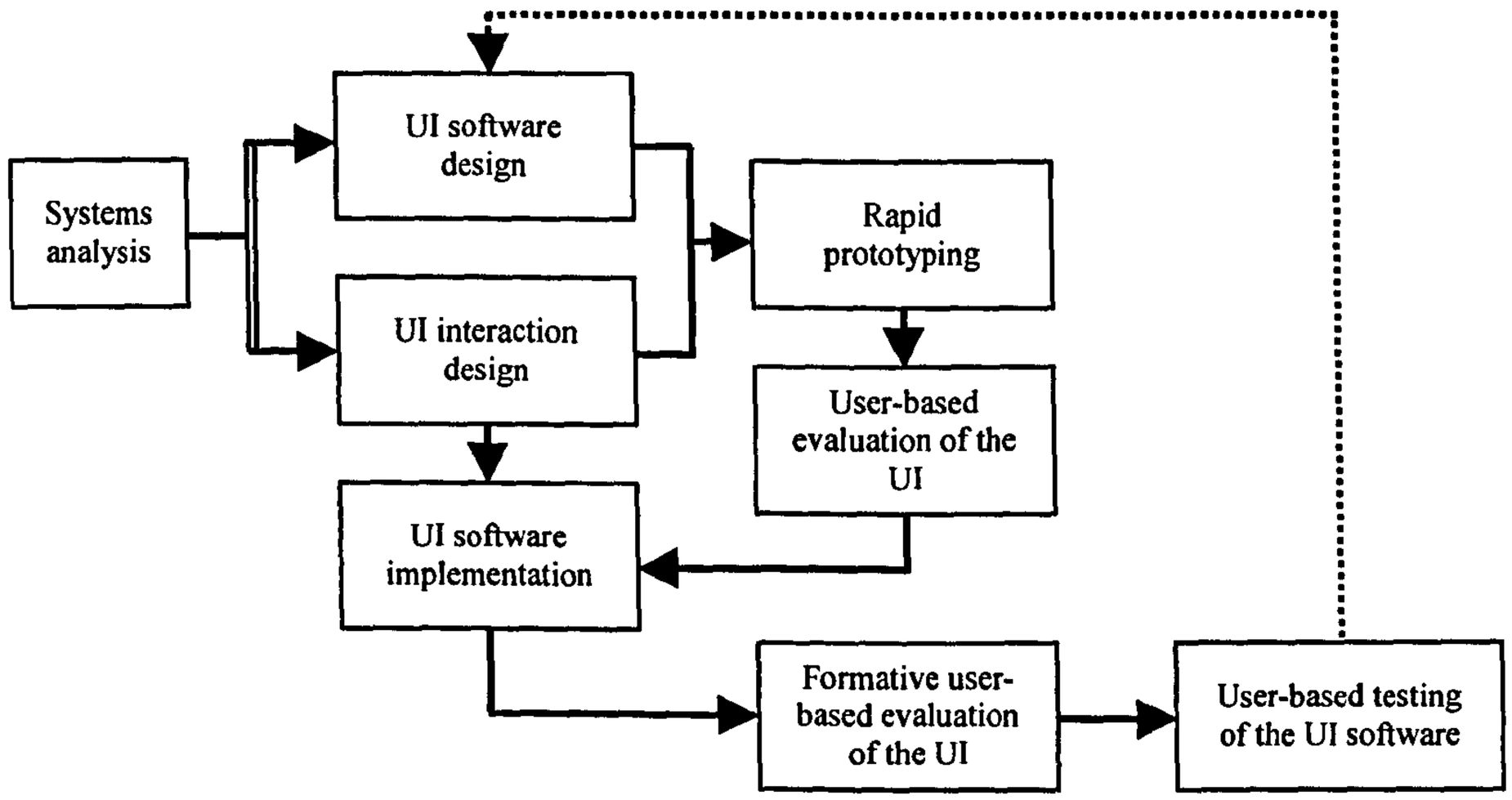


Figure 3.22(b) – *Another process for educational software development*

The second model shows a more complex communication mechanism and sequencing - the user interface interaction design and the user interface software design take place as parallel sub-components of the overall model. The two sub-processes feed into the development of a prototype, and thereafter both the prototype and the current level of implementation are tested and evaluated using a variety of evaluative techniques. The results of the evaluation efforts are then fed back into the development loop, hence suggesting that this user interface development process is iterative.

The third respondent did not identify a process based on the steps indicated. Instead, he explained that the organisation did a lot of repetitive work, where a new system was essentially a slight variation on one of their previous systems. The company therefore simply produce a prototype based on a previous system and subsequently employ it for user-based testing. The results of the testing are then fed back into the prototype, which is

iteratively evolved (with further testing) into the final system. This illustrates that companies can mould (or truncate) their development process to account for the context in which they are working. In this example, the company had already completed the full development cycle for previous educational applications and so was able to bypass a number of development steps such that they only had to alter pre-existing systems and test the altered versions in order to produce new systems.

3.4.3.7 Information Systems Application Domains

Within this domain, one respondent identified his development process and another indicated some of the steps performed but gave no sequencing information - the responses were essentially incomplete. The (partial) model for the former can be seen in Figure 3.23.

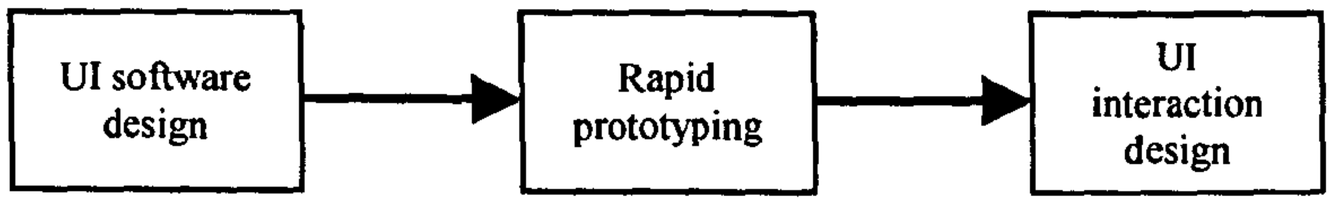


Figure 3.23 – *A process for information system development*

It is unlikely that the above process constitutes an entire typical information system development cycle. Furthermore, the suggested ordering of the constituent steps does not reflect a developmentally practicable or logical sequence of action - for example, within most processes discussed previously prototypes are the outcome of interaction design (where interaction design is an independent component).

Although the sequencing of the steps cannot immediately be explained, one possible (albeit tentative) explanation for the lack of completeness is that the remaining steps may be performed by independent sections within the organisation and the results fed into the central development steps shown here. For example, the development team may be given the results of the systems analysis, which they use to design and thereafter implement a user interface. They might then pass this onto a user testing department, which in turn may feed the results back to the design and implementation team. If this is the case, then the design and development team may know nothing about the mechanisms of the steps performed before or after them and as such, can only indicate their own components. Although hypothetical, this suggests that development of applications within some organisations *may* be compartmentalised, with development being physically or organisationally separated. A general overview of the entire company would be required in order to ascertain whether a more extensive process is followed.

The respondents that merely indicated the steps performed used five of the steps listed by Hix and Hartson with strong emphasis on testing.

3.4.3.8 Real-Time Application Domains

One development process for real-time systems was identified, and is shown in Figure 3.24. Failure to explicitly represent user interface interaction design suggests that this activity is incorporated within the step entitled 'rapid prototyping'. Similarly the absence of explicit representation of a user interface software design activity suggests that this is incorporated within the user interface software implementation component.

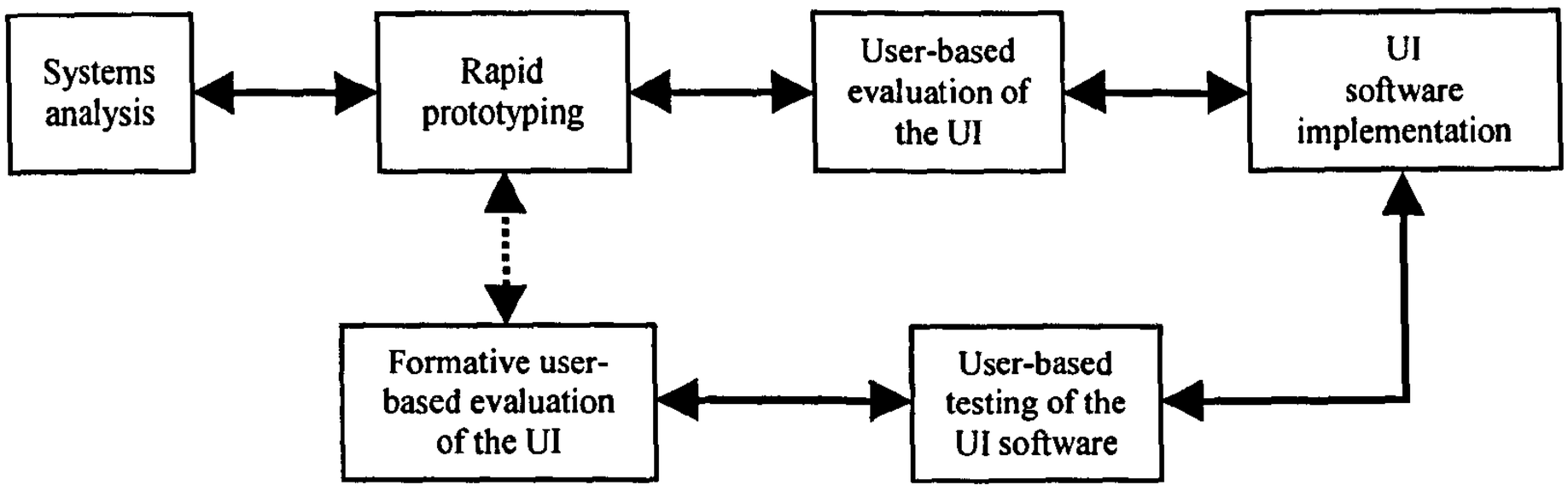


Figure 3.24 – *A process for real-time application development*

3.4.3.9 Use of Prototypes

Respondents were asked to provide information, where applicable, regarding the nature of the prototypes used in their development processes. Prototype styles range from paper-based prototypes to revolutionary prototypes (for example, Hix and Hartson, 1993) and include:

1. *Horizontal* – wide range of features are incorporated, but none are implemented in detail;
2. *Vertical* – only a small number of features are prototyped but they are implemented in detail;
3. *Global* – covers much of the system giving the user a good idea of the final product. These can be both horizontal or vertical;
4. *Local* – prototype of a single specific detail (often used to evaluate design alternatives). These are similar to vertical prototypes but can never be horizontal;
5. *Evolutionary* – a single prototype which becomes increasingly sophisticated during design iterations until it constitutes the final artefact;
6. *Revolutionary* – prototypes which are 'thrown away' after they have served their evaluative, demonstrational, or testing purposes, and from which lessons are learned and fed into the next (new) prototype;
7. *Paper-based* – paper-based sketches of parts of the artefact under design.

Figure 3.25 shows the number of companies which use each of the various forms of prototype.

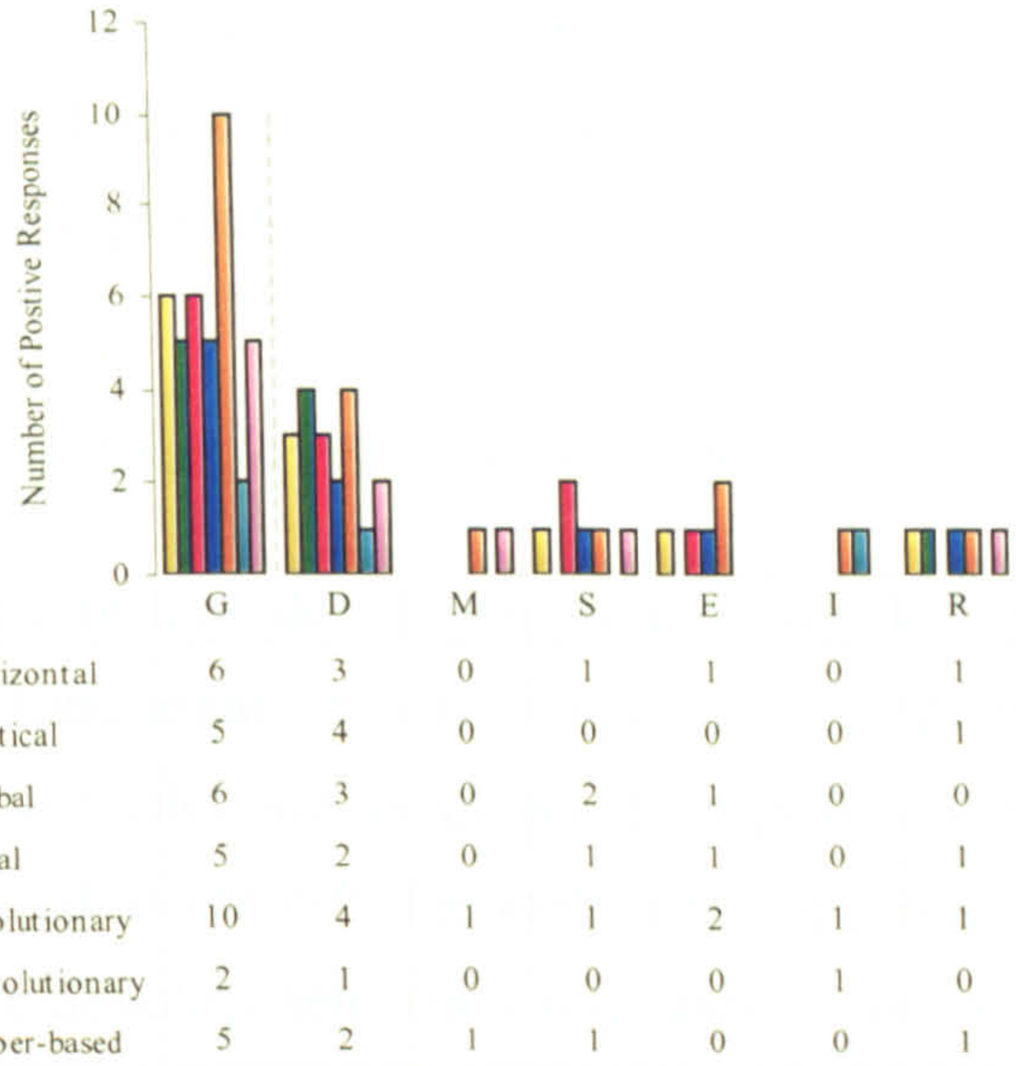


Figure 3.25 - *Use of prototypes*

The most popular style of prototype is evolutionary. Companies that use local prototypes also make use of horizontal prototypes. Since, according to the listed definitions, the two are mutually exclusive this suggests that these companies make particular use of prototypes for specific elements of design/development. The extensive use of prototypes reflects the adoption of iterative prototype-based design and development and is consistent with the developmental strategies outlined previously.

3.4.3.10 Use of HCI Specialist in the Design of the Development Process

Domain	HCI expert involved in design of development process
G	0
D	0
M	0
S	0
E	0
I	0
R	0

Table 3.12 – *Use of HCI specialist in design of the development process*

To determine the influence of (or consideration given to) HCI during the establishment of software development life cycles, respondents were asked to indicate whether HCI experts were involved in the design of their development processes. Table 3.12 shows that, within the scope of this study, HCI specialists were uniformly absent from the design of the software development processes. Despite this, some of the practical development

processes are not that far removed from documented processes that were devised by experts in the field of HCI (Hix and Hartson, 1993). One explanation given for the structure of the development process within a particular company was that it was 'common sense'.

3.4.3.11 Conclusions About User Interface Development Processes

It is anticipated that user interface development activities will potentially inform one part of the complete set of functional requirements that comprise the criteria upon which UIDTs will be compared for selection given an appropriate evaluation methodology. This investigation was conducted to identify the *actual* (as opposed to textbook) activities used in *practical* user interface development such that they could be included in the framework of a new UIDT evaluation methodology and thus make the evaluation methodology and framework more amenable to the current needs of industrial software developers.

There are a number of conclusions that may be drawn from the investigation[50]. There is no immediately obvious re-occurrence of development process structure across the different domains. Indeed, it is only for database applications and software tool development domains that some similarity exists between the processes typically adopted by different companies developing within the *same* domain. These same processes most closely resemble the Hix and Hartson model (Hix and Hartson, 1993). The more complex the interface to an application, the greater the degree of iteration present within the development process.

All of the constituent steps suggested by Hix and Hartson (Hix and Hartson, 1993) were used by one or more of the processes modelled. Least practised of the activities was user-based evaluation of the user interface - the companies surveyed tended to practice more user-based testing. This may represent a bias on the grounds of the relative costs involved to perform each, or it may represent differences in interpretation of the meaning of 'evaluation' as opposed to 'testing'.

The most commonly performed activity was rapid prototyping. This suggests that industrial user interface design and development is predominantly prototype-centric and that a UIDT will therefore need to provide sufficient prototyping facilities for it to rank highly during evaluation of such tools. The point at which rapid prototyping was used differed between processes. It ranged from use at the beginning for design purposes to

[50] Given the relatively small scale (albeit detailed nature) of this survey no attempt is made to assign statistical significance to the reported observations and conclusions.

being pivotal throughout the entire development process. This suggests that there may be some degree of variation between the specific prototyping facilities required in UIDTs.

None of the respondents included additional development activities in their overall processes. This suggests, within the scope of this survey, that the user interface development tasks outlined by Hix and Hartson capture the essential activities performed in industrial user interface development (Hix and Hartson, 1993). Hix and Hartson stated that they were making no attempt to impose a sequencing on the steps in their process and that they were unsure as to the rôle and placing of both rapid prototyping and formative user-based evaluation within the development cycle (Hix and Hartson, 1993). Together with the observations reported in this survey, this suggests that the development process of Hix and Hartson can be considered a template user interface development process, outlining potential activities and suggesting a possible sequencing strategy. Based on this template developers re-order, combine, remove, and link the components to tailor the process to meet the needs of their organisation and application domain. Aside from the practical guidance it affords to the development of user interfaces, the contents of the template have been shown to be consistent with, or sympathetic to, industrial practice. They may therefore be used to inform the developmental process-related criteria within a new UIDT evaluation framework.

3.4.4 Software Support Used

Section 3E of the questionnaire investigated the use of software support for application development within the identified domains. The questions were divided into four categories relating to: (1) no use of software support (i.e. unassisted programming in any given programming language); (2) use of pre-provided libraries of routines/components; (3) interface builders; and (4) application software development environments. On the basis of the findings of the Myers and Rosson survey (discussed in chapter 2) it was hypothesised that individual projects would make use of tools within more than one of the given categories. The investigation therefore aimed to illustrate the rôle played by each type of support within the overall development of an application. To inform the development of a new UIDT evaluation method, the questions were also designed to determine the reasons behind the choice of each of the tools – what kind of evaluation was done prior to purchase, the level of satisfaction with the tools etc.

3.4.4.1 No Software Support Used

Table 3.13 shows the number of respondents who claimed to make no use of software support for some or all aspects of user interface development. Where applicable, respondents were asked to approximate the percentage of the total application code which

was developed without the use of software development support. Where available, these figures are shown in the second column of Table 3.13.

Domains M and S are the least reliant on software support - the respondent in domain M did not rely on software support for a large proportion (90%) of the application code.

Domain	No software support used	% of total code
G	8	60 – 90
D	3	60 – 90
M	1	90
S	2	
E	2	
I	0	
R	0	

Table 3.13 – *No software support used*

The development of applications that include the most varied input and output media rely most heavily on software support (for example, domain I) - although they do generate some application code without software assistance. Applications in domain R are not programmed directly in any given programming language - instead, their user interface development makes substantial use of the remaining categories of software support.

3.4.4.2 Use of Pre-provided Libraries of Routines/Components

Domain	Pre-provided libraries of routines/components	% of total code
G	9	10 – 35
D	3	15 – 35
M	0	
S	2	
E	3	
I	0	
R	1	

Table 3.14 – *Use of pre-provided libraries of routines*

Table 3.14 shows the number of respondents who reported making use of pre-provided libraries of routines or components – ranges of total code percentages are given where available. Reasons given for selection of the various libraries ranged from their ease-of-

use and flexibility, to the existing knowledge base of the personnel within the organisation. No respondent cited the nature of the application as the primary reason for selecting the library.

Pre-provided libraries are used across all reported applications within several domains (for example, S, E, and R). In domain D they are used by approximately two thirds of the respondents but account for little of the overall application code. This suggests that libraries are used to access specialised components rather than for general application development.

3.4.4.3 Use of Interface Builders

Respondents were asked to indicate whether they used interface builders during user interface development. Where applicable, they were asked to further specify: whether demos of the tools were received prior to purchase; whether any tool evaluation had been performed; an estimate of their perception of the percentage of the overall interface builder functionality that was actually used; whether they could identify missing functionality (that is, functionality that they required but that was absent from the tool); to state whether they were considering changing the software; and to indicate whether they reviewed the software support on a frequent basis (see question 3.15(c)). Table 3.15 shows their responses - figures for percentage of total code and percentage of functionality use are given where available.

Domain	Interface Builders Used	% of total code	Demos given	Evaluation done	% functionality used	Missing functionality identified	Changing the software	Frequent software review
G	7	20	3	2	60 – 90	3	2	4
D	2	20	0	1	60	1	1	2
M	0	n/a	n/a	n/a	n/a	n/a	n/a	n/a
S	2		1	0	90	1	0	0
E	2		1	1		1	1	1
I	0	n/a	n/a	n/a	n/a	n/a	n/a	n/a
R	1		1	0	90	0	0	1

Table 3.15 – *Use of interface builders*

Indicative costs relating to the use of these tools were in the region of £10,000. Interface builders are adopted across four out of the six domains and are used by approximately half

of the organisations, but account for little of the total code in each case. On average, less than half of the companies which reported use of interface builders had been given a demo of their selected tool prior to purchase. Little or no evaluation was performed by the respondents prior to purchase. In general, and within each domain, half of the users of the interface builders were able to identify missing functionality. A slightly higher proportion claimed to review the software frequently, and approximately one third of the respondents using interface builders were considering changing their software support.

3.4.4.4 Use of Application Development Environments

Respondents were asked the same series of questions for application development environments as listed for interface builders. The results are shown in Table 3.16.

Domain	Application development environments used	% of total code	Demos given	Evaluation done	% functionality used	Missing functionality identified	Changing the software	Frequent software review
G	12	20 - 100	4	3	60 - 100	6	4	5
D	4	30 – 100	1	1	60 –100	3	1	2
M	1	90	0	0	100	0	1	0
S	1	20	0	0	70	0	0	0
E	3		1	1	100	2	1	1
I	2	40 – 100	1	1	70	1	1	1
R	1		1	0	80	0	0	1

Table 3.16 – *Use of application software development environments*

Reasons given for the choice of tool included being customer driven, the industry standard nature of the tool, and their prototyping ability. Installation costs for these tools ranged from £3000 to £500,000 (the size and profitability of the organisations played a major rôle in the expense which they could apportion for the purchase and installation of any one given tool). A number of respondents provided additional commentary regarding their use of this type of software support. Such comments included that the organisations had to write add-ins to the tools to compensate for lacking functionality and that, where performed, evaluation was not systematic – it was just an *ad hoc* comparison of various tools.

Application development in every domain uses development environments, often for a large percentage of the overall application code. Despite this, only a small proportion

(approximately one third) of the organisations who use these tools had been given a demo of the tool prior to purchase and only a quarter had done any evaluation. The percentage of tool functionality that was actually used was high across all domains, suggesting that the tools are being used close to their perceived functional limit. For a number of domains over half of the respondents could identify missing functionality or hindrances in the tools. A number of respondents across most of the domains are considering changing their software tools. Most review such tools at least every two years, with many indicating that their process of software review was continuous, especially regarding recently acquired application development environments.

3.4.4.5 Satisfaction With Current Tool Evaluation Methods

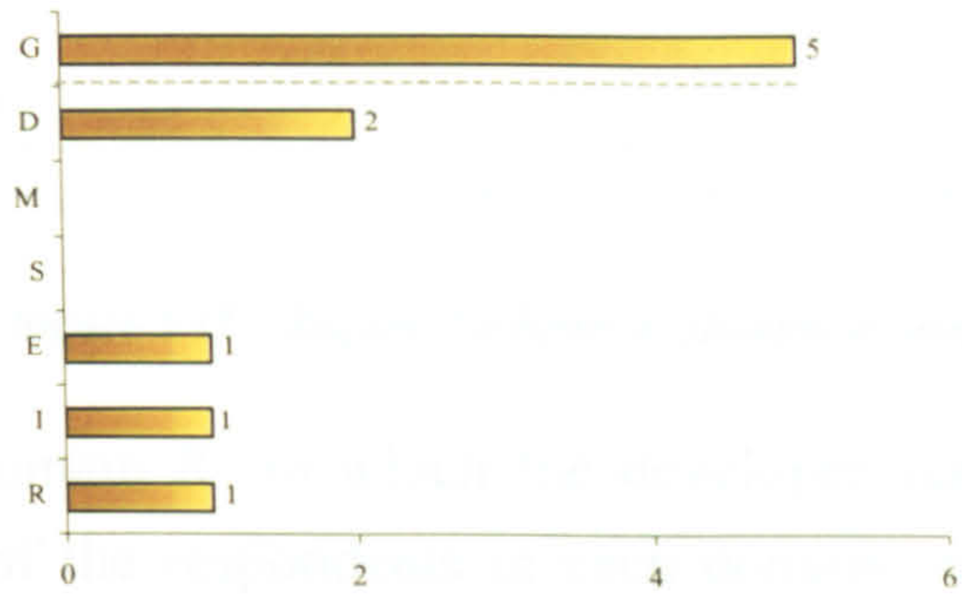


Figure 3.26 - *Respondents satisfied with current tool evaluation methods*

Respondents were asked to express their satisfaction with current evaluation methods for software support tools. Figure 3.26 shows the number of respondents, across each application domain, who are happy with current available evaluation methods.

With the exception of domain R (in which there was only one respondent), the majority of respondents in each domain are not satisfied with current available evaluation methods. The respondent in domain R performs no evaluation and therefore his response can be interpreted in two ways: (1) it could be that he does not uphold the need for UIDT evaluation in principle; or (2) since he does not perform evaluations he may not have had cause to give due consideration to the available mechanisms, and therefore may not have been in an appropriate position from which to pass judgement. Further investigation would be required to clarify this situation. Less than half of the respondents were happy with the evaluation methods available to them at present and respondents who performed no evaluation (that is, domains M and S) seemed to be the least enamoured with the current methods. This raises the question of whether the methods were not used because they are: inadequate; disliked; inaccessible to the respondents; or because the respondents did not know about them in the first place due to their lack of widespread international industrial adoption. Comments from the respondents to this questionnaire, the interviewees in the original pilot interviews, and responses from subsequent discussions with industrial

software developers (see chapter 7) suggest it is a combination of all four; many are emphatic that their needs are not currently being met by what is presently available.

3.4.4.6 Request for Better Evaluation Methods

Respondents were asked to indicate whether they would like to see the development of a new UIDT evaluation method. Figure 3.27 shows the number of positive responses.

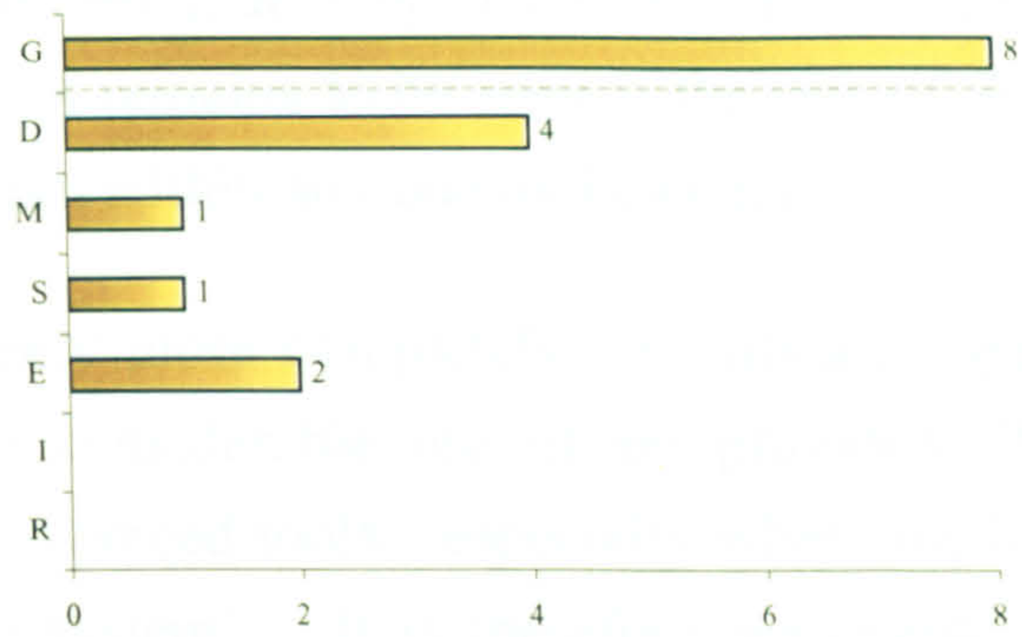


Figure 3.27 – *Request for better evaluation methods*

With the exception of domain R (in which the developer stated that no evaluation was done) no less than half of the respondents in each domain requested a better evaluation method – especially those who stressed least satisfaction with current methods. Assuming that their desire for a new evaluation method is indicative of their intention to use one (which therefore suggests attempts to find or develop such a method in the past) this implies that the respondents were in fact aware of existing evaluation methods and/or had failed to develop a suitable strategy themselves. This suggests that there is a definite need for an alternative means to evaluate UIDTs that enables software developers to select the correct tool according to their needs.

3.5 Relationships Among the Results Obtained

This chapter has thus far presented the collected survey data with commentary on application or administration level relationships. This section will discuss: (1) the relationships among the characteristics of application domain and the software support tools used; and (2) the relationships between the software engineering methodologies and the software support tools used, taking into consideration the nature of the application domain[51].

[51] No attempt will be made to generalise the observed relationships beyond the scope of the survey.

3.5.1 Relationships Among Application Characteristics & the Use of Software Support Tools

If the characteristics of the specified applications are considered in conjunction with the information related to the use of software development tools, it is possible to make a number of observations regarding the relationships between these elements.

The size of the application being developed does not influence the type of software support tool purchased, although it remains to be seen if the size of an application influences the choice of specific tool from within any one tool category.

Safety-critical systems rely more completely on software support than other application domains. They make considerable use of pre-provided libraries of components (or routines) and the more advanced tools - especially where the locus of control is classified as 'operator serving the system'. It is therefore reasonable to assume that the safety-criticality of the system being developed influences the choice of software support tool type.

Large proportions of some of the applications that promote user autonomy are programmed directly in a given programming language without the support of third-party software. This does not mean that the project teams make no use of software support tools for the development of the entire application, but rather that they choose to program some aspects directly without the assistance of a higher level support tool. Developers building applications that constrain user autonomy use tools in all categories as well as coding directly in a given programming language. These observations suggest that the locus of control within a particular application can strongly influence the choice of type and extent of use of software support.

The development of applications designed to run on multiple platforms relies more heavily on advanced tools than does the development of single platform applications. Whether this is a consequence of the complexities involved with trying to make applications platform independent or is dictated by some other factor, it suggests that the intended portability of an application will influence the choice of software support tool type.

Although the data intensity of an application does not influence the choice of software support, it is likely that the intended means of storing that data will determine certain functional requirements of the selected software support. In contrast, the interaction intensity of applications corresponds to considerable use of application development environments, and is therefore potentially influential when choosing software support.

Development of applications which include a wide range of input devices and output media makes use of software support across the range of identified categories (including direct coding). The one notable exception to this being real-time systems development which does not include direct programming in the underlying programming language. This indicates that the nature of the input devices and output media for a given application will determine a number of the functional requirements driving the selection of appropriate software support.

There is no obvious connection between the cognitive load of applications and the tools used to develop them. It would therefore be interesting to investigate whether (all other factors being equal) changing the type of software support used would affect a significant enough change on the resultant user interface such that it, in turn, would return a different cognitive load estimation. Development of applications that were designed to cope with the highest critical loads either makes most extensive use of application development environments and/or makes no use of software support at all. In contrast, development of applications with the lowest critical loads makes use of the range of types of support. This suggests that the anticipated critical load of an application is influential in determining appropriate software support.

The observations outlined above highlight the fact that many application characteristics strongly influence the choice and subsequent use of available software support tools and must therefore be catered for in a new UIDT evaluation framework.

3.5.2 Relationships Between Software Engineering Administration & the Use of Software Support Tools (Accounting for Application Characteristics)

Examination of the identified software engineering processes in connection with the use of software support tools reveals a number of relationships. Where project teams exceed the average number of members (that is, 5), applications are developed using the full range of software support. In general, such applications are not safety-critical but they incorporate the widest range of input devices and output media. Three possible conclusions may be drawn from these observations: firstly, user interface complexity is directly proportional to the size of the overall project team; secondly, as mentioned earlier, the range of input devices and output media is directly proportional to the range of software support that is used on any one application; and thirdly, the size of the project team is directly proportional to the range of software support that is used by the team. This suggests that there is a loose cyclic dependency between these components - that is, the overall size of a project team is influenced by the nature of the application and in turn influences the range of software support used to develop the application.

In contrast, there is no immediately obvious relationship between the size of the HCI sub-team and the range of software development tool types used. However, the fact that the more safety-critical applications tend to demand larger HCI sub-teams suggests that there is a connection between the size of the HCI sub-team and the characteristics of the application.

The range of training within a team is directly proportional to the range of types of software support tool used - organisations which include graphic designers in their HCI teams use all tool types. In general, the most extensive use of no tools or the less advanced tools coincides with teams in which there are members with no computing science training. The data collected in this respect is, unfortunately, insufficient to determine which members of the teams are using which tools and therefore no conclusion can be drawn in this regard at this stage.

The majority of companies that do not involve everyone throughout the entire project make use of tools across the whole range of software support, and tend to be non-safety-critical. Furthermore, these companies include a wider range of experience within their HCI teams. It would be interesting to further investigate whether specific team members are not involved throughout the duration of a project on account of their specialisation - that is, to determine whether they serve a consultant rôle within the organisation and their time is therefore divided across multiple projects at once. However, the observations suggest that the HCI experience of the team members and the manner in which they are deployed during the course of a project can potentially influence the type of software support adopted.

There was only one domain for which the projects did not allocate responsibility for the underlying functionality development to the user interface developers. Such applications, which were non-safety-critical and made use of the greatest range of input devices and output media, also made use of all the available categories of software support. Two conclusions may be drawn from this observation: firstly, the nature of safety-critical applications is such that it is more effective for the same developer to work on the user interface and the underlying functionality; secondly, increased user interface complexity in terms of the input devices and output media used dictates that it is less appropriate to allocate responsibility for development of the entire application to the same person(s). The latter may be due a need for specialist training to process the various media which is reflected in the use of a wide range of software support. This suggests that allocation of responsibility within a project team is reflected in their choice of software support and should therefore be carefully considered when selecting an appropriate tool.

With the exception of applications in one domain, user interface construction is allocated the highest importance rating where the range of software support is used. Respondents who consider HCI to be appropriate after the underlying functionality has been specified develop applications that are both safety-critical and non-safety-critical and either use all four kinds of software support or restrict themselves to just application development environments. Since the credibility of these ratings is questionable, these observations can only be taken at face value and no stronger surmise made in this regard.

Applications (of which many are safety-critical) that separate user interface design from that of the rest of the system make use of software support tools in each of the categories. Despite the absence of graphic design expertise, application development in one domain further separates graphic design from the design of the user interface. Projects in other domains which do have graphic design expertise do not make this separation. Although the experience base within a project team influences the nature of the tools used within each step of the design process, it does not map intuitively onto the breakdown of the design effort itself. This suggests that the way in which the design of the user interface is related to the design of the rest of the system influences the choice of software support, and that the expertise of the team members should be carefully considered to ensure that the best tools are purchased for the people using them.

The implementation of the user interface is rarely separated from that of the rest of the system. Projects that do make this separation are similar both in terms of the types of software support used (that is, use of all types of software support), and (although not identical) in terms of their use of input devices and output media. Development of safety-critical applications makes no such separation. It is unclear whether this observation of software support use is truly a reflection of implementation separation or, given that the extent of inclusion of different input devices and output media within a user interface is directly proportional to the range of software support types used, is instead an indirect result of the nature of the user interfaces themselves. The likely scenario is that there is a complex inter-relationship between all three factors; it is therefore necessary to consider these carefully when choosing an appropriate UIDT.

The same personnel are typically responsible for both the design and implementation of the user interface. Where this is true: (1) applications are generally non-safety-critical; (2) the design is communicated between relevant parties via the use of prototypes; and (3) software support in all four of the identified categories is used. This suggests that design and implementation separation has the potential to influence the choice of software support.

With the exception of real-time systems, projects that dedicate the greatest percentage of time to development of the user interface are mainly those in the safety-critical sectors and have smaller numbers of people in the project teams - despite having the proportionately largest HCI sub-teams. There is no immediately obvious similarity between these projects with respect to their HCI training. Each of these projects uses application development environments and a range of software support from the remaining categories - programming directly in a given programming language is not popular. This suggests that the proportion of development time devoted to the user interface of a single application is potentially a factor of the safety-criticality of the software and is influenced by the proportional size of the HCI project sub-team. These issues are therefore potentially influential in the choice of software support.

Projects that use prototypes extensively make use of support from all four categories. Where evolutionary and revolutionary prototypes are used application development environments tend to be used exclusively. In contrast, projects that make use of horizontal prototypes use all forms of software support. This is the same for vertical and global prototypes and, with the exception of some projects developing safety-critical systems, it is true for local prototypes. This suggests that the nature of the prototypes that are to be used in the development of a user interface will potentially influence the selection of software support - especially with regard to the type of software support.

Although there is no obvious connection between the use of prototypes and the separation of the user interface design from that of the rest of the system, there is a slight correspondence between prototype use and the separation of user interface and underlying functionality implementation – projects that impose the implementation-centric separation make extensive use of prototypes. Furthermore, such projects tend to assign responsibility for user interface design and implementation to the same personnel. The greatest amount of prototyping is done in companies with the proportionally larger HCI sub-teams, comprising the broadest range of HCI training.

The safety-criticality of applications is directly proportional to the use of step-completion mechanisms, versioning mechanisms, and signing-off procedures. Some of the projects that most fully employ the practice of prototyping make least use of these mechanisms. In particular, projects which rely heavily on revolutionary prototypes make the least use of signing-off procedures and are amongst those with the smallest teams but proportionately largest HCI sub-teams. Finally, development of smaller applications (in terms of lines of code) makes less use of these control mechanisms than larger application development. These issues pose an interesting question as to which came first - are the facilities used because they are there (that is, does their provision prompt uptake of these strategies ?), or

are the tools selected because they provide a service which is needed ? (that is, are these strategies already in place within the development process of an organisation and are therefore instrumental in influencing the choice of software support ?). Unfortunately, further study would be required to adequately investigate this issue. However, assuming that (at least in some cases) the strategies are in place before purchasing software support, these observations indicate that application characteristics and developmental strategies are important in terms of the administrative protocol-related functionality that might be demanded of new software support tools.

The above discussions raise one particularly interesting question: *Are people using a **range** of software support because that is the best solution/working process for them, or because they bought the wrong tool in the first place and have had to buy subsequent tools to 'patch' the gaps in functionality ?* The survey results do not provide a definitive answer to this question. However, responses to the pilot use of the questionnaire suggest that, at least in some circumstances, it is the latter. If a range of software support is indeed the best solution, then it is important that a new UIDT evaluation methodology is able to support the effective selection of UIDTs in each of the categories. Alternatively, if it is not the optimal solution and is instead the result of misguided purchases, it is important that a new UIDT evaluation methodology assists developers to identify their functional and contextual requirements and to analyse and compare tools according to these specific criteria such that they are able to identify shortcomings in tools prior to purchase. The proposed UIDT evaluation facility should therefore be designed to deal with both scenarios.

3.6 Conclusions

No investigation had previously been conducted into the *relationships* between user interface development - in terms of development processes and application characteristics - and the use of UIDTs. The qualitative results of this survey illustrate the complexities of these relationships and thereby highlight the important issues that must be considered during the development of a new UIDT evaluation methodology designed to be practicable and relevant to current industrial practice.

It is possible to draw abstract conclusions from the detailed data presented in this chapter. Firstly, a domain-specific development identity is not present in any of the specified domains - that is, there is no strong cohesion between the results for projects in any given domain (there are, in fact, as many if not more similarities between application development across the different domains). It is therefore neither appropriate nor

necessary for a new UIDT evaluation methodology to restrict its applicability, or focus, on the basis of a single application domain.

The majority of the respondents indicated that their typical applications are of medium size and so the conclusions drawn from this research are only truly applicable within this scope of application size. A new UIDT evaluation methodology based on this research cannot therefore make claims to its applicability beyond such dimensions. Although it is likely that the same principles will apply for both larger and smaller applications, this hypothesis would have to be empirically tested. Considered beyond the scope of this thesis, further discussion of this issue is restricted to chapter 8.

Amid the complexities of the relationships outlined in this chapter it is possible to identify three principal areas upon which a new UIDT evaluation methodology need focus. These are: (1) the environment in which the UIDT will be used; (2) the media - both input and output - which comprise the user interface; and (3) the issues regarding the developmental processes that are followed by the user interface development team.

3.6.1 The Environment in Which the UIDT Will Be Used

This refers to the human and administrative factors that contribute to the environment in which the user interface development will take place. The results show application development teams typically comprise five people and that these five people can have different training backgrounds. Furthermore, all project members are not necessarily involved in all aspects of user interface design and development and this is influenced by the training of the members. Since the survey identified the extensive use of prototypes for design and testing of user interfaces, it is likely that project team members' design rôles will often be prototype-centric. Additionally the range of training among the team members was shown to be potentially influential in the choice of software support types.

These observations suggest that a new UIDT evaluation method must consider the rôle of each team member in terms of the overall application development. Specifically, it must allow for consideration, on an individual level, of whether each team member will be involved with the design and/or the development. Furthermore, given the potential diversity in training and therefore experience of the team members, the evaluation methodology must take into account their specific capabilities in terms of software interaction. This may be particularly important for team members allocated to design if the design is to be based on the development of non paper-based prototypes.

The safety-criticality of applications was shown to be an important factor in various issues of interface development. The facility to implement aspects of safety - especially for real-

time applications - normally lies in the power of the underlying programming language. The necessary programming language is therefore likely to be fundamental in that it will effectively short-list candidate UIDTs for selection. Characteristics of the software development companies themselves did not demonstrate influence over the use of software support - other than, perhaps, their ability to meet the cost of deploying the selected software support. Factors such as cost and programming language may prove important influences on UIDT selection and should therefore be catered for in a new UIDT evaluation methodology.

3.6.2 The Media Comprising the User Interface

The results of this survey clearly show the influence of the range of input devices and output media on the selection of UIDTs. When considering UIDTs, these components constitute the low-level functional requirements. In other words, the input and output devices and media that are included in a user interface represent the fundamental criteria on which UIDTs are judged. If, for example, a user interface needs to include speech then this will be an important factor in the decision to select any given tool - if a tool fails to provide the facility to manipulate speech it is unlikely to be rated as highly as another tool that does support speech manipulation. A new UIDT evaluation methodology must therefore cater to the needs of examining user interface media.

However this is essentially common sense - when evaluating UIDTs, a developer will consider the user interface-specific functionality they provide. Of greater importance is the fact that the survey demonstrates the variation in the use of media types across different applications. This therefore suggests that, to be practicable, a new UIDT evaluation methodology must allow the evaluator to tailor this list of functionality such that he need only consider those components of immediate interest within the scope of his current project.

3.6.3 Issues Pertaining to the Developmental Processes

The results of this survey illustrate the software engineering mechanisms employed in industrial user interface development. The survey highlights specific user interface development activities that are typically undertaken, including those relating to testing and evaluation. Furthermore, it highlights the significance of prototypes within the development processes and illustrates the range of prototypes used. It reports on the use of facilities such as version control, step-completion methods, and signing-off procedures, the nature of the underlying data and the implications of its storage, and comments on issues such as user interface response times. Given the extent to which they are each

represented within the responses to the survey and the influence they exert over the choice of UIDT, it is important that a new UIDT evaluation methodology accommodates them. Furthermore, the variation in the use of these facilities, and the individuality of the development processes, dictates that a new UIDT evaluation methodology is sympathetic to the need to tailor the representation of such issues during UIDT comparison.

3.6.4 Summary

This survey illustrates the nature of the components which must be included in a new UIDT evaluation methodology. Furthermore, it highlights the extent of the individuality across software applications in terms of their characteristics, the structure of their development process, and the environment in which they are developed. In so doing, it illustrates that inclusion of the components within a new UIDT evaluation methodology is not sufficient - they must be included in such a way as to allow the realisation of that individuality in terms of selecting a UIDT which best meets the needs of an individual project. The following list summarises the key points to be addressed or observed by a new UIDT evaluation methodology based on the results of this survey:

- *the new methodology cannot sensibly be restricted according to application domain;*
- *the new methodology will only be applicable for middle-sized applications;*
- *the new methodology must individually consider the nature of the involvement of each team member and the interaction capabilities of each - it should accommodate no less than 5 records by default;*
- *the new methodology must consider high-level issues (for example, programming language and cost) that shortlist candidate UIDTs but that are non-deterministic at a detailed level;*
- *the new methodology must consider all user interface media (including interaction objects) and allow their inclusion in any evaluation to be tailored according to the individual requirements of a given user interface;*
- *and the new methodology must consider all aspects of the software engineering process - for example, version control - and allow their inclusion in any evaluation to be tailored according to the individual requirements of a given development project.*

Finally, the need for a new UIDT evaluation methodology is highlighted not only by the fact that less than one third of the respondents were satisfied with the current means by which to evaluate and select a UIDT, but also by the observation that more than half of the respondents wanted access to improved evaluation and selection facilities.

CHAPTER 4 : SUIT - A FRAMEWORK AND METHODOLOGY FOR THE SELECTION OF USER INTERFACE DEVELOPMENT TOOLS

4.1 Introduction

The industrial survey highlighted the complexity of the relationships between practical industrial software development processes and the means by which UIDTs are selected to support these processes. It was concluded that there is a need and demand for an improved UIDT evaluation methodology; the basic approach and key points that need to be addressed by a new methodology were determined. On the basis of these conclusions and existing work in the field of UIDT evaluation, SUIT - a new methodology for the *S*election of *U*ser *I*nterface Development *T*ools - was developed. This chapter describes SUIT and should be read in conjunction with Appendix B which outlines the complete framework and methodology.

4.2 SUIT

SUIT comprises a framework and methodology for the ***s***election of ***u***ser ***i***nterface development ***t***ools. It provides a context-sensitive, extensible, and systematic means by which developers can determine the UIDT that best fits their needs. Motivated by the identified degree of project-specificity in terms of development process and application characteristics - and the influence of these factors on the selection of UIDTs - the principal aim of SUIT is to facilitate the selection of a UIDT relative to the specific needs of an individual software development project. The following sections present the structure of the SUIT framework and the methodology to guide its use.

4.2.1 Developing the SUIT Approach to UIDT Selection: Project-Specific UIDT Selection Related to Software Requirements Analysis and Specification

The SUIT approach applies to UIDT selection a concern for context sensitivity similar to that which has developed requirements engineering over the last twenty years (Holtzblatt and Jones, 1993, Kyng, 1994, Kyng, 1995). As requirements capture has matured, it has become increasingly concerned with the impact of context on software requirements;

developers realised that requirements cannot adequately be identified and accessed without consideration of the context in which their proposed application is to be deployed and that, without such contextualised requirements capture, customers were often dissatisfied with the software they received. Considerable work has therefore been done to contextualise requirements capture - for example, the introduction of stakeholder and viewpoints (Holtzblatt and Jones, 1993, Kyng, 1994, Kyng, 1995, Sommerville, 2000) - and tools have been developed to allow software engineers to describe context and associate it with the functional requirements of the software applications they are designing.

Project-specific UIDT selection is similar to software requirements engineering. Requirements analysis generates a statement of a problem and the requirements that define a successful solution to the problem. A requirements specification identifies the technical requirements of the system including precisely what the system is supposed to do and any operational constraints. Requirements analysis is critical to the development of successful software; if analysts do not correctly understand and communicate the contextualised requirements, then designers cannot be expected to develop the right system (Mynatt, 1990). When selecting a UIDT for a specific project it is equally important to identify contextualised project-specific requirements so that an evaluator can successfully choose a UIDT that meets the identified needs of the project. Thus, where software development *builds* the software defined during requirements analysis, UIDT selection *chooses* a software system on the basis of identified requirements; a requirements specification embodies software development requirements - using SUIT, requirements for UIDT selection are specified in an ideal tool profile (see section 4.2.5.2).

Systems analysts typically ask a series of investigative questions to identify their requirements (Mynatt, 1990). Modified slightly for UIDT selection, they are:

- *Who* is involved in the use of the new UIDT? What rôle is played by each of these people? Who will be using the selected software? Will the users be highly trained individuals?
- *What* functions are to be performed by the selected UIDT?
- *When* or how quickly must the new UIDT system be in place?
- *Where* will the new UIDT fit into the old environment? Where will the personnel fit into the new system?
- *How* will the new UIDT function? Are there constraints on how the new UIDT can operate? Are there constraints, for example, on the hardware, the cost of the UIDT, or the programming language to be used?

Just as an analyst must understand a customers' problem to define an acceptable solution, so too must an evaluator understand the situation or context in which a new UIDT is to be used in order to select the most appropriate solution; to select the UIDT that is a best-fit for

a given project, an evaluator must determine (a) the contextualised requirements that the UIDT must satisfy, and (b) goals and constraints delimiting the selection.

UIDT selection prior to SUIT reflects requirements engineering prior to the introduction of contextualisation; in both domains, procedure concentrated on required functionality - the *what?*. Stakeholder views in UIDT selection and deployment are overlooked given the absence of means to capture and represent them during the selection process (see chapter 2).

On the basis of its identified similarity with the process of requirements capture, UIDT selection needs: (1) a documented methodology; (2) a means to describe context of use; and (3) tools to enable UIDT evaluators to associate the context in which the selected UIDT is to be used with the functional requirements demanded of it, and to analyse candidate UIDTs with respect to this association. Taking as its model the process of requirements capture, the SUIT framework and methodology support this considered context-sensitive approach to UIDT selection. The provision of tool support is discussed in chapter 6.

4.2.2 The SUIT Framework

Based on observation of industrial practice, chapter 3 identified high-level categories of components that should be included within the framework of a new UIDT evaluation methodology. These categories are:

1. the environment in which the UIDT will be used;
2. the media, both input and output, which comprise the user interface;
3. the issues related to the developmental processes followed within the user interface development team.

The SUIT framework assumes, and elaborates, this high-level organisational structure. The framework is an extensible reference model of the functionality and support features that might be found in a typical UIDT. Additionally, it includes the facility to consider and record the environment into which a selected UIDT is to be integrated. The criteria included in each of the high-level sectors are discussed below.

4.2.2.1 Environmental Context of Use

A software development project takes place within the context of an *environment* - that is, a set of conditions determined by the characteristics and experience of the organisation and/or project, the personnel, and management issues. In its conclusions, chapter 3

demonstrated that not only is the environment potentially unique for every project, but it influences UIDT adoption. SUIT models this environment in terms of:

1. human resources;
2. institutional/project goals and/or constraints.

Project characteristics, characteristics of project personnel, and management issues are often quoted as factors that contribute to the difficulty of developing software (Mynatt, 1990, Sommerville, 2000). In general, the first and third of these amount to institutional and/or project-specific goals and constraints that delimit software development, and consequently the selection of a UIDT to fit in with that development. Characteristics of project personnel determine available human resources, and therefore identify the capabilities of the anticipated UIDT users.

Analogous with requirements analysis, these factors resemble issues typically included in the specification of non-functional requirements for a software development project (Mynatt, 1990, Sommerville, 2000). Failure to use these aspects to inform and guide UIDT selection means that decisions regarding the suitability of tools are made without reference to the context in which the tools are to be used. This can potentially result in UIDT choices that may be appropriate in terms of their functional provision, but are inappropriate in terms of the environmental resources and constraints that affect the success of tool use/adoption.

Human Resources

> *"Computer systems are used by people. If the limitations and abilities of these people are not taken into account when designing the system, they will not use it in the best possible way"* (Sommerville, 2000)

This comment highlights the importance of considering the capabilities of software users not only when *designing* software, but also when *selecting* a software tool. The industrial survey illustrated that software is typically developed by teams of people who as individuals, have varied and often specific skills, different UIDT experience, and work on varying amounts of the project. As a group they provide a complex set of related resources and constraints that need to be considered carefully when selecting the most suitable UIDT for a project.

SUIT provides the means to explicitly include consideration of the interactive capabilities of project team members in the UIDT selection process. To accommodate the potentially different rôles of individuals in the team (see chapter 3), their experience of rapid prototyping and end-product implementation is recorded separately. The survey suggested that teams typically comprise five people, and so by default the framework includes the

facility to record data for that number. Given its extensibility, however, it is possible to increase or decrease this number as needed for a specific project. For each, the following information is recorded:

- the interaction mechanism(s) with which the team member is familiar (including specific programming languages where appropriate);
- the level of semantic knowledge the team member possesses regarding the domain for which the application is being developed;
- the level of stress under which the team member is placed whilst working on the project;
- the degree to which the team member's rôle in the project is routine;
- and the workload of the team member.

Sommerville claims that it is easy for programmers who know one programming language to learn a new one of the same type - for example, programming languages belonging to the Von Neumann model (Sommerville, 2000). Loosely applying this principle to developers learning to use a new UIDT suggests that, to ease the learning process, it is important to consider the interaction mechanisms or styles that are familiar to the developers. Mynatt suggests that the prior experience or training of software developers is one of the primary characteristics of personnel that contributes to successful software development (Mynatt, 1990). Supported by the findings of the industrial survey, SUIT records the interaction mechanism(s) with which team members are familiar.

In terms of the efficiency and effectiveness of development, it is an advantage if developers have semantic knowledge of the application area for which the software is being developed (Mynatt, 1990). This can therefore be noted for each team member.

Developers' ability to withstand stress is an important personality trait that contributes to the overall success of a software development project - as stress builds up, performance suffers, which in turn delays product development (Sommerville, 2000). SUIT allows the level of stress under which team members are working to be acknowledged so that, if necessary and where possible, a UIDT can be selected to reduce (or at least not add to) stress levels by matching a new UIDT with the users' known environments.

To build software, developers need to understand the problem, work out a solution strategy, and then translate it into a paradigm. Understanding the problem involves it entering working memory from short-term memory, at which point it is integrated with existing data in working memory and analysed to work out a solution (Sommerville, 2000). By enabling evaluators to record users' workload and the degree to which each user's rôle in the project is routine, SUIT makes it possible to acknowledge the existing demand on the working memory of individual team members, and thereby their potential capacity to

effectively learn a new UIDT. Where considerable demands on team members' working memory already exist, it is important, if possible, to avoid introducing a new UIDT that further increases it.

Interaction mechanism information is used directly to generate project-specific ideal tool profiles (see section 4.2.5.2). The remaining ancillary information can be used to assist in the comparison process if, after examining the UIDT data, it is not immediately obvious which tool is most suited to the project.

Institutional/Project Goals or Constraints

A software development company may have specific long-term goals that factor in the acquisition or use of a UIDT. Additionally, individual projects may have to work within certain operational or implementational constraints - for example, programming language restrictions (identified in chapter 3). These delimiting issues must therefore be considered when selecting a UIDT. SUIT captures the following constraining information:

- the available funding for purchase and installation of the UIDT;
- the speed with which the development must take place;
- the staff training which is possible upon purchase of the UIDT;
- the universe of UIDTs from which the selection can take place;
- the programming language which needs to be supported by the UIDT;
- and the development platform(s) on which the UIDT must operate.

Management issues such as project goals, funding and scheduling are widely acknowledged bounding constraints or determinants in software development projects (Mynatt, 1990, Sommerville, 2000, Brown et al., 1992) - hence their representation in SUIT. A team member is an asset to a project/organisation if trained in the use of a tool; without formal training a team member may take a certain amount of time before becoming an asset with respect to a tool. Availability of staff training may therefore either constrain or expand UIDT selection choices. Similar project goals or constraints include the platform on which the selected UIDT is to operate and the set of UIDTs from which the selection can take place. SUIT therefore allows each to be recorded as part of the selection process.

The principal rôle of these factors is to constrain the set of potential tools from which the evaluator can choose. If a tool cannot be found that conforms to these constraints, they may have to be negotiated to make a selection possible. Initially though, they set the high-level boundaries within which the evaluator must operate - although not directly involved with the actual evaluations, these environmental aspects influence the process at the highest level (see chapter 1).

4.2.2.2 User Interface-Specific Requirements

Like requirements analysis, the functional requirements of a UIDT must be specified so that prospective tools can be compared against the facilities they are required to support. SUIT partitions functional requirements into: (1) user interface-specific requirements; and (2) developmental context of use. This section discusses the former and the latter is considered in section 4.2.2.3.

The survey illustrated the varied use of input devices and output media within different user interfaces. The purpose for which an application is being developed and the design of its user interface generates a set of *project-specific* user interface-related functional requirements that focus UIDT selection. If a UIDT is to be complete in its support of the features that are to be included in a specific user interface, each of these requirements must be catalogued and checked against the functionality provided by each potential UIDT. It is unlikely that user interface-specific requirements will always be clearly defined from the point at which UIDTs are considered. It is therefore important that the framework and methodology accommodate iterative refinement of the selection process - the ability to estimate requirements where they are ill-defined. The process of requirements estimation is similar to that of software development cost estimation - for example, using expert judgement or estimation by analogy (Sommerville, 2000).

User interface-specific features incorporate those components that could appear in a user interface (including its associated peripherals). In chapter 2 it was suggested that the components itemised within the Hix *et al* checklists constitute the most comprehensive categorisation of user interface features currently available. Due to lack of extensibility, however, over a decade after their inception the checklists have become incomplete and will continue to do so. Exploiting the basic user interface-specific component listings in the Hix *et al* checklists, the SUIT framework extends and re-organises them (Hix and Ryan, 1992, Hix and Schulman, 1991, Hix et al., 1989). Whilst updating the lists, the SUIT framework is structured so that new technological advances can be included to handle future developments; observing the data record structure, new records can simply be added to the various categories to extend them.

The SUIT framework adopts a hierarchical organisation for the user interface-specific features. The list below describes the two highest levels of classification. Reference should be made to the actual framework (see Appendix B) for details of the lower levels of classification (that is, the actual concrete components or features).

- *General Properties of the User Interface*
  - Screen Navigation
  - Dialogue
  - Active Devices
  - Miscellaneous
- *Input/Output Devices*
  - Input Devices
  - Output Devices
  - Target Monitor
  - Other Hardware Devices
- *Output Presentation Types*
  - Text
  - Graphics
  - Animation
  - Audio
  - Video
  - Data Driven/Dynamic Objects
- *Combinations of Output Presentation Types*
  - One Screen/Different Windows
  - One Screen/Same Window
- *User Interface Features*
  - Boxes
  - Menus
  - Forms
  - Windows
  - Text Area

The hierarchy of these lists: (1) makes navigation around the lists of features potentially easier; and (2) enables an evaluator to select and operate on an entire category rather than having to consider each individual component - thereby facilitating requirements estimation where requirements are ill-defined.

The SUIT framework extends the manner in which each individual component is considered. Where applicable, individual components are sub-divided to allow independent examination of the manner in which they are instantiated, configured, and laid out within a user interface. This component sub-division has been implemented on the basis of observed variation between the techniques used to achieve these three fundamental aspects of feature inclusion. For example, using tools like Visual Café™ or Delphi™: developers instantiate a component by dragging-and-dropping it from a component palette onto their workspace; they configure it by filling-in fields in the component's property sheet; and locate it either by further drag-and-drop operations or by entering co-ordinates

in the property sheet[52]. To enable accurate illustrations of individual tools to be recorded, this sub-division is therefore provided, as appropriate, for each user interface-specific feature.

Chapter 3 concluded that simply including the appropriate features in a UIDT evaluation framework was not sufficient - the list of features must be tailorable to project-specific needs. Using the SUIT framework, listed components can be included or excluded from consideration according to project-specific requirements during UIDT selection. Conversely, if using SUIT to generically compare UIDTs, all components are used. The methodological processes of which these activities are part are discussed in sections 4.2.5.2 and 4.2.5.1 respectively, and the mechanism is explained in detail in Appendix B.

4.2.2.3 Developmental Context of Use

Chapter 3 illustrated the variation in individual software development processes, the nature of the prototypes used, and the use of administrative procedures such as version control. Given the influence such issues were shown to have upon the use of UIDTs, it was essential to include them in SUIT. As with the user interface-specific features, these components are tailorable to accommodate project-specific requirements and change.

The developmental context of use is primarily concerned with the integration of the UIDT into existing working practice and the ways in which the UIDT can enhance or support these practices. Features include the design and development methodology adopted by the organisation for the completion of the project, and associated technological support that must be provided by the UIDT. It is important for a requirements definition to establish and document a systems context - that is, the relationships between the systems being specified and other human and computer systems. More leverage is obtained from a tool if it appears (from the users' perspective) to be integrated with the context in which it is to be used (Sommerville, 2000). Assuming that a tool will be most easily adopted if it is compatible with, or enhances, the developmental needs and working practice of a project, SUIT allows the evaluator to assess the degree to which tools provide:

- user interface design and development steps;
- evaluation assistance;
- development aids;
- project management facilities;
- tool characteristics;
- and quality attributes of the target system.

---

[52] These tools also enable a developer to program the instantiation, configuration, and layout of a component directly in their supported programming language.

The Hix *et al* listings were again exploited as the basis for these categories, and were expanded and radically re-organised (Hix and Ryan, 1992, Hix and Schulman, 1991, Hix et al., 1989). A category for 'user interface design and development steps' was included. It lists all the identified user interface development activities (see chapter 3) and itemises types of prototype to let evaluators examine the extent to which a given UIDT will support project-specific development strategies. The category of 'evaluation assistance' was included given the extent to which evaluation activities were identified by respondents of the survey. It lists automated evaluation assistance that might be provided by a UIDT. Having observed that UIDTs can be used to generate up to 90% of user interface code (see chapter 3), the category 'development aids' considers practical coding aspects of user interface development and lists the facilities that might be provided by a UIDT to support this level of user interface development - for example, file comparison facilities and hierarchy editors.

The 'project management' category was included to reflect the identified use of project administration aids such as version control in UIDT selection. To account for the more software engineering-centric activities associated with user interface development, the 'tool characteristics' category was introduced. Listed components deal with some of the issues identified by both Bass *et al* and Valaer and Babb - for example, the compatibility of the UIDT with other software (in particular, its ability to integrate with storage mechanisms for application data types), the modifiability of UIDT features, and the reusability and portability of the code produced by the UIDT (Bass et al., 1994, Valaer and Babb, 1997). Finally, the 'quality attributes of the target system' category lists intended attributes of quality of the user interface produced using a UIDT. Although based on the issues of quality itemised by Bass *et al* (Bass et al., 1994), they are not considered in the same way.

It did not seem appropriate to apply the instantiation, configuration, and layout component sub-division to all issues of developmental context of use. With the exception of components in the 'design and development steps' and 'quality attributes of the target system' categories, developmental context of use components are therefore assessed according the mechanisms by which they are launched (instantiated) and configured. For example, version control can be considered according to how it is activated and configured to the needs of the user.

No sub-division has been applied to 'design and development steps' components given no obvious need, or means, for such. 'Quality attributes of the target system' components have been treated differently. Although based on the quality attributes identified by Bass *et al*, they are quantified from a different perspective to remove evaluator bias and increase the practicability of UIDT evaluation (see chapter 2). Rather than consider these attributes

in terms of the target system, SUIT examines UIDT support for monitoring these attributes. For each tool SUIT records whether there is support to observe established standards/guidelines that lead to measurements of quality and whether the tool supports testing of quality attributes in the target code. Examples of such facilities include the User Interface Design Assistant (UIDA) which critiques window layouts for compliance with Motif style guidelines (Boker, 1995) and Wave (Kasday, 1999) which checks web page accessibility against the W3C Web Accessibility Guidelines (Chrisholm et al., 1999). By advocating that UIDTs are evaluated in terms of their facility to assist the achievement, and to test the presence of, quality in developed code - as opposed to evaluating the target code itself - evaluator bias should be greatly reduced, if not eliminated, from this aspect of UIDT selection.

Developmental context of use components are tailored according to project-specific needs when performing a project-specific UIDT selection, and are used *en masse* when generically comparing UIDTs.

4.2.2.4 Dimensions of Measurement

Chapter 2 outlined the dimensions against which UIDTs are measured according to the different evaluation methods. These included: the functionality and usability of the tools (Hix and Ryan, 1992, Hix and Schulman, 1991, Hix et al., 1989, Sundaram and Ramamurthy, 1996, Valaer and Babb, 1997); the learning curve associated with the tool; and the run-time efficiency of the software produced by the tool (Sundaram and Ramamurthy, 1996). Although a number of the issues raised by Bass *et al* and Valaer and Babb have been included as constituent components in the SUIT framework, certain of their criteria amount to dimensions against which tools can be measured (Bass et al., 1994, Valaer and Babb, 1997). The dimensions for consideration are therefore:

- functionality;
- usability;
- learnability;
- run-time efficiency of the software produced by the tool;
- and user support provided within the tool.

Only one of the five dimensions was completely rejected for use in the SUIT framework - that of the run-time efficiency of software produced using a UIDT. It was abandoned on the grounds that: (a) it prevents an evaluation focussing exclusively on the UIDT and essentially measures a developer's ability to design efficient software; and (b) it introduces considerable potential evaluator bias to UIDT evaluation. Albeit modified, the remaining four dimensions were accepted. Each is discussed in turn below.

Functionality

The (potentially tailored) list of UIDT components asserts the functional criteria that need to be examined. SUIT, like the Hix *et al* checklists, measures the provision of functionality in terms of the interaction mechanism(s) used to achieve the functionality. For example, if a List Box is listed as a functional requirement an evaluator would determine the interaction mechanism(s) used by each UIDT to instantiate, configure, and layout a List Box. The SUIT framework itemises the five most common interaction mechanisms - graphical manipulation, graphical programming language, programming language, scripting language, and form-filling. These mechanisms were chosen on account of their wide acceptance both in literature (Shneiderman, 1998, Hix and Ryan, 1992, Hix and Schulman, 1991, Hix et al., 1989, Sundaram and Ramamurthy, 1996, Hix and Hartson, 1993, Newman and Lamming, 1995, Preece et al., 1994) and in existing UIDTs (Visual Café™, Delphi™, Macromedia Director™, Microsoft® Access97). Perhaps noticeable by its absence from this list is command line interaction - it was omitted on the grounds that contemporary UIDTs are typically graphical in style and so do not adopt a command line paradigm. By virtue of its extensibility, the framework does, however, allow any other interaction mechanism(s) to be recorded if necessary.

Usability/Learnability

> *'The situated nature of usability means that a system may be usable by a particular user for a particular task in a particular environment but unusable or poorly usable with another combination of user, task, and environment.'* (O'Neill, 1998).

Chapter 2 discussed the inappropriateness of measuring usability in terms of the ease with which an evaluator is able to interact with the UIDT to achieve a specified goal, and questioned the suggested measurement of learnability.

SUIT recognises the importance of assessing both usability and learnability when selecting a UIDT. However, to increase project-orientation in the measurement of these factors, avoid evaluator bias, and increase the practicability of UIDT selection, SUIT adopts a different stance towards their measurement.

Hix and Hartson define usability to be a combination of: ease of learning; high speed of user task performance; low user error rate; subjective user satisfaction; and user retention over time (Hix and Hartson, 1993). It would not be feasible to perform empirical studies of the usability of a tool for each particular context of use - that is, empirical studies of the usability for a given set of project-specific functionality and user group; an alternative analytic method is therefore required. Hix *et als'* method - that is, asking evaluators to

judge usability - is unacceptable because it is has been shown to return low inter-evaluator reliability (Hix, 1986, Hix et al., 1989, Hix and Schulman, 1991, Hix, 1991, Hix and Ryan, 1992). In the case of SUIT, evaluators have access to information about the interaction capabilities and experience of the project team members. SUIT therefore focuses on a *prima facie* measure of learnability as its measure of usability (see below).

Dix *et al* define learnability to be:

> *'the ease with which new users can begin effective interaction and achieve maximal performance'* (Dix et al., 1993).

and suggest that it is a measure of the system's:

- *predictability* - the deterministic behaviour of the system from the user's perspective;
- *synthesisability* - user's ability to determine the effect of future interactions depends on the user having a mental model of how the system behaves;
- *familiarity/guessibility* - the extent to which a user's knowledge and experience in other real world or computer-based domains can be applied when interacting with a new system. For a new user the familiarity of an interactive system measures the correlation between the user's existing knowledge and the knowledge required for effective interaction;
- *generalisability* - users try to extend their knowledge of specific interaction behaviour to situations that are similar but previously un-encountered. Generalisability can apply across a single application or across a variety of applications and leads to a more complete predictive model of the system for the user;
- and *consistency* - likeness in behaviour arising from similar situations or task objectives (Dix et al., 1993).

Of the above, predictability and consistency are the least attributable across applications. The remaining factors reflect users' knowledge of other systems and the manner in which that knowledge can assist/deter their ability to effectively use a new UIDT. Since users are considered an asset - with respect to a UIDT - when they can effectively and efficiently use the tool (see section 4.2.2.1) it is assumed that reducing the time taken for users to achieve this status is desirable. *Prima facie* learnability must therefore be assessed from the perspective of the anticipated end-users. To do this the capabilities of the people who will be using the tool - the human resources (see section 4.2.2.1) - must be considered.

Although the usability of each potential UIDT could be formally assessed with each anticipated user[53], this would render UIDT selection impracticable and not cost-effective.

[53] Using accepted usability evaluation methods such as protocol analysis, heuristic evaluation, cognitive walkthroughs etc. (Lindgaard, 1994).

With practicability in mind, SUIT assesses the *prima facie* learnability of a new UIDT from the perspective of the anticipated end-users according to the lowest common denominator across users and applications alike - that is, the interaction mechanisms with which the users are familiar, or prefer, on the basis of previous experience with other applications. It is hypothesised that selecting a UIDT that lessens disparities between the interaction mechanisms used by the tools and those which are appropriate for the project-specific team members will increase the familiarity/guessibility and generalisability of the selected UIDT and therefore increase the degree to which the UIDT is accepted (see discussion of tool acceptance in section 2.4.2). SUIT therefore considers *prima facie* learnability to be a correlation between: (1) a combination of the interaction mechanisms familiar to, or preferred by, team members and the mechanisms most suited to the software being designed; and (2) the concrete interaction mechanisms employed by a UIDT. Besides considering the preferred or familiar interaction mechanisms of the users, it is necessary to consider the interaction mechanisms that are most appropriate for a given task. Although these may coincide, there may be cases where specific interaction styles are more suited to (or indeed better support) a particular development task, and so SUIT allows them to be included in the correlation.

Using SUIT, an evaluator does not explicitly measure usability/learnability when collecting UIDT data. Instead, the predictive assessment of usability/learnability is based on the measurement of functionality - the record of interaction mechanisms used - and is deferred until the interaction mechanism data recorded for each tool is compared to the human resource information in the project-specific ideal tool profile, and matches/mis-matches identified. The remaining dimensions in SUIT can be called upon to determine the quality of match across tools where necessary.

Interaction Assistance (Provided by UIDT)

Interaction assistance (or user support) provided by a UIDT is mentioned in both the Hix *et al* checklists and the Valaer and Babb evaluation methodology (Hix and Ryan, 1992, Hix and Schulman, 1991, Hix et al., 1989, Valaer and Babb, 1997). It is hypothesised that the presence or otherwise of interaction assistance can potentially influence not only the learnability of a UIDT, but also the extent to which the functional mass of the UIDT is used (Dix et al., 1993, Shneiderman, 1998). Rather than simply record the provision of interaction assistance relative to the UIDT as a whole (see Hix and Ryan, 1992, Hix and Schulman, 1991, Hix et al., 1989), SUIT adopts a more pervasive approach to measuring interaction assistance; it records the interaction assistance types provided to assist the use of each UIDT component. SUIT identifies four of the more common types of interaction assistance - adoption of default values, use of wizards, availability of context-sensitive

help, and provision of online tutorials - and, again by virtue of its extensibility, provides the facility to record any other forms of interaction assistance if necessary. For each active component in the SUIT framework, an evaluator measures user support in terms of which of the listed interaction assistance types are provided.

Additional Dimensions

The SUIT framework measures UIDTs according to three additional dimensions:

- cognitive demands placed the on the user;
- quality of feedback returned to the user;
- and any miscellaneous issues not covered elsewhere.

Allocated secondary status on the grounds that they can introduce evaluator bias, each is measured via the use of a modifier - '+', '0', or '-' - and an accompanying explanatory comment. These dimensions let an evaluator record ancillary information about individual components in the SUIT framework such that, if a UIDT cannot be selected on the basis of the primary dimensions (for example, if evaluated tools are so similar that it is not possible to select one on the basis of functionality, *prima facie* usability/learnability, and user support) this additional information can be used to assist the final selection. The alternative means by which these dimensions are measured highlights that they are essentially a rating of opinion and therefore potentially more subject to bias.

4.2.3 General Layout of the SUIT Framework

The SUIT framework is organised around tables - an example of which is shown in Figure 4.1. There is a separate table for each of the categories discussed in sections 4.2.2.1 to 4.2.2.3. The rows in the tables correspond to individual components (and their sub-division), and the columns to the measurement dimensions discussed in section 4.2.2.4.

As illustrated in Figure 4.1, the label for each component is preceded by a small checkbox. Selection of a checkbox includes the associated component within an evaluation process - that is, the component is considered active. When choosing a tool for a specific project, only those components that are relevant to the project are activated (see sections 4.2.2.2 and 4.2.2.3). In contrast, when generically comparing tools, all checkboxes are selected.

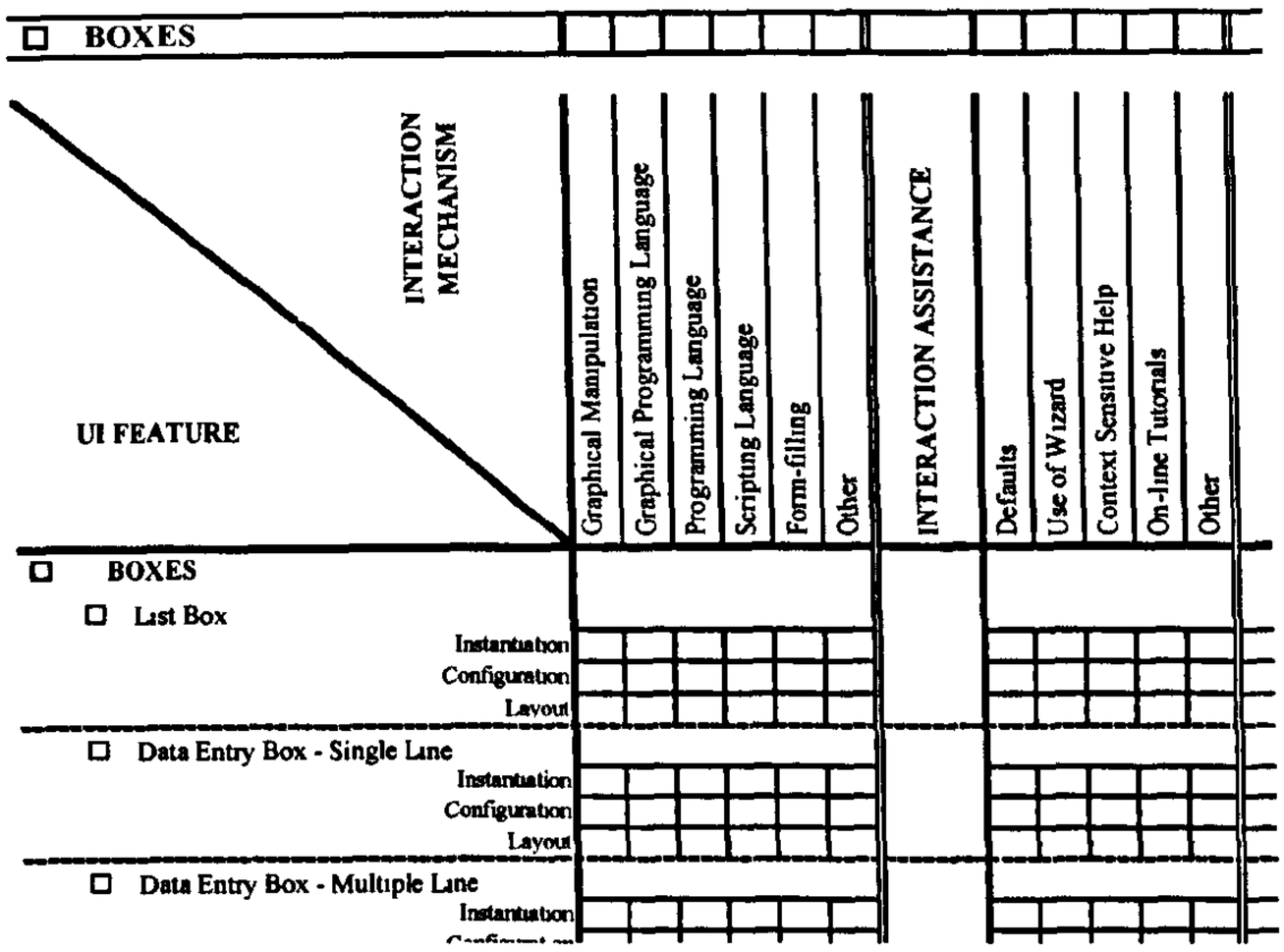


Figure 4.1 - *A section of a typical SUIT framework table*

Project-specific functional requirements usually vary in significance. Some UIDT evaluation methods allow components to be ranked or weighted according to their importance (Hix and Ryan, 1992, Hix and Schulman, 1991, Hix et al., 1989, Mosley, 1992, Mosley, 1995). However, such mechanisms are time consuming and complex, impeding *practical* UIDT evaluation (Mosley, 1992, Mosley, 1995). The SUIT framework replaces weighting factors with tailorability - in essence, project-specific framework tailoring is the same as weighting factors of relevance to the project more heavily than factors that are of little interest (a two valued ranking system)[54].

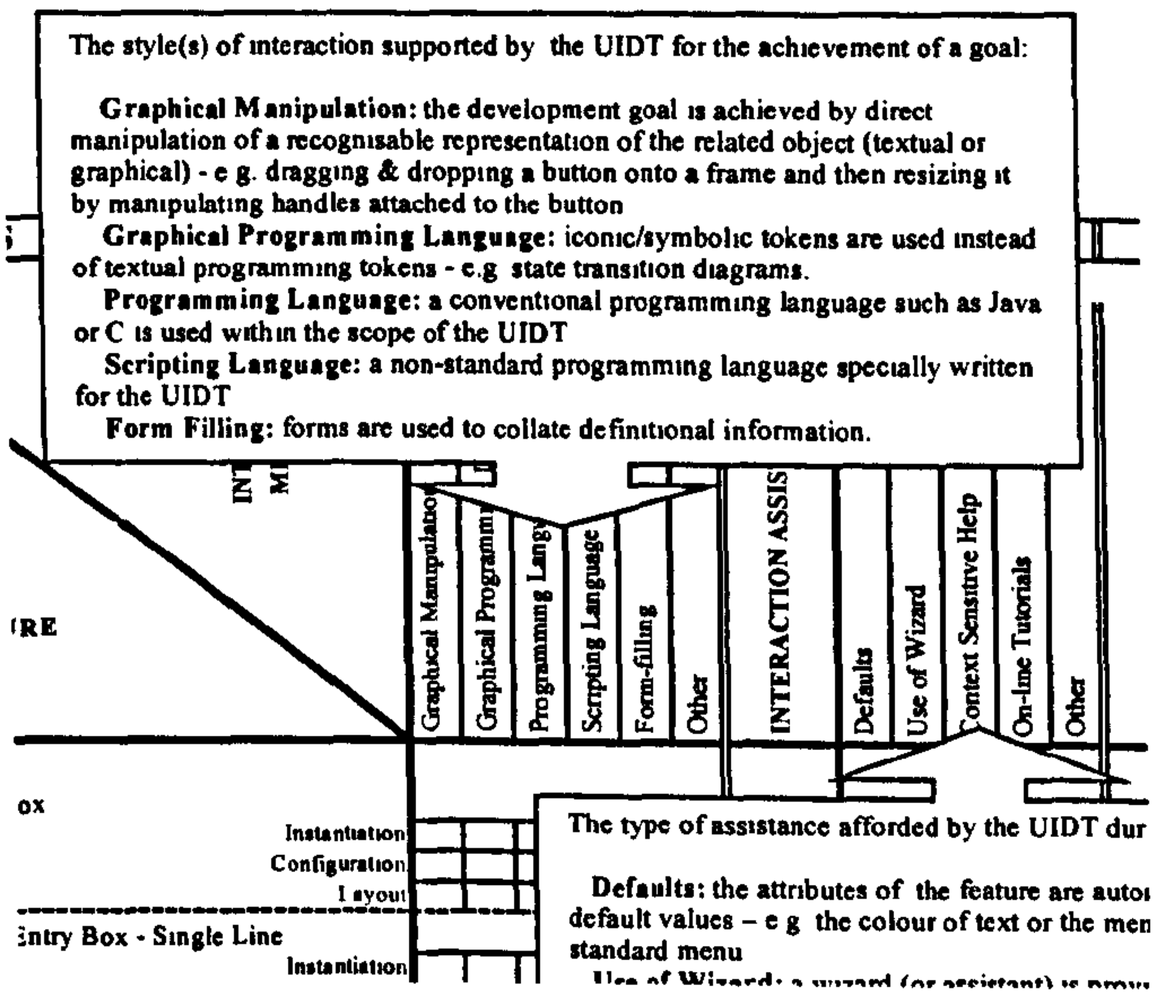


Figure 4.2 - *A section of a typical SUIT framework glossary page*

[54] Additional means of achieving basic component prioritisation are available in the SUIT visualisation environment, and are discussed in chapters 6 and 7.

4.2.4 Integrated Help in the SUIT Framework

Each page in the SUIT framework has a corresponding glossary page. These are annotated versions of the framework tables that define every term in each table. The glossary pages are located opposite their corresponding framework page for ease of reference when using the framework. An example of part of a glossary page is shown in Figure 4.2.

4.2.5 The SUIT Methodology

The illustrated diversity among software development projects and its identified influence over UIDT selection and use (see chapter 3) suggests that, for project-specific UIDT selection, it is not a UIDT's gross functionality that is important but rather it is the match of tool functionality against project-specific functional requirements that determines UIDT suitability.

Preceding work in the field of UIDT evaluation concentrated on generic UIDT comparisons (see chapter 2). Given that their strength generally lies in the 'more-is-better' approach to UIDT evaluation, they afford little support for capturing the project-specificity of practical software application development. Although SUIT can be used to generically compare UIDTs, it is also designed to *facilitate the context-sensitive selection of a UIDT which best meets the needs of a given project* - not just in terms of the user interface-specific functional requirements, but also taking into account the environmental and developmental contexts of UIDT use. Hence, SUIT can be used in three ways:

1. to select a UIDT based on a generic comparison of tools;
2. to select the 'best-fit' UIDT for a project based on the specific context and requirements of that project where the project has no precedent within the organisation and hence there is no access to existing comparative data (see below);
3. and to identify an appropriate UIDT for a specific project based on comparisons with previous projects.

The applicability of each approach is based on: (1) the stage of design and/or development of the project; (2) the novelty of the project within the context of the organisation; and (3) the intended specificity of the outcome of the use of SUIT. Each approach dictates the appropriate route through the SUIT methodology (see Figure 4.3) and the manner in which the framework is manipulated - the tailorability alluded to in sections 4.2.2.2 and 4.2.2.3 - and used.

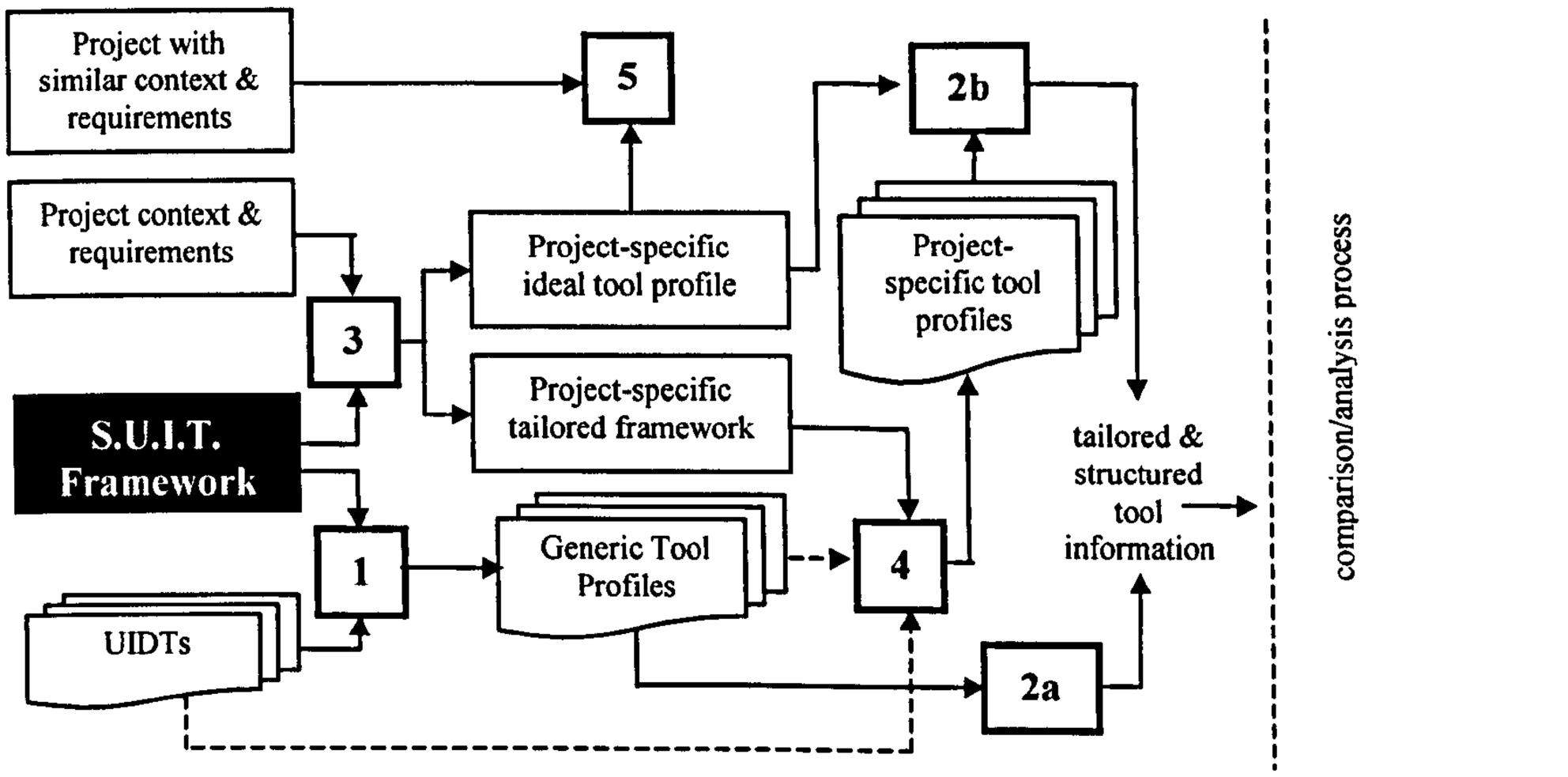


Figure 4.3 - *The route map of the SUIT methodology showing all paths*

The SUIT framework is essentially an extensible reference model of all functionality and support features that might be found in a UIDT (as described in section 4.2.2). Based on the appropriate form of use, the SUIT framework is tailored – that is, framework components are included or excluded from consideration – to provide a structure or pilot for data-collection and thereafter a context for the interpretation of that data. The degree to which the framework *can* be tailored is determined by the amount of information available to the evaluator at the time of using SUIT. The degree to which the framework *is* tailored also depends on the intended use - for example, generic comparison may require no tailoring.

In terms of what is possible and appropriate, the three different routes through the SUIT methodology reflect the natural progression of UIDT selection as an organisation matures. Initially an organisation may not have any well-defined projects upon which to base a selection and so might generically compare UIDTs. As it matures, the organisation may need to select UIDTs for specific projects - either from an existing library of UIDTs or for direct purchase. Once well established, and having used SUIT for several project-specific UIDT selections, the organisation will have a body of SUIT-specific selection data upon which they can base new selections.

This pattern or process of maturation of use of the SUIT framework and methodology is analogous with the Capability Maturity Model (CMM) of software development (Humphrey, 1989, Paulk et al., 1993). Just as the CMM was designed to guide software organisations when improving their software development strategies by identifying their current level of process maturity and those issues most critical to software quality and process improvement (Paulk et al., 1993), SUIT is designed to guide evaluators when

trying to improve their process of UIDT selection by identifying their evaluation capabilities and critical selection requirements and enabling them to perform increasingly sophisticated and/or efficient UIDT selection. UIDT evaluation and selection prior to SUIT can be likened to the activities of an 'immature software organisation' (Humphrey, 1989, Paulk et al., 1993) – without SUIT, UIDTs are selected in an unstructured or *ad hoc* fashion with little or no consideration of the context in which they are to be used (see chapters 2 and 3). Like the intermediate levels of process maturity identified in the CMM (Humphrey, 1989, Paulk et al., 1993), the use of SUIT - first for generic UIDT comparisons and then project-specific UIDT selections - demonstrates increasingly structured, systematised, specialised, and rationalised UIDT evaluation and selection. Once an organisation has consistently and effectively used SUIT to perform well documented project-specific UIDT selection, it will reach the level of maturity whereby it is able to reuse the results of UIDT evaluation and selection with confidence and accuracy. This is analogous with the activities of a 'mature software organisation' (Humphrey, 1989, Paulk et al., 1993). With time and extended use of SUIT, organisations may achieve an equivalent of the 'Optimising Level' of the CMM (Humphrey, 1989, Paulk et al., 1993) whereby they monitor their SUIT-based UIDT evaluation and selection procedure and the impact of their selection results in order to optimise their activities. This is, however, beyond the immediate scope of this research. The following sections illustrate the various uses to which SUIT can be put.

4.2.5.1 UIDT Selection Based on a Generic Comparison of Tools

SUIT can be used to perform generic - that is, non project-specific - comparisons of tools. Independent of a project context, a generic comparison requires no knowledge of the project or the context in which the project is to take place.

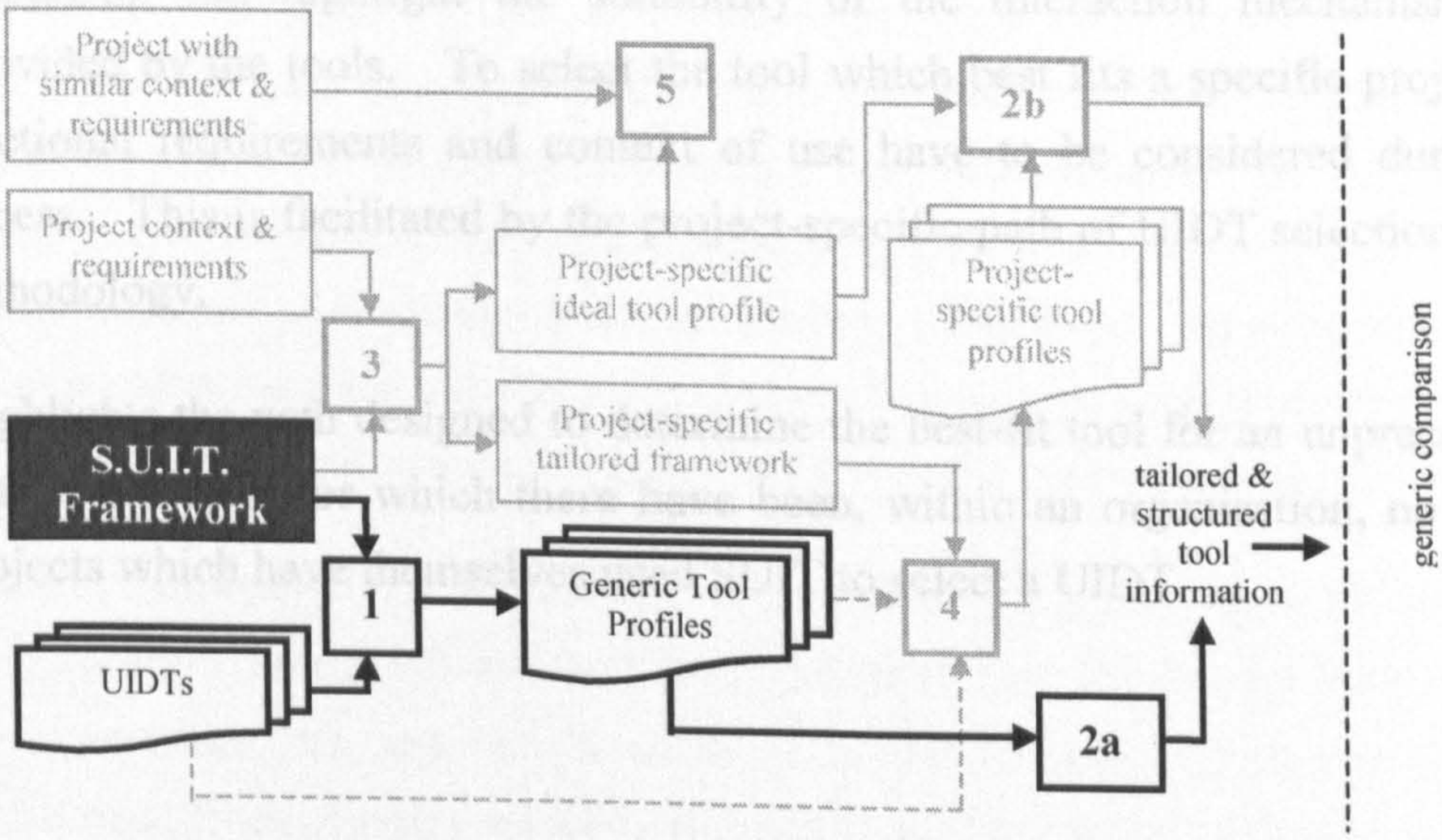


Figure 4.4 - *Performing a generic comparison of UIDTs using SUIT*

Figure 4.4 highlights the path that would be taken through the SUIT methodology to perform a generic comparison of UIDTs. The SUIT framework is used as a complete reference model - every component is an active criterion for the comparison of the tools.

At step 1 in Figure 4.4, each tool is examined in turn to determine which of the components listed in the SUIT framework are/are not present - those present are measured according to the dimensions discussed in section 4.2.2.4. For each tool, the information recorded during this process forms a generic profile of the individual tool – that is, a profile of the tool in relation to the entire SUIT framework (or reference model) uninfluenced by the specifics of a project. Generic profiles can be collated (step 2a) for use in the final stages of generic tool comparison.

Given the complexity of the decision space in which UIDT selections take place, together with the lack of knowledge about the manner in which UIDT-specific trade-offs and goal/constraint negotiation occurs (see chapter 1), SUIT provides the information, structure, and a means of data visualisation (see chapter 6) for data comparison but does not stipulate the process by which the final comparison should be performed. The detailed instructions for the use of SUIT (see Appendix B) do, however, indicate what data comparisons are valid. This is the case for all uses of SUIT.

4.2.5.2 Project-Specific UIDT Comparison

Generic comparisons highlight differences between tools, but it is in the context of a specific project that the significance of the differences becomes apparent. For example, if a project requires a specific subset of UIDT functionality, a generic comparison does not *easily* highlight which of the tools examined best meets these requirements – a project-specific comparison does. Furthermore, with respect to project team members, a project-specific comparison can highlight the suitability of the interaction mechanisms and assistance provided by the tools. To select the tool which best fits a specific project, the project's functional requirements and context of use have to be considered during the selection process. This is facilitated by the project-specific path of UIDT selection within the SUIT methodology.

Figure 4.5 highlights the path designed to determine the best-fit tool for an unprecedented project – that is, a project for which there have been, within an organisation, no similar preceding projects which have themselves used SUIT to select a UIDT.

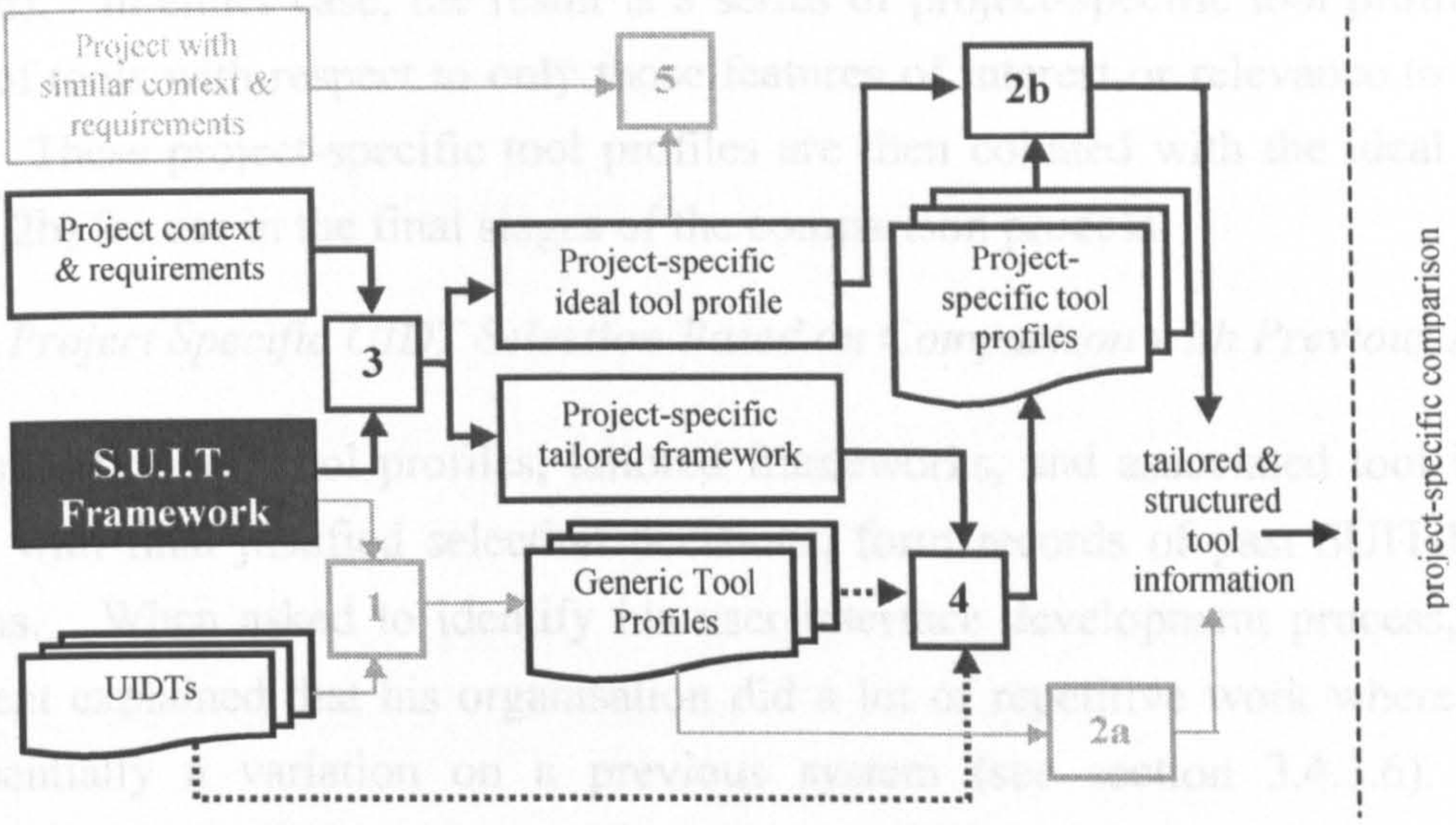


Figure 4.5 - *Performing a project-specific comparison of UIDTs using SUIT*

The SUIT framework is tailored according to project requirements - that is, components are added to and/or excluded from the framework - to produce a project-specific tailored framework that only considers data relevant to that project. Information about the environmental context of use for the project is recorded in one copy of the tailored framework, which is then further augmented by correlating the most appropriate interaction mechanisms (see section 4.2.2.4) with the active components, creating a profile of the *ideal tool* for that project - step 3 in Figure 4.5. It is against this profile (similar to a requirements specification - see section 4.2.2.4) that the data about *real* tools is compared to determine which tool best matches the ideal.

The stage of project development at the time of using SUIT determines the level of detail at which an evaluator can tailor the framework and subsequently generate the ideal tool profile. To enable framework tailoring to be specific and decisive, it is recommended that SUIT be used for project-specific UIDT selection at the point when the project team has been finalised and when user interface requirements are relatively well defined. SUIT can, however, also be used iteratively as information becomes available. If, for example, requirements are not well defined when first considering UIDT selection, an evaluator can choose to include all/most components to err on the side of caution. As requirements are defined the active component list can be tailored to exclude components that are definitely not required, and the UIDT data compared again in light of the increased level of detail.

A project-specific tailored SUIT framework is essentially a project-specific reference model which is used to pilot the collection of (only relevant) data for the real tools. Step 4 in Figure 4.5 shows that this data can be sourced in one of two ways: it can either be extracted (or filtered) from existing generic tool profiles or it can be collected via direct examination and measurement of the actual tools (as with generic comparison data-

collection). In either case, the result is a series of project-specific tool profiles - that is, profiles of tools with respect to only those features of interest or relevance to the specific project. These project-specific tool profiles are then collated with the ideal tool profile (see step 2b) for use in the final stages of the comparison process.

4.2.5.3 Project Specific UIDT Selection Based on Comparison with Previous Projects

Project-specific ideal tool profiles, tailored frameworks, and associated tool information, together with final justified selection decisions, form records of past SUIT-based UIDT selections. When asked to identify his user interface development process, one survey respondent explained that his organisation did a lot of repetitive work where one system was essentially a variation on a previous system (see section 3.4.3.6). In these circumstances if SUIT has been used to inform UIDT selections for previous projects, identification of the 'best-fit' tool for a new project can exploit the results of the tool selection from previous, closely matching, projects.

Figure 4.6 highlights the path through the SUIT methodology which would be followed if a tool selection is to be performed for a project with similar predecessors, which have themselves used the SUIT evaluation framework and methodology to select their UIDTs. This therefore has the potential to facilitate context-sensitive reuse of evaluation results and effort - not just across projects, but also across different organisations. This is further discussed in chapter 8.

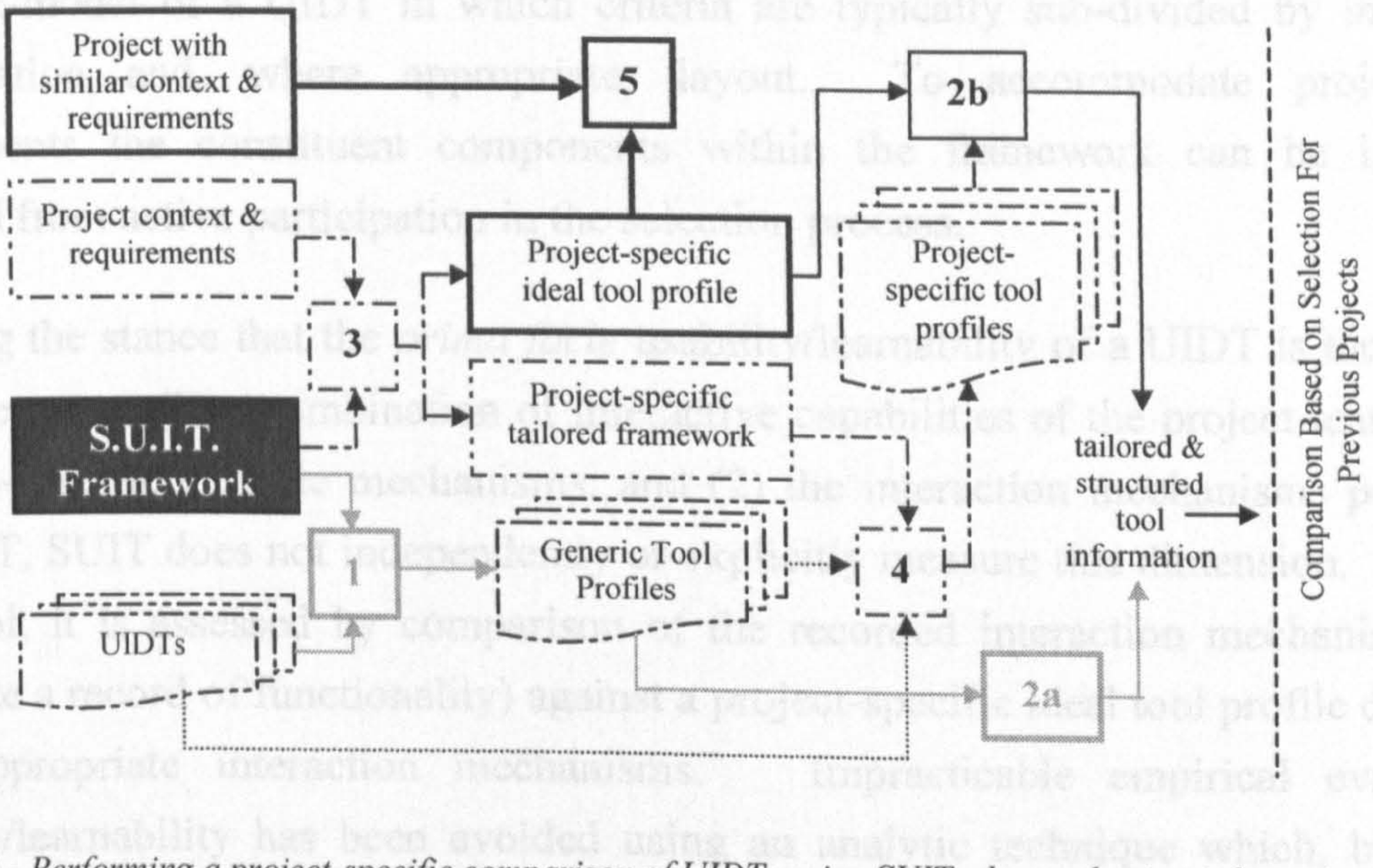


Figure 4.6 - *Performing a project-specific comparison of UIDTs using SUIT where selection information is available for similar preceding projects*

When drawing on information from tool selections for similar preceding projects, the new project context and requirements are examined to determine whether or not they would generate a project-specific ideal tool profile that matches a previous ideal tool profile (see

step 5 in Figure 4.6). If this is found to be the case, the tool recommendation as made for the preceding project would also be the best-fit for the new project.

If there are only slight differences between the ideal tool profile for a new project and an existing project, the project-specific tailored framework and ideal tool profile for the preceding project can be copied and tweaked (that is, minor changes made) and the altered versions used to complete the selection process as described in section 4.2.5.2, steps 4 and 2b.

Given either of the above scenarios, the time and effort expended on previous tool evaluations reduces the cost of tool selection for new projects.

4.3 Summary

This chapter has outlined SUIT - a framework and methodology designed to support contextualised UIDT selection.

Based on the survey results, SUIT asserts that 'context of use' is the result of complex *project-specific* interaction between the development environment, the user interface-specific features of the application under construction, and the developmental processes by which the user interface is generated, and claims that this is influential in the selection of a UIDT. Reflecting these contributory factors, the SUIT framework provides an extensible reference-model of a UIDT in which criteria are typically sub-divided by instantiation, configuration and, where appropriate, layout. To accommodate project-specific requirements the constituent components within the framework can be included or excluded from active participation in the selection process.

Adopting the stance that the *prima facie* usability/learnability of a UIDT is the result of a match between: (1) a combination of interactive capabilities of the project team members and task-specific suitable mechanisms; and (2) the interaction mechanisms promoted by the UIDT, SUIT does not independently or explicitly measure this dimension. Instead, for each tool, it is assessed by comparison of the recorded interaction mechanisms (which constitute a record of functionality) against a project-specific ideal tool profile detailing the most appropriate interaction mechanisms. Impracticable empirical evaluation of usability/learnability has been avoided using an analytic technique which, by explicitly representing and considering the actual team members (that is, context-sensitivity), reduces evaluator bias. SUIT additionally allows modifying comments to be recorded in order to enrich the comparison.

The SUIT methodology supports three modes of use: (1) it facilitates generic UIDT comparison in that a potentially tailored framework can be used to collect and compare context-independent UIDT data; (2) it supports the selection of a UIDT for a specific project whereby a project-specific ideal tool profile embodies the context of use - the functional requirements correlated with the interactive mechanism(s) best suited to the capabilities of the identified team members and/or development activities; and (3) it supports the selection of a UIDT for a specific project on the basis of its similarity with previous projects.

It is not the rôle of the SUIT framework and methodology to make a UIDT recommendation. Rather, it facilitates the collection and organisation of UIDT data such that it can be compared in light of the context of use. SUIT avoids the generation of cumulative measures on the grounds that they hide much useful evaluative information and detract from the context-centric approach.

In conclusion therefore, the combination of an extensible, tailorable, reference model-based framework and a flexible methodology which strongly advocates context-sensitivity is the result of an identified need and demand for improved UIDT selection procedures and a qualitative illustration of the criteria on which this support should be based.

CHAPTER 5 : EMPIRICAL STUDY OF THE USE OF THE SUIT FRAMEWORK & METHODOLOGY

5.1 Introduction

SUIT is based on the key principle that the context in which a UIDT is to be used should be considered the core factor motivating and directing UIDT evaluation and selection. This chapter describes an empirical study designed to investigate the degree to which SUIT directs evaluator attention to issues of context of use during UIDT evaluation and selection. The study, which also examined the use of the Hix *et al* checklists to draw comparison between the different methods, additionally aimed to assess the viability or practicability of SUIT as a paper-based facility for UIDT selection (Hix and Ryan, 1992, Hix and Schulman, 1991, Hix et al., 1989).

This chapter starts by outlining the manner in which the study was conducted. Following analysis of the results, it discusses the experimental validity of the study. The chapter closes with a summary of the conclusions that may be drawn as a result of the investigation.

5.2 The Study

A class of twenty seven final year undergraduate computing science students[55] was divided into two groups - one comprising fourteen students and the other, thirteen students. The students in each group were then divided into pairs[56] and, according to their assigned group, were allocated an evaluation method - either SUIT or the Hix *et al* checklists (Hix and Ryan, 1992, Hix and Schulman, 1991, Hix et al., 1989). The subjects in each group were taught how to use their means of evaluation. During the tutorial session for the SUIT group, the subjects in the other group (hereafter known as the Hix group) were asked to vacate the room (and vice versa) to ensure that only the subjects to whom the method had been allocated were given method-specific tuition[57].

---

[55] These students had all opted to study HCI and over the course of their undergraduate degree (which was drawing to a close at the point at which the study was conducted) had established expertise in the principles and practice of HCI.

[56] The odd number of students necessitated one group of 3 subjects.

[57] Ethically, it could be argued that at the end of the study all subjects should have been tutored in the use of their non-allocated evaluation mechanism in order to avoid disadvantaging any student. However, the students were not being examined on UIDT evaluation and so it was not considered necessary to provide additional tuition in this instance.

The subjects were each given the same problem scenario - see Appendix C - outlining the context and requirements for a project. Each pair was then given 30 hours to complete an evaluation of two web page authoring tools and make a written recommendation for the selection of one. The same set of web authoring tools was made available to all subjects, who were also at liberty to investigate, obtain, and evaluate alternative web authoring tools.

The problem scenario focussed on the evaluation of web page authoring tools for two principal reasons: (1) web page development was familiar to all subjects and as such provided a common basis on which to structure the study; and (2) web page authoring tools are readily available as freeware/shareware, can typically be installed without administrative privileges[58] and, as a genre of UIDT, were familiar to all subjects.

Each pair[56] of subjects within the SUIT group was given three copies of the SUIT framework - one on which to produce the project-specific ideal tool profile, and the remaining two for recording the information about two web authoring tools. They were also given an abridged copy of the SUIT methodology that described only those sections relevant to project-specific UIDT selection for an unprecedented project (see section 4.2.5.2) and that had been edited to remove any mention of comparison validity. Each pair of subjects in the other group was given two copies of the Hix *et al* checklists - one for each web authoring tool.

The subjects in both groups were asked to submit: (1) an individual written report recommending and justifying the selection of one web authoring tool; (2) a log of their time allocation during the evaluation - see Appendix C - which was to remain anonymous; and (3) their collective evaluation sheets.

With the support of the students' class co-ordinator, the study was incorporated into the teaching syllabus. The subjects were accordingly assessed on their submitted recommendation reports and as such were given relevant motivation to complete the study task to the best of their ability.

Upon submission of their required information, subjects were asked to complete a questionnaire based on the NASA Task Load Index (TLX) to gauge their perceived experience of workload as a result of performing the evaluation (see Appendix C) (Hart and Staveland, 1988, Hart and Wickens, 1990).

[58] It was not possible to assign administrative privileges to the subjects for the purpose of the study.

The principal hypotheses of this study were that: (1) SUIT is effective in terms of directing evaluator attention to issues of context of use during project-specific UIDT selection; and (2) SUIT is viable as a paper-based UIDT evaluation and selection facility. Given the variety of formats in which the data was returned (see above), the methodology used to analyse each in order to prove the above hypotheses is described in association with the presentation of the corresponding results.

5.3 The Results

According to type, with the exception of the collective evaluation sheets, all the submitted data (as described above) was analysed - the results are discussed in detail in the following sections of this chapter.

Before discussing the detailed analysis, briefly consider the evaluation sheets. The SUIT subjects were syntactically correct in their use of the sheets. They were (at least superficially) thorough and extensive in their use of the SUIT tables, and there was an observed consistency in the tailoring of the tables across the subjects in this group. The majority of the subjects used the Hix *et al* checklists as directed. However, a number of the subjects in this group made mistakes (often substantial) in their calculation of the various ratings specified by Hix *et al*.

5.3.1 Analysis of the Evaluation Reports

The handout given to each subject stated that the written report should include: (1) a brief outline of the evaluation method used; (2) a short summary of each evaluated web authoring tool; and (3) a justified recommendation for the selection of one of the evaluated tools. The brief method outline was included to indicate the subjects' understanding of their allocated methods. The short tool summaries provided an illustration of the level of detail at which the subjects had considered each tool and a check that they had no serious misunderstandings about either tool. However, the component of the report dedicated to the recommendation for the selection of one web authoring tool was of greatest interest, and its analysis is discussed in detail below. Whilst completing the following analysis, quotes were extracted from the reports in order to qualitatively demonstrate the subjects' opinion of their allocated evaluation methods - these are discussed in section 5.3.4.

5.3.1.1 Analysis of Terminological References

Given the principal aim of this study, the reports were analysed to provide evidence that SUIT is effective in terms of directing evaluator attention to issues of context of use during UIDT selection.

It is hypothesised that the language used by the subjects when justifying their recommendation is an indicator of their primary focus of attention and thus reflects the effectiveness of their allocated evaluation method in terms of directing their consideration to issues of context of use. The language of interest to this study is that which refers to the core issues promoted by the project-specific route through the SUIT methodology - that is, terminology that reflects the ideal tool profile concepts (the project-specific context of use) and associated dimensions of measurement included in the SUIT approach. The reports were therefore analysed to quantify the use of such language in each report, and thus the subjects' focus of attention.

Since the English language permits the synonymous use of terminology, rather than analyse the use of terms on a term-specific basis, the following list of terminological categories was generated:

- *required attributes* - any reference to the functional requirements of a project; in other words, references made to the actual components a tool would have to support in order to meet the needs of the project - typical terms in this category include (non-exhaustively): required attributes; required features; necessary features; necessary attributes; and project needs;
- *specific unsupported attributes* - any reference to a specific component or set of components which was not supported by the tool under evaluation; an example of such a reference would be mentioning that a tool lacked support for including animation in a web page - in other words, any reference to the inability of a tool to support particular named features;
- *general unsupported attributes* - any non-feature-specific reference to unsupported components within the tool under evaluation; for example, if the evaluators mention that the tool lacked 'many of the features' required for the web pages but did not specify which - analysing such references illustrates the level of detail at which the evaluators were considering and then comparing data;
- *specific supported attributes* - any reference to a specific component or set of components which was supported by the tool under evaluation; an example of such a reference would be mentioning that a tool supported the inclusion of animation in a web page - in other words, any reference to the ability of a tool to support particular named features;
- *general supported attributes* - any non-feature-specific reference to supported components within the tool under evaluation; for example, if the evaluators mention that the tool supported the 'majority (or all) of the required features' but did not elaborate further - as with the general unsupported attributes, analysis of terms in this category illustrates the level of detail at which the evaluators were working;
- *ideal tool* - any reference to the kind of tool which would be ideal or best suited to the given project; terms included in this category are (non-exhaustively): ideal; perfect; most suitable; best-fit; and best;

- *ease-of-use/feedback* - any reference to the 'user-friendliness' or quality of feedback inherent in the tool under evaluation; this category includes (non-exhaustively): user friendly; easy-to-use; good/poor feedback; hard; difficult; and frustrating;
- *interaction mechanism* - any reference to one or more interaction mechanisms listed in either evaluation method; in other words, references made to any of the mechanisms of interaction adopted by the tools (for example graphical manipulation); terms include (non-exhaustively): graphical manipulation; direct programming; scripting language; form-filling; and text editing;
- *tool help* - any mention of assistance provided by the tool under evaluation; terms considered within this category include (non-exhaustively): online help; context-sensitive help; assistance; and wizards;
- *team member* - any reference to the team members as described in the project profile; within this category the term 'user' was also considered when used in the context of the project rather than in the general sense - similarly, the term 'developers' was considered synonymous for the purpose of this analysis;
- *learning time* - any reference to the learnability or time taken to learn or become familiar with a tool; this includes mention of the lack of time available for the team members to learn to use the tool;
- *cognitive demand* - any reference to the mental demands placed upon the users of the tool being evaluated; such references include mention of the difficulty or ease of understanding associated with the tool;
- *constraints* - any reference to the constraints placed upon the selection of use of the tool within the context of the project described in the problem scenario - for example, the time constraints under which the team members were expected to work;
- *usability rating* - any direct reference to the usability ratings calculated for the tool under evaluation; obviously, unlike all the preceding terms, the use of this term is only relevant within the reports of the Hix group - it is therefore not applicable when considering the reports submitted by subjects in the SUIT group; this category is included to illustrate the extent to which the usability rating is actually referred when making a UIDT recommendation based on an evaluation performed using the Hix *et al* checklists;
- *functionality rating* - any direct reference to the functionality ratings calculated for the tool under evaluation; as with the usability rating category, the use of this term is only relevant within reports from the Hix group and is illustrative of the extent to which it is referred during recommendations based on this method of evaluation and selection.

The meaning attributed to each term listed above is generally derived from the context in which they are used, and so they were considered relative to the context in which they were referred. In turn, individual reports were examined and an independent tally was generated for the use of terminology within each of the above categories. To verify the

analysis, the examination process was documented and, for a random selection of evaluation reports[59], was repeated by an independent assessor.

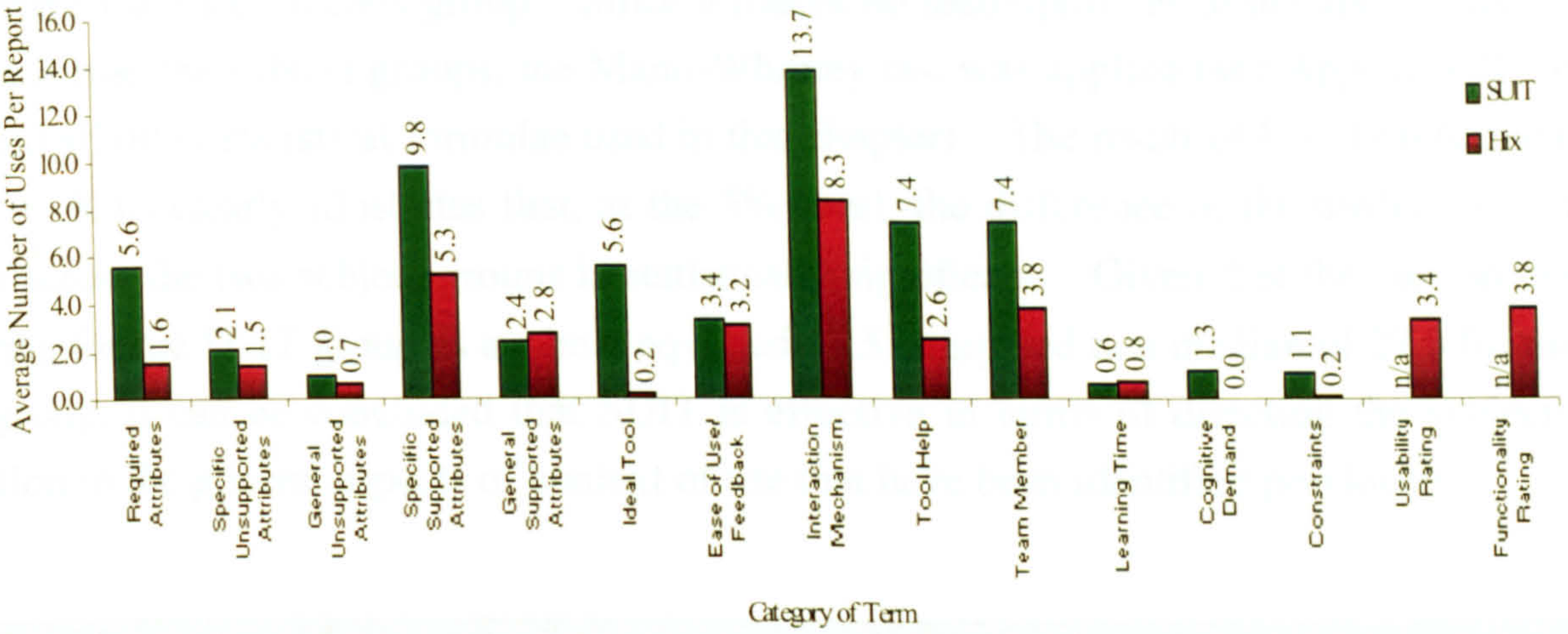


Figure 5.1 - *Average term use for SUIT and Hix*

Figure 5.1 presents the average number of references to each of the terms made by the subjects in each group[60]. At this level, there are a number of apparent differences between the results returned from each group - for example, subjects in the SUIT group made an average of 5.6 references to terms related to the 'ideal tool' in contrast to an average of only 0.2 amongst subjects in the Hix group. Although the majority of the above results suggest that SUIT is effective in terms of directing evaluators' attention to issues of context of use, it is only with closer statistical analysis that the significance of the illustrated differences becomes apparent. The remainder of this section therefore discusses the results more fully.

Before considering the category-specific differences between the two, it is interesting to illustrate the general difference between the groups in terms of the extent to which *any* of the specific references were made. That is, to look at the differences between the average total use of any of the specified terms for the SUIT subjects and those in the Hix group. For the purpose of this comparison the use of terms referring to usability and functionality ratings have been omitted on the grounds that they are specific to the calculations in one method and, as such, would not represent a meaningful comparison. Since the subjects who took part in the study are unlikely to constitute a representative population sample (see section 5.4) it is not reasonable to assume a normal distribution across the results in

---

[59] Approximately 10% of the total number of reports.

[60] It should be noted that the average use of the terms is being quoted in preference to the raw totals since the two groups were not of identical size.

each group. Furthermore, given the relatively small size of each group, it is unlikely that a particularly reliable measure of distribution would be obtained. Hence, it is not possible to attribute statistical significance to the difference in the average total use of terms between each group on the basis of a *t*-Test - an observation which is confirmed by the calculated variance for each group. Since it makes no assumption as to the distribution of results across the subject groups, the Mann-Whitney test was applied (see Appendix D for this and all other statistical formulae used in this chapter). The result of U = 14.0 (critical value = 50.0) clearly illustrates that, at the 5% level, the difference in the median use of terms across the two subject groups is statistically significant. Given that the median use of terms for the SUIT group as a whole equalled 61.5 compared to a median of 27.0 for the Hix group, it can be concluded that SUIT is effective in terms of directing the subjects' attention to the general aspects of context of use that have been identified previously.

TERM CATEGORY	MANN-WHITNEY VALUE	TERM CATEGORY	MANN-WHITNEY VALUE
Required Attributes	**32.0**	Ease of Use/Feedback	77.0
Specific Unsupported Attributes	87.0	Learning Time	86.0
General Unsupported Attributes	69.5	Tool Help	**36.5**
Specific Supported Attributes	**50.0**	Team Member	**28.5**
General Supported Attributes	79.5	Cognitive Demand	**26.0**
Ideal Tool	**0.5**	Constraints	**29.5**
Interaction Mechanism	52.0	*Critical value = 50.0 (at p = 0.05)*	

Table 5.1 - *Lower U values from Mann-Whitney test applied to terminological use (bold = significant)*

The difference in the use of each term category was analysed to determine which received significantly different focus between the two groups. Since it is not possible to assume normal distribution across the results for each group, the Mann-Whitney test was applied in preference to the *t*-Test. The Mann-Whitney value for each test is shown in Table 5.1, in which the statistically significant values are highlighted.

Among the SUIT subjects, the median use of terms concerning project-specific functional requirements (Required Attributes) is four times that of the subjects in the Hix group. Statistically significant at the 5% level, this observed difference shows that SUIT is particularly effective at directing evaluators' attention to the functional aspects relevant to a specific project - that is, SUIT increases evaluator awareness of the project-specificity of requirements such that they are considered more extensively when making UIDT recommendations.

Consider now the cluster of terms concerned with both general and specific support and lack of support for functional attributes. Of these four categories Specific Supported Attributes is the only one for which the difference in use is significant at the 5% level (see Table 5.1) - the median use across SUIT subjects is 8 references compared to a median of 5 for the Hix group. This demonstrates that, although SUIT effects little change in terms of evaluators' *general* perspective of UIDTs' functionality, it is effective at assisting an evaluator to assess which of the individual attributes required by a specific project (see above) are supported by a UIDT. There is no correspondingly significant difference between the median use of terms relating to specific *unsupported* attributes - although, in terms of average use, SUIT is higher.

Based on the hypothesis that a UIDT should be selected on the grounds that it best meets the identified needs of a given project, SUIT has therefore been developed to encourage systematic consideration of those needs - in terms of functionality and context of use - during UIDT selection. SUIT has been shown to be effective in terms of directing evaluator attention to the functional aspects of project-specificity. It is now important to consider the effectiveness of SUIT in terms of directing evaluator attention to the context (essentially environmental) in which that functionality is to be used. The remaining categories of terms focus on this issue.

SUIT embodies what constitutes a project-specific 'best-fit' in a profile of an ideal tool - essentially a correlation between the required functionality and the most appropriate interaction mechanisms. In this respect SUIT is unique, and so it would be unreasonable to assess the ability of different evaluation methods to focus evaluator attention on the match between a given UIDT and the contextual needs of a project purely on the grounds that the evaluators specifically mention the term 'ideal tool' (when documenting their evaluation findings). For this reason the 'ideal tool' category encompasses any mention of what would be best or ideal for a given project (see above). Applying the Mann-Whitney test to the results for this category (see Table 5.1) clearly highlights the significance (at the 5% level) of the difference between the median use of these terms in the SUIT and Hix groups - the median number of references are 5.5 and 0 respectively. It can therefore be comprehensively concluded that SUIT is effective at encouraging an evaluator to consider, and thus recommend, UIDTs on the basis of their fit with the context of use for which a tool is being selected.

Table 5.1 highlights that no significance is attributable to the difference in use of terms relating to either Interaction Mechanisms or Ease-of-Use/Feedback. As the two primary dimensions of measurement within the Hix *et al* checklists, it is not unsurprising that these score highly in the reports submitted by the subjects in the Hix group. Since SUIT

measures both functionality and *prima facie* learnability in terms of employed interaction mechanisms, it is to be expected that terms related to the interaction mechanisms used by a UIDT will also feature strongly in SUIT reports. SUIT assesses *prima facie* UIDT learnability/usability in terms of the match between the interaction mechanisms used and those that are familiar to the project team member(s) and/or most suitable for the project-specific activities - it does not explicitly measure ease-of-use in the same sense, and to the same extent, as the Hix *et al* checklists. Therefore, although it was expected that SUIT reports would include use of terms relating directly to quality of feedback and discussion about interaction mechanism suitability, it was not anticipated that they would necessarily include terminology such as 'user friendly' or 'easy-to-use'. Further examination of the reports led to the observation that, on the whole, reference to ease-of-use or feedback was done within the context of reference to project team members and environment of use within the SUIT reports, and outwith this same context within the Hix group's reports. Hence, although the SUIT reports included terminology that could be construed to reflect evaluator bias, such terminology was actually being used contextually with respect to the project. Albeit qualitatively, this suggests that SUIT encourages consideration of ease-of-use with respect to the project-specific team members. Although it would be interesting to investigate whether the unprompted use of terminology such as 'user friendly' and 'easy-to-use' is inherent in the culture of either the subject population or the task of software tool evaluation, this is beyond the immediate scope of this research and so is relegated to discussion in chapter 8.

Of the remaining terminological categories, only 'Learning Time' lacks significance with respect to the difference in its use between the two groups. Since the SUIT methodology promotes consideration of learnability, it was hypothesised that SUIT reports would contain significantly more references to such terms than reports based on use of the Hix *et al* checklists which make no mention of learnability. This was not found to hold, and so the evaluation reports were again further examined. It was discovered that, in general, within the SUIT reports issues of learnability were referenced in conjunction with references to project team members and time constraints placed upon the project - an observation that was not mirrored within the Hix group's reports. Although this is again a qualitative observation of the strength of SUIT in terms of contextually focusing an evaluator's attention, it is substantiated in part by the fact that there are considerably more references made to team members and constraints within the SUIT reports (see following discussion). The fact that the subjects using the Hix *et al* checklists also considered learning time suggests that, even unsupported, evaluators may consider this aspect of UIDTs. As with the issue of terminological use discussed above, further investigation of this hypothesis is beyond the immediate scope of this research and so is considered in chapter 8.

The use of each of the final four terminological categories - that is: Tool Help; Team Member; Cognitive Demand; and Constraints - is significantly different (at the 5% level) between the two groups (as shown in Table 5.1). In each case, the median use of terms is considerably higher for the subjects within the SUIT group than for those in the Hix group. First consider 'Tool Help'. SUIT promotes interaction assistance as a core dimension of UIDT assessment. In contrast, only a small proportion of one page in the Hix *et al* checklists is devoted to consideration of tool help. On this basis, it was hoped that SUIT reports would include significantly more references to tool help than those from the Hix group. Proven to be the case, this provides evidence that SUIT is effective in terms of directing evaluators' attention to this aspect of UIDT suitability. The apparent lack of importance attributed to tool help within the Hix *et al* checklists is contradictory to the relative importance it receives within the associated recommendation reports (the median number of references to tool help is close to that for general references to supported functionality). Like the issue of learnability, this observed anomaly suggests that even without support evaluators may consider tool help during a comparison of UIDTs. Once again, this is further discussed in chapter 8.

Consideration of project-specific team members is central to the SUIT methodology. The significantly greater reference to project team members within the SUIT reports (at the 5% level) is evidence of its effectiveness at encouraging evaluators to evaluate, and therefore recommend, a UIDT from the perspective of the anticipated end-users (as opposed to their own perspective). This observation is further enhanced by their significantly higher associated use of reference to the cognitive demands imposed upon the team members by the UIDTs. The notable absence of any mention of cognitive demands within the Hix group's reports suggests that - unlike learning time and tool help - unless supported, evaluators may not explicitly consider cognitive demands. This too requires further investigation that is beyond the scope of this research (see chapter 8).

Although the Hix *et al* checklists make no direct provision for recording constraints placed upon a UIDT selection, constraints are likely to be heeded implicitly during evaluation - for example, if constraints are placed on the programming language a UIDT must support, only UIDTs which support that programming language are likely to be considered. Since the problem scenario for this study outlines a number of constraints that should have been observed by the subjects, it was hoped that they would be mentioned in the written evaluation reports from the Hix group as well as the SUIT group. Although this was the case, the significantly more extensive mention of constraints in the SUIT reports (at the 5% level) indicates that SUIT is effective at reminding evaluators to consider this aspect of the context in which a selected UIDT is to be placed and used.

Although the two methods cannot be compared with respect to reference to usability and functionality ratings, it is interesting to consider the extent to which these were referred by the subjects using the Hix *et al* checklists. Despite their priority within the Hix *et al* evaluation procedure and the relative amount of time allocated to their calculation (see section 5.3.2), they were referred to less, on average, than some of the other listed terms - in particular, Interaction Mechanisms and Specific Supported Attributes for which tallies were highest in this group. Although the latter constitutes the foundation on which the functionality rating is based, this finding suggests that subjects preferred to consider provided functionality on a component-by-component basis rather than cumulatively as a numerical rating. As will be discussed in section 5.3.4, the ratings were not considered wholly reliable, appropriate, or relevant by the subjects using the Hix *et al* checklists.

As a final analysis of the recommendation reports, the Chi^2 test was applied to the total tallies for all terms (excluding those for usability and functionality ratings) to assess the degree to which the proportional use of the various terms differed between the two subject groups. The calculated value of $Chi^2 = 79.80$ far exceeded the critical value of 26.22 at the 1% level (assuming 12 degrees of freedom), and therefore provides evidence that as groups, the subjects using SUIT and the subjects using the Hix *et al* checklists differed significantly with respect to their proportional use of the identified terminology. In particular, proportional use of terms relating to required attributes, general supported attributes, ideal tool, ease-of-use, and cognitive demand were most noticeably different between the two groups. This suggests that not only does the median use of the investigated terminology differ as a result of the influence of the SUIT methodology, but the manner in which it is used is also significantly different for each of the two groups.

Collectively, the above observations provide evidence that, in general, SUIT encourages UIDT evaluators to consider context-sensitive aspects when selecting a UIDT for a given project. Furthermore, it has been shown that SUIT significantly increases the extent to which these issues are considered than would otherwise be the case, given either the support provided in the Hix *et al* checklists or what can only be called evaluator intuition in the absence of further research (see discussion regarding Learning Time and Tool Help).

5.3.2 Analysis of the Log Sheets

The subjects were asked to use the log sheets to record the breakdown of their evaluation task and their allocation of time to each sub-task in order to quantitatively assess the manner in which they structured their tasks and the time taken to complete each sub-task, and therefore to complete an evaluation.

Unfortunately the return rate and quality of the log sheets was poor (only 22 were returned, of which only 14 were complete). Despite the anonymity of the logs, subjects may have been suspicious of the intended use of the results (see section 5.4) and therefore chose not to submit them. Alternatively, the apparently incomplete nature of those that were returned suggests that the logs were generally generated after-the-fact, and so in some cases may have been forgotten about altogether. The data contained within the logs is therefore not as reliable as would have been hoped and could not be validated (see section 5.4). The results and associated analysis discussed in this section cannot therefore be considered accurately quantitative. Taken at face value, however, and considered accordingly, the results and subsequent analysis illustrate the reported measurements of evaluation procedure. In the case of the SUIT framework and methodology this is previously untapped information, and is therefore of considerable interest.

The log sheets were examined collectively (irrespective of their associated evaluation method) to determine the sub-tasks identified by the subjects. With the exception of calculating ratings which was particular to the Hix *et al* subjects, the following set of evaluation sub-tasks was easily identified:

- *understanding the method* - any log sheet entry related to gaining an understanding of their allocated evaluation method; only a small proportion of the submitted log sheets included such an entry and although the reason for this is unknown, it is likely to be that the subjects did not consider their learning process to be an integral component of their overall evaluation task;
- *creating the ideal profile/identifying requirements* - any log sheet entry relating to the identification of requirements or physical preparation for the data-collection; in the case of the SUIT subjects, this usually included reference to the generation of the ideal tool profile and tailoring of the SUIT frameworks - similarly subjects in the Hix *et al* group normally mentioned determining what was required and preparing the checklists;
- *using tools* - any log sheet reference to the time dedicated to hands-on use of the tools;
- *performing calculations* - any log sheet reference to the calculations required for the Hix *et al* ratings; naturally this was specific to the subjects in the Hix group and so was absent from the SUIT subjects' log sheets;
- *comparison of the tools* - any log sheet reference to the process of comparing the tools or data collected;
- *analysis of results and write up* - any log sheet entry regarding decision making activities and writing up of evaluation reports.

	Understanding the Method	Creating Profile/ Identifying Requirements	Using Tools	Performing Calculations	Comparison of Tools	Analysis of Results & Write Up
SUIT	n/a	189	297	n/a	95	173
Hix	n/a	25	365	122.5	72.5	150

Table 5.2 - *Average time in minutes for evaluation activities*

Table 5.2 illustrates the average time in minutes allocated to the identified sub-tasks. Average times have not been included for 'Understanding the Method' on account of the scarcity of data across all subjects. The remainder of the averages have been calculated for the number of subjects in each group who returned a completed log sheet[61]. Further averages were calculated for the total time required to complete an evaluation using each method. The average total time spent using SUIT was 765 minutes compared to 750 minutes when using the Hix *et al* checklists.

On the basis of the aforementioned problems with the returned log sheets, together with the fact that not enough data was provided to accurately measure distribution across the two groups, it was not considered viable to run detailed statistical analysis over the data. Instead, the data is considered in terms of the average time allocation to sub-tasks and the average time required to complete an evaluation using each method. No statistical significance is attributed to the associated observations.

The averages for the total time required to complete an evaluation using each method illustrate that, at least superficially, there is little to separate the two methods. This suggests that, despite the alteration of accepted existing activities and the introduction of additional activities, the overall time required to complete an evaluation using SUIT is not appreciably greater than that required for a Hix *et al* checklist-based evaluation. In both cases, it was shown to take each subject an average of just over 12 hours to complete and write up an evaluation.

On the basis of the average times allocated to sub-tasks - as was expected given the activities required to complete tailoring of the SUIT framework and generation of an ideal tool profile - the task of 'Creating Profile/Identifying Requirements' takes considerably longer using SUIT. However, the effort expended in preparatory tasks pays dividends in terms of the amount of time required to use the actual tools in order to collect the necessary UIDT data - SUIT subjects spent on average one hour less on this sub-task.

[61] Since the number of returned log sheets was different for each group, averages have been used in preference to total values.

The calculation of ratings is obviously not applicable to the SUIT subjects. However, it is interesting to note that the Hix subjects took, on average, two hours to complete these calculations. Despite this, as was commented previously, there were many mistakes apparent in the calculated ratings within the submitted evaluation sheets. Subjects using SUIT took, on average, approximately twenty five minutes more to compare their results and similarly on average approximately twenty five minutes more to analyse and write up their results.

Figure 5.2 represents the figures given in Table 5.2 as relative percentages of the total time required to complete an evaluation. This highlights that using SUIT, approximately 25% of evaluation time is spent on preparatory work. In comparison, a mere 3% of the overall evaluation time is dedicated to preparation using the Hix *et al* checklists. Approximately 39% and 49% of evaluation time is consumed by 'hands-on' use of the UIDTs when using SUIT and the Hix *et al* checklists respectively. This further highlights the earlier observation that the preparation necessary to perform a SUIT evaluation reduces the time evaluators require to extract the important information from the tools - by assisting the evaluators to identify and record which features should be investigated, SUIT lets them focus on only relevant features and so use their time with the UIDTs more efficiently.

On average, over 16% of evaluation time is used to calculate the Hix *et al* ratings. The relative lack of importance these ratings were shown to have within the recommendation reports, together with the subjects' response to their intrinsic value (see section 5.3.4), suggests that this is not a particularly productive (nor cost effective) use of evaluator time.

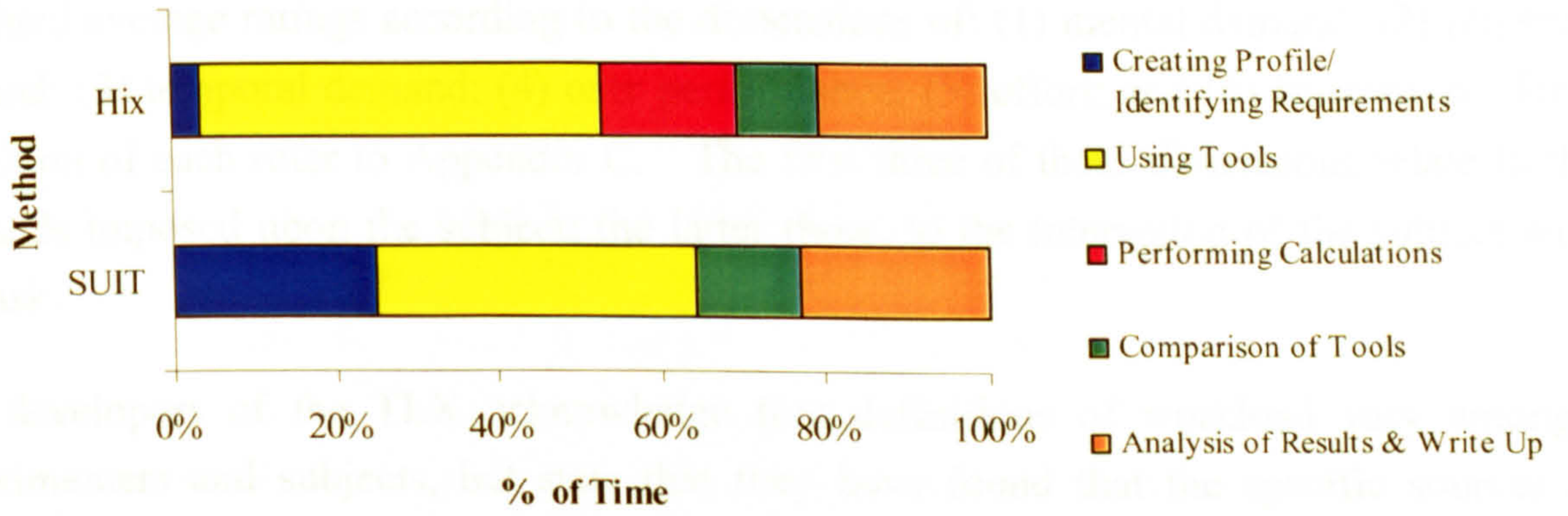


Figure 5.2 - *The relative percentage of time allocated to evaluation sub-tasks for SUIT and Hix*

Approximately 12% of evaluation time is spent on tool comparison using SUIT compared to approximately 9% using the Hix *et al* checklists. Analysis and write up of evaluation results accounts for approximately 23% and 20% of overall evaluation time using SUIT and Hix respectively. Although, when combined, these two tasks in SUIT account for a

slightly higher percentage of the overall evaluation time (as was expected given the additional considerations introduced by SUIT), the preceding section has shown that the net result is a UIDT recommendation that considers components of the UIDT within the context of its intended use (see section 5.3.1.1).

The nature of SUIT therefore dictates that effort expenditure is heaviest at the start and end of the complete process - that is, preparation and comparison or analysis - as opposed to the central activities of data-collection. Conversely, the Hix *et al* checklists necessitate greatest allocation of time to the collection of data and calculation of results.

5.3.3 Analysis of the Workload Questionnaire Responses

Immediately following their submission of the requested documentation at the close of the evaluation exercise, the subjects were each asked to complete a questionnaire designed to measure their perceived workload experience during the study. The questionnaire used the NASA Task Load Index (TLX) Version 1.0 and is included in Appendix C (Hart and Staveland, 1988, Hart and Wickens, 1990).

Before analysing the results of the workload questionnaire it is worth considering the structure of the questionnaire and thereby briefly introducing the TLX.

5.3.3.1 NASA Task Load Index (TLX)

The NASA Task Load Index (TLX) was developed by the Human Performance Group at the NASA Ames Research Centre (Hart and Staveland, 1988, Hart and Wickens, 1990). It is a multi-dimensional rating procedure that provides an overall workload score based on weighted average ratings according to the dimensions of: (1) mental demand; (2) physical demand; (3) temporal demand; (4) user performance; (5) effort; and (6) frustration - for a definition of each refer to Appendix C. The first three of these dimensions relate to the demands imposed upon the subject; the latter three, to the interaction of the subject with the task.

The developers of the TLX acknowledge that definitions of workload vary amongst experimenters and subjects, but state that they have found that the specific sources of loading imposed by different tasks are a more important determinant of workload experiences. The TLX therefore combines ratings that subjects weight according to their subjective importance in the specific task (Hart and Staveland, 1988, Hart and Wickens, 1990).

The degree to which each of the six listed dimensions contributes to a subject's experience of workload for a given task is assessed according to the subject's response to an

exhaustive series of pair-wise comparisons of the dimensions (see Appendix C). A subjective magnitude rating for each dimension is also recorded. Combining these responses enables the dimensions deemed most significant with respect to workload to be given a greater weighting with respect to the overall workload score, and thereby the sensitivity of the scale is increased. During evaluation of the TLX, NASA found that these derived workload scores led to significantly less variation between subjects than was found for uni-dimensional ratings (Hart and Staveland, 1988, Hart and Wickens, 1990).

5.3.3.2 The Workload Questionnaires

To compare the subjects' reactions to their allocated methods, it was decided that the subjects should be asked: (1) to rate the difficulty of understanding and using their allocated method; (2) to indicate the extent to which time pressure affected them during the course of their evaluation task; (3) to assess the effort they needed to expend in order to complete their evaluation task; (4) to indicate their confidence in the results of their evaluation; and (5) to illustrate the degree to which their allocated evaluation method annoyed or angered them.

Not only did the NASA TLX neatly incorporate all of the above factors, but it provided a clear, comprehensive, tried-and-tested means[62] by which to elicit subject responses to these factors and, moreover, to determine an overall impression of the subjects' perceived experience of workload. The NASA TLX was therefore incorporated into a questionnaire that was distributed to each of the subjects upon completion of their evaluation and submission of their requested documentation (see Appendix C). The return rate for the questionnaires was 93%.

5.3.3.3 Analysis of Subject Perceived Workload

When discussing analysis of terminological use in the recommendation reports, it was noted that the *t*-Test was not an applicable measure of statistical significance on the grounds that the results could not be considered to be normally distributed across the subjects in each group. For this reason, the Mann-Whitney test was used to assess statistical significance. The same rationale applies when considering the results obtained from the workload questionnaires - with the exception of comparison of the average overall workload scores. In the case of the latter, as suggested by NASA, variance between subjects was demonstrated to be very similar across both groups (for formula see Appendix D) and so it was considered justifiable to adopt the *t*-Test to measure the significance of the difference between the average overall workload for each group. The

[62] Albeit the validation of the measure was carried out in a rather different setting compared to this study.

mean overall workload score for the SUIT group was calculated to be $\bar{x}$= 12.22 and for the Hix group it was calculated to be $\bar{x}$= 11.89. When the *t*-Test was applied, the resulting value of t_{23} = 0.363 fell far short of the critical value of 2.069 at the 5% level. It can therefore be concluded that there is no statistical significance attributable to the difference in the average overall workload scores recorded for each group.

Since the average overall workload scores represent an average measure of the subjects' perception of the work considered necessary to complete their tasks, the fact that there is no statistical significance between these average scores for the two groups illustrates that, despite introducing more preparatory tasks and comparison and analysis considerations into the evaluative process (see section 5.3.2), SUIT does not significantly increase the overall impression of workload experienced when completing an evaluation.

The weighted ratings for each dimension (with the exception of physical demand) for each group were examined and compared to see whether there is a significant difference across any of the contributory dimensions. Physical demand was interesting to superficially observe, but since the evaluation task was not a truly physical task, it has been omitted from detailed analysis. Although it was generally considered to be the case that the *t*-Test was not applicable for analysis of the weighted ratings, this was confirmed by assessing the variance and normal distribution across the subject groups for each of the five examined dimensions. In general, as expected, the variance between the subjects in the two groups was found to differ greatly - there was not a normal distribution across the subject groups. The Mann-Whitney test was therefore applied to assess the significance of the difference in the median weighted rating for each of the dimensions registered by the subjects in each group. The resulting Mann-Whitney value for each test is shown in Table 5.3 in which the significant value(s) are highlighted.

DIMENSION	MANN-WHITNEY VALUE
Mental Demand	**37.0**
Temporal Demand	60.5
Performance	43.0
Effort	54.0
Frustration	49.5
Critical value = 41 (at p = 0.05)	

Table 5.3 - *Lower U values from Mann-Whitney test applied to weighted ratings(bold = significant)*

Mental Demand is the only dimension for which the difference in the weighted ratings is statistically significant (at the 5% level). Across the subjects in the SUIT group, the median weighted rating for mental demand is 21.0 in comparison to only 7.5 for the Hix

group. This provides evidence that SUIT increases the thought process that is required to achieve an evaluation, and is consistent with the preceding findings in which SUIT was shown to be significantly better at directing evaluators' attention to context-sensitive issues, and that preparatory and comparison or analysis activities consume the greatest proportion of time when using SUIT.

The fact that the median weighted ratings for Temporal Demand are not significantly different substantiates the observation made in section 5.3.2 - that there is nothing to separate the two methods in terms of the overall length of time required to complete an evaluation - and suggests that the constraint of time is felt similarly across all the subjects irrespective of their allocated evaluation method.

Despite the significantly higher levels of mental demand required to complete a SUIT evaluation, the median weighted ratings for Effort are not significantly different. This suggests that, although the subjects using SUIT were required to consider more (potentially complex) factors (mental demand) this did not impact upon the overall effort considered necessary to complete an evaluation. At the level of conjecture this could be due to the difference in the amount of time required for data-collection during 'hands-on' use of the tools (see section 5.3.2). This therefore suggests that although the focus of effort differs between the two methods, the overall perception of the effort required to complete an evaluation is considered to be similar. There was not a significant difference between the two groups in terms of their perceived levels of frustration. The source of this frustration is discussed in section 5.3.4.

Although the result of the Mann-Whitney test does not indicate that there is a significant difference between the groups' median weighted ratings for performance, since the returned value is very close to being statistically significant, it is interesting to consider this dimension further. In contrast to the other dimensions, it is desirable for subjects to return a high weighted rating for performance - being a subjective measure of how well subjects rate their performance, a high rating corresponds to confidence in their performance and conversely a low value to uncertainty in their performance. The median weighted ratings calculated for the subjects in the SUIT and Hix groups were 22.0 and 13.5 respectively and, although no statistical significance can be attributed to the difference in these medians, they suggest that the SUIT subjects were generally more confident in their own performance than the subjects in the Hix group.

Since the overall workload experienced by the subjects is influenced by each of the dimensions measured, it was decided to determine if there is any correlation between the various dimensions within each group. To this end, the Pearson (r) coefficient of correlation was calculated (see Appendix D) using the weighted rating for each pairing of

dimensions (again excluding physical demand) independently within each group. The results obtained for the SUIT subjects are shown in Table 5.4 and the results for the Hix subjects are listed in Table 5.5 - for both tables, the statistically significant (at $p = 0.05$) instances of correlation are highlighted.

	MENTAL DEMAND	TEMPORAL DEMAND	PERFORMANCE	EFFORT	FRUSTRATION
MENTAL DEMAND		**-0.678**	0.434	0.286	-0.526
TEMPORAL DEMAND	**-0.678**		-0.180	-0.081	0.220
PERFORMANCE	0.434	-0.180		0.232	0.199
EFFORT	0.286	-0.081	0.232		-0.550
FRUSTRATION	-0.526	0.220	0.199	-0.550	

Table 5.4 - *Coefficient of correlation (Pearson r) for weighted ratings of SUIT subjects (bold = significant)*

	MENTAL DEMAND	TEMPORAL DEMAND	PERFORMANCE	EFFORT	FRUSTRATION
MENTAL DEMAND		-0.044	**-0.649**	0.176	-0.293
TEMPORAL DEMAND	-0.044		-0.088	-0.027	-0.163
PERFORMANCE	**-0.649**	-0.088		0.026	0.127
EFFORT	0.176	-0.027	0.026		-0.193
FRUSTRATION	-0.293	-0.163	0.127	-0.193	

Table 5.5 - *Coefficient of correlation (Pearson r) for weighted ratings of Hix subjects (bold = significant)*

There is almost no significant correlation between the weighted ratings in either group. For SUIT, a significant (negative) correlation is only observed between weighted ratings for mental and temporal demands. A possible interpretation of this is that increased mental demand results in reduced temporal demand, and vice versa. Although this does not represent a causal relationship between the two dimensions, it suggests that the more thought that is applied to the preparatory work when using SUIT, the less time is required overall to complete an evaluation - since only relevant components are considered during data-collection, time is used more efficiently. Similarly, when comparing and analysing the collected data, the initial mental effort expended to prepare for the evaluation may be effective in terms of reducing the time required to identify and therefore compare the most important data. A similar strength of correlation along these dimensions is not present for the weighted ratings in the Hix group.

In the Hix group the only statistically significant correlation is between mental demand and performance, and is again negative. A possible interpretation of this is that increased

mental demand results in a drop in subjects' confidence in their performance, and vice versa - again, this cannot be considered a causal relationship. This could be a result of the complex calculations that are required to produce the various Hix *et al* ratings - the harder the subjects found the calculation, the less confident they were in their overall performance. The corresponding coefficient of correlation for the SUIT subjects, although not statistically significant, is at least positive - that is, increased mental demand corresponds to increased confidence in performance. Had this been statistically significant it might have suggested that the more thought a subject put into an evaluation, the more confident the subject might be that all the relevant issues had been covered and hence, that the resulting recommendation was reliable[63].

Finally, Chi^2 was calculated to assess whether the distribution of workload across the various dimensions was significantly different for each group. The returned value of Chi^2 = 217.3 far exceeds the critical value at the 1% level and so it can be conclusively stated that the distribution of perceived workload is significantly different for the two groups. In other words, the SUIT and Hix subjects apportion workload differently across the various dimensions. The most noticeable difference lies within the distribution of perceived mental demand - an observation that is substantiated by the findings in all other aspects of this empirical study

5.3.4 Reflection on Subjective Responses to the Evaluation Methods

Although statistical significance cannot be afforded to the opinions expressed by the subjects in this study, their comments are valuable in that they provide a qualitative illustration of their reaction to their allocated evaluation methodologies. Whilst not a quantifiable measure of the success of the SUIT evaluation methodology, this feedback can be used to inform future development of SUIT. This section highlights selected quotes[64] taken from the subjects' evaluation reports. To organise the following discussion, where possible the quotations have been grouped according to their associated evaluation methodology and therein according to topic.

Given the circumstances of this study, there is a high probability that the Hawthorne Effect (and possibly other subject bias) will have influenced the subjects' comments (see section 5.4). Each of the following quoted comments should therefore be considered with this in mind.

---

[63] It should be stressed that none of the noted instances of correlation represent a causal relationship between the dimensions. In the associated discussions, hypotheses are merely presented as to the possible cause of correlation. Further investigation - which is beyond the scope of this research - would be required to test these hypotheses.

[64] Identified on the basis that they represent an expression of subjects' opinion aimed directly at their allocated evaluation method.

5.3.4.1 Comments Directed at SUIT

A number of the evaluators commented positively regarding SUIT, including its perceived strength as a reference model and UIDT comparison guide, and its ability to focus evaluators' attention on specific details and to determine UIDT suitability. One subject commented on the manner in which he approached his task and one criticised the framework.

The Strengths of SUIT as Systematic Evaluation Guide

A number of the subjects commented on the strengths of SUIT in terms of its ability to guide them through a systematic evaluation procedure:

> *"the SUIT method was useful in performing the comparison because it provided a reminder of all possible comparable aspects of the tools. Without the SUIT framework, the evaluation would have been much more heuristic in nature, and may have degraded into a vague comparison of ease-of-use"*

and

> "[SUIT] *directed your considerations to specific things and ensured that you didn't overlook any details of user interface design. It proved to be hard to split the two packages we compared as they were very similar. In the end, the recommendation came more out of discussion and consideration of certain aspects of the scenario rather than from some metric derived from the framework"*

and

> *"Time is important for the current project* [referring to the problem scenario] *so the amount of time to learn to use a new environment can be drastically reduced if the environment can be made similar to one that the target-user already knows"*

The above can be summarised as highlighting the following points:

- SUIT provides a structure that usefully reminds and guides an evaluator in terms of what features to consider and what to measure for each feature;
- SUIT records information in such a way as to enable evaluators to assess the degree of difference between tools and to support and inform their decision making process;
- SUIT encourages evaluators to consider project-specific constraints.

The first comment suggests that without a framework and methodology to inform an evaluation, there is a risk that evaluation will resort to a comparison of evaluator-perceived ease-of-use. The proposal that, as a result of this the evaluation would become vague,

suggests that the subject considers that an ease-of-use comparison might not be adequately quantitative.

In the second comment the subject made reference to the *'aspects of the scenario'* as the core motivation behind the recommendation, indicating that SUIT had been effective at directing his attention to the context of use. Despite the availability of the ideal tool profile - effectively a model of the correlation between the requirements and context specified in the problem scenario - the subject returned to the textual problem definition as the source of this information. General discussion with the subjects using the SUIT evaluation methodology highlighted that the volume of paper proved to be a hurdle with respect to the management and comparison of data. This unmanageability is likely to be a reason behind the subject's reversion to the problem scenario. Assuming this to be the case, it indicates a need to maintain and visualise the data electronically (see chapter 6).

The second comment, and those in section 5.3.4.2, support the decision to avoid cumulative ratings in SUIT and illustrate the strength of providing the data in such a way as to highlight the specific differences between the tools and thereby inform discussion at this level.

The final comment suggests that the decision to focus on learnability rather than ease-of-use, and the manner in which *prima facie* learnability is measured in SUIT, has been accepted by the subject. It provides further evidence that SUIT successfully focuses subjects' attention on the environmental context of use - it encourages evaluators to perform a context-sensitive evaluation.

Project-Specific Data Comparison and Analysis Strategies

One subject commented on his data comparison and analysis strategy:

> *"*[data was] *compared page wise and conclusions drawn* [as to] *which tool (if any) fitted the ideal more accurately. This gave a more accurate picture of the relative strengths and weaknesses of each tool than would otherwise have been possible"*

and

> "[the comparison] *involved individually comparing the provision of each* [required] *attribute and attempting to decide which tool was better in each case. Collectively, this then resulted in an overall conclusion as to which tool was better suited to our scenario"*

Each of the above comments (from the same subject) suggests that, despite the identified inconvenience of manipulating several lengthy paper-based frameworks, the ideal tool

profile is nevertheless used directly and is central to the evaluative process. Furthermore, only relevant components are examined on an individual basis and compared across each of the tools.

These comments indicate that the subject adopted a systematic approach to data comparison to which project-specific requirements were continuously referred. Further investigation into adopted comparison and analysis strategies is documented in chapter 7.

Justification for Using SUIT

SUIT has not been developed to make effective UIDT evaluation less time consuming - the methodology alone suggests that accurate project-specific UIDT selection requires considerable expenditure of time and effort. Some might question the justification of expending this amount of time and effort on the selection of a UIDT. Although the counter arguments to this are clearly demonstrated in chapters 1, 2 and 3, the next quotation - which was taken from a point in an evaluation report immediately after a discourse in which the subject explained that neither of the two evaluated tools matched the project-specific requirements or context - lends further support:

> *"it would probably be worth the time investment to investigate another tool or two to find one that would be more suitable. The time required for this would be likely to be lower than the time which would be required for the team members to gain competency in HTML, which would be required if either of these two tools was to be used"*

This comment suggests that SUIT has been effective at encouraging the evaluator to maintain a context-sensitive focus during his evaluation to the extent that he has recognised that neither of the evaluated tools is particularly suitable within the given context and establishes the worth in further UIDT investigation. This illustrates the trade-off that is essential regarding the investment of time and effort in comprehensive UIDT evaluation - that is, that the time required to complete or extend an evaluation may ultimately be less costly than bypassing or restricting the evaluation, whereby inappropriate tool selection is made and at best resulting integration costs are increased, and in the worst case the tool is abandoned altogether (see section 2.4.2).

Using the SUIT Framework

One subject criticised the SUIT framework tables:

> *"Doing the ideal profile was very confusing. The framework was of no help. Pages are cluttered and glossary of little help"*

This subject's frustration is consistent with the high frustration levels recorded for both groups of subjects (see section 5.3.3). Informal discussion with the subjects confirms that their principal objection to the SUIT approach is the large quantity of paper that has to be organised, the tables thereon completed, and then the contained data compared. This strongly suggests that there is a need for automated support for the activities involved in SUIT evaluations (see chapter 6).

5.3.4.2 Comments Directed at the Hix et al Checklists

In general, the subjects using the Hix *et al* evaluation checklists were more critical of their allocated method. A selection of their comments are discussed in the following sections.

Problems With Cumulative Ratings

A number of the subjects commented critically with respect to the cumulative usability and functionality ratings promoted by the Hix *et al* checklists. These comments include:

> *"although Tool A scored marginally higher than Tool B on the usability ratings, I would offer the personal opinion that this does not reflect the fact that, in some cases, Tool A can be unintuitive"*

and

> *"this is a case which highlights a flaw in this methodology: Tool A suffered from the same 'importing' problem, yet because it performed other functions, its rating was pulled up....cases could occur which distort results unfairly"*

and

> *"we feel that the figures obtained do not convey the extent of the gap between the two packages. For an evaluation method which attempts to produce results in the form of hard figures, we feel that this is a shortcoming"*

and

> *"the results are not very meaningful and it is hard to get an idea of the flavour of the tool only by reading the results. The results are too abstract and often merge a wide range of functionality, which is not of any relevance, with a few points, which are of high relevance"*

The above criticisms of cumulative ratings can be summarised as:

- they hide important information;
- they distort the results;
- they hide the extent of the differences between the tools;
- they obscure the relevance of individual components.

Together, the four comments substantiate the hypothesis of SUIT that it can be the small differences between tools that determine the absolute suitability of a tool, and therefore tools need to be compared according to their constituent components rather than as a set of cumulative ratings.

Consider the first comment in which the subject notes that the usability rating was higher for the first tool despite instances wherein interaction was 'unintuitive'. The subject states that the cumulative rating of usability hides important evaluative information. SUIT avoids such ratings for this very reason, and instead promotes the individual examination and comparison of component data by means of structured project-specific focus.

The subject's measurement of usability is a reflection of his own experience with the tool - *'Tool A can be unintuitive'*. This illustrates one of the principal criticisms of the Hix *et al* approach - that an evaluation conducted in this manner may lead to recommendation based on the preferences or capabilities of the evaluator as opposed to the match between the UIDT and the actual end-users (see section on Evaluator Bias).

The second and third comments focus on further problems with cumulative ratings. Furthermore, they suggest that it is important to consider components in the checklists on an individual level. Despite the fact that the Hix *et al* approach promotes (and only fully supports) generic UIDT evaluations, these subjects identified the need to treat components individually to assess the extent of the differences between the tools. SUIT addresses this need by manner of its project-specific tailorability and avoidance of cumulative ratings.

Highlighting a difference in relevance between components in the checklists, the fourth comment indicates that the subject considers the context of the project to be important (his mention of 'relevance' with respect to listed components) and that, as such, all components should not be treated equally or cumulatively. Suggesting that the subject does not consider the cumulative ratings helpful in comparing important issues, the comment essentially supports tailorability of evaluation criteria as a means by which to compare the relevant components. This is the basis of the SUIT approach.

Evaluator Bias

Some of the subjects noted the influence of evaluator bias in terms of expertise - both tool experience that was obtained prior to the evaluation session, and that which was developed during the evaluation itself:

> *"one of the main problems during evaluation was the rating of 'ease-of-use'. The longer the evaluation process went on, the more comfortable we felt by using a particular tool and therefore the usability ratings got higher in the end of the process"*

and

> *"evaluation form is very verbose and much is inapplicable. Unrealistic to expect good results unless evaluators are experienced with the application"*

These comments suggest that the experience gained by an evaluator:

- prior to an evaluation will influence his ability to generate accurate results for any given tool, which will in turn introduce variation between the quality of the results for different tools;
- during an evaluation will generate variation in the accuracy or response to usability measurements between the assessments made early in the process and those made nearer the end;

and the first of the comments in the previous section adds:

- prior to an evaluation will determine his preferences and therefore response to usability ratings.

These comments collectively suggest that the experience of an evaluator has the potential to distort the results of an evaluation. In particular, they focus on its influence on the measurement of usability as promoted by Hix *et al*, and suggest that the validity and reliability of the usability measurements can therefore be called into question.

The first comment suggests that, even within the duration of a single evaluation, his progressing experience has been reflected in the usability ratings he afforded the UIDT. The effect of evaluator familiarity with UIDTs can only be more pronounced across a range of tools, some of which are known to the evaluator and others which are unknown.

The exact meaning of 'good results' in the second comment is unclear. The subject may be referring to the unreliability of results given an evaluator's lack of experience or equally may be suggesting that the usability ratings recorded for a UIDT with which the evaluator is familiar are likely to be higher than for one with which he is unfamiliar. Irrespective of the actual meaning, this comment further supports the observation that when usability is measured in such a way as it reflects the evaluator's opinion, the results are exposed to evaluator bias.

SUIT addresses this problem in two ways. Firstly, it does not measure usability directly, but instead promotes the notion of *prima facie* learnability as a measure of the match between recorded project-specific ideal interaction mechanisms and the actual interaction mechanisms employed by the UIDT. Additionally, the interaction assistance provided by a UIDT is recorded to supplement the interaction mechanism information. Secondly, SUIT promotes the idea of a library of UIDT templates that have been (potentially

collaboratively) generated by experts in the various UIDTs and that are then available for use in UIDT selection processes. This concept is further discussed in chapter 8.

Lack of Tailorability According to Tool Type & Project Requirements

Subjects using the Hix *et al* checklists were aware of the irrelevance of many of the components in the lists:

> *"a methodology designed specifically for web authoring tools would have offered more rewarding feedback...a number of questions we found ourselves answering were of no relevance to web authoring tools"*

and

> *"as the method is to be used with a variety of UIMSs, it was frequently the case that pages could be completely missed out. I feel that it would be better if we could define a set of rules for the system we wish to create and then use them as a basis for our evaluation"*

In summary, these comments suggest the need for tailorability of the checklists according to:

- UIDT type;
- project-specific functional requirements.

Together, these comments indicate that the subjects wanted to tailor the checklists but were unsupported in this respect by the Hix *et al* approach, and suggest that there is a genuine requirement to support tailorability of the checklists on the basis of tool type. Mosley introduced tailorability according to tool type in her CASE tool evaluation methodology (Mosley, 1992, Mosley, 1995). Valaer and Babb devised a categorisation of UIDT types which was further manipulated in chapter 2 to demonstrate the non-linear relationship between the various categories (Valaer and Babb, 1997). On the basis of this categorisation it should therefore be possible to support tailoring of the complete component set in both the Hix *et al* checklists and SUIT framework, in line with the work in the field of CASE tool evaluation. This is a feature of UIDT evaluation that would benefit from automated support (see chapters 6 and 8).

The subjects in the SUIT group made no similar comment regarding the suitability of the framework for evaluation of web authoring tools. It is hypothesised that this is due to the facility and encouragement in SUIT to tailor the component lists to only consider components relevant to the current evaluation. As evidenced by their completed SUIT evaluation sheets, the subjects using SUIT tailored the framework for web authoring tools as a matter of course. That is not to say that they would not have appreciated some

assistance in this respect, but rather that the framework and methodology made it possible to achieve this (obviously necessary) tailoring. In contrast, the inability to tailor the Hix *et al* checklists made the need to tailor according to type all the more evident.

Missing Categories and/or Dimensions

The structure of the Hix *et al* checklists came under some criticism:

> *"although the results do not show this, Tool X can create a page within a matter of minutes"*

and

> *"the evaluation merely measures the presence of support for an interface style and does not attempt to measure the depth to which that interface class is supported"*

and

> *"doesn't necessarily fit well into one of these* [usability options] - *not consistent categories. Could be simple steps but takes many tries to get it right"*

These criticisms can be paraphrased as:

- there is no facility to record miscellaneous or non-standard information;
- interface components are not assessed in sufficient detail;
- the available usability options are incomplete and inconsistent and the checklists do not provide scope for flexibility of usability assessment.

Common to the above, is subject dissatisfaction with the facility to record non-standard information. The first comment illustrates a situation where the subject has made an important non-standard observation but, since he is unable to record it, it is not reflected in the evaluation results. The last comment further supports the provision of flexibility in the framework so that the evaluator is not restricted to a pre-defined set of options in each measurement dimension. The second comment suggests that the subject wanted to consider the individual interface components rather than just assess these features on the basis that the UIDT supports the style, and perhaps to assess each of them against additional dimensions.

SUIT addresses the first of these issues via the provision of a facility to record modifier or miscellaneous comments at the level of individual components. Anomalous features of the tools can therefore be recorded and included in the decision making process. To address the second issue, SUIT not only enables evaluators to consider individual interface components, but it further sub-divides the examination of components into their

instantiation, configuration, and layout in order that the contributory factors of using an interface feature can be independently measured. SUIT also enables an evaluator to record information such as the interaction assistance, cognitive demand, and quality of feedback for each of these sub-components in order that evaluators can record as much or as little information as they want. Furthermore, the measurement options provided in SUIT for interaction mechanisms and interaction assistance are tailorable and extensible. SUIT therefore provides flexibility in terms of the level of detail at which the interface (and indeed all other components) can be evaluated.

Using the Checklists

Like SUIT, there were complaints about the use of the Hix *et al* checklists:

> *"Overall, I found the ticking part of the method tedious and repetitive and often found the terms misleading - even with the help of the glossary"*

This confirms the need for automated support for the use of the checklists. The Hix *et al* checklists are actually available in electronic spreadsheet (textual) format. However, for the purpose of this study, both methods were used as paper-based versions for comparability.

5.4 The Experimental Validity of the Study

5.4.1 Potential Limitations in the Experimental Design and Execution

Since the intended user population for both methods is professionals involved in the field of user interface development - including graphic designers, cognitive scientists, and software engineers - the subject group cannot be considered adequately representative. By selecting a class of university students an elite bias was introduced to the sample - the subjects did not adequately reflect the differences in educational background, work ethic, levels of experience, mental attitude, and age that are likely to occur within the intended user population. However, limited resources dictated that this sample population be used; that said, although not wholly representative of the user population, the majority of the subjects can be considered usability experts given their chosen specialisation (see footnote 55) and are likely to become part of the intended user population after graduating.

When tutoring the subjects, researcher bias may have been introduced into the study. Although the subject groups were independently tutored in the use of their allocated method to prevent subjects acquiring detailed knowledge of the method other than that which was allocated to them, the methods were not kept anonymous. Subjects were thus

made aware of the method that was developed by the researcher and subsequently were able to determine what feedback would be most amenable.[65] Furthermore, it was not possible to prevent the subjects discussing their allocated methods during the course of the study. That said, the subjects in neither group knew the manner in which their submitted work was going to be analysed and they did not know what were considered to be the shortcomings of the Hix *et al* approach.

Consider now, the allocation of subjects into the two groups. The subjects were divided into two groups by the researcher. Threatening the validity of the study, it could be argued that the partitioning of the subjects may have resulted in two groups neither of which were representative of the class as a whole. This is not, however, considered to be the case for two principal reasons: (1) since none of the subjects were known to the researcher prior to the study, the allocation of groups did not reflect researcher bias in terms of prior knowledge of the capabilities of individual subjects; and (2) the average grade for each of the subjects was obtained in order to determine whether there was an obvious intellectual bias between the two groups - the grades suggest that, at least in terms of their ability to score on academic assessment, there was no clustering or bias among the students in each group.

Subject motivation is often a difficult aspect of empirical studies. In this case, the subjects were being assessed on the recommendation reports submitted at the end of their evaluation exercise. Although this provided real and relevant motivation not only to complete the evaluation, but also to conduct it to the best of their abilities, the motivation is not representative of motivation in the intended user population. The fact that they were being assessed on their submitted reports may have exacerbated the Hawthorne Effect; the subjects in the SUIT group may have worked harder and for longer to complete their evaluations given that they knew the results were going to be used for further research. However, since the primary concern of the subjects in both groups was clearly to complete the exercise to achieve a high academic grade, and the difference in the two groups' median weighted ratings for temporal demand and effort were not statistically significant, these concerns are minimal.

A further potential flaw with respect to the sample population was the omission of action to assess the prior knowledge and attitudes of the subjects regarding evaluation methodologies - both prior and after using their allocated method. Hence, it is possible that implicit subject bias has been overlooked during the execution of this study.

[65] It is also possible that researcher bias may have been introduced subconsciously by the researcher by virtue of aspects such as the language used and body language demonstrated during the tutorial sessions; this cannot, however, be substantiated.

It could be argued that the subjects' choice of candidate tools for evaluation may have influenced the results obtained and that each pair of subjects should therefore have been given the same set of web page authoring tools to evaluate. However, since the freeware/shareware web page authoring tools available for use during the study were generally very similar in terms of functionality and look and feel, the potential for differences between subjects' choice of candidate tools for evaluation to influence the results was minimised. Furthermore, it was considered important to structure the problem scenario such that it was as realistic as possible within the limitations imposed upon the study; to have overly restricted the candidate tools would have lessened the realism of the task.

Finally, there are a number of issues concerning the execution of the workload questionnaires. NASA TLX studies are usually performed immediately upon completion of a short-term, uninterrupted (normally manual) task. In the case of this study, the questionnaires were distributed upon completion of an extended task during which work was intermittent and interrupted. At the point at which the TLX questionnaires were distributed, the subjects would potentially have been experiencing relief at completing a substantial task and may perhaps already be under pressure from additional tasks and workloads outside the scope of this study. Given the extended nature of the evaluation task - it spanned approximately seven weeks - the workload results obtained are likely to demonstrate a tail-end bias - that is, they will reflect the latter sub-tasks of the overall evaluation exercise. This may have been further exacerbated by the fact that the subjects are unlikely to have distributed the completion of the task evenly across the study duration. However, these factors will similarly have affected both groups and are therefore unlikely to be reflected in the differences observed and analysed between the two groups.

5.4.2 Impact of the Potential Execution Limitations on the Data Analysis

It is to be expected that limitations on, or flaws in, the process of data-collection will have an impact on the reliability of the data analysis and therefore the claims that can be made in light of a study. This section outlines the measures that were taken to accommodate the identified flaws in this study.

5.4.2.1 Impact on Recommendation Reports and their Analysis

As mentioned, to reduce researcher bias during the analysis of terminological use, the procedure was documented and repeated by an independent assessor for a random selection of reports, and the results verified.

Given the potential problems discussed above, advice was obtained from an expert statistician to validate the use of statistical measures over this - and the other data - in this study (Tweedie, 2000). On the advice of the statistician, statistical analysis techniques were restricted to those which made no assumption or claims about the distribution of the subjects and their associated results (Lindley and Scott, 1984, Woods et al., 1986). All appropriate statistical analysis techniques assume randomness in the sample population and so, in this sense, consideration of the statistical results presented in this chapter should acknowledge that their accuracy may be compromised as a result of the lack of randomness in the sample population. However, by applying due caution with respect to the matter of sample population distribution, the statistical claims made can be considered to be as accurate as possible[66].

The potential influence of the Hawthorne Effect may have actually reduced the significance of the difference between the use of specific terminology between the two subject groups - subjects in the Hix group may have used specific terminology on the basis that they anticipated it to be that which the researcher was looking for. They may therefore have artificially increased the tallies for some terms within the Hix group and in so doing affected the difference - and significance thereof - between the observed use of the terms across the two groups. It is therefore possible that statistically, SUIT was more effective at directing evaluators' attention to issues of context of use than is reflected in the results.

During discussion of quoted subject comments in section 5.3.4, reference was made to the pervasive issue of the Hawthorne Effect and it was stressed that the comments should be considered with due scepticism and caution.

5.4.2.2 Impact on Log Sheets and their Analysis

In section 5.3.2, when discussing the results obtained from the log sheets, the unreliability of the results was duly acknowledged. This unreliability (and scarcity of data) is likely to be a consequence of the motivation behind the exercise - the students were probably wary of publicly declaring the amount of time they had spent on an assessment. Furthermore, there was no means by which to verify or validate the subjects' record of time. On the advice of the statistician, analysis of this data was restricted to comparing average values for both groups. No statistical claims were attributed to the observations; instead the results were treated as illustrative rather than quantitative data.

[66] Due caution should nevertheless be exercised if trying to accurately apply the claims beyond the scope of the sample population.

5.4.2.3 Impact on Workload Results and their Analysis

Little could be done after the fact to reduce the tail-end bias which has potentially affected the subjects' perception of their overall experience of workload. Similarly, it was not possible to eliminate the extraneous factors that are likely to have influenced this perception. The results obtained from the workload questionnaires were therefore taken at face value and analysed as such. Statistical analysis of the workload results was generally restricted to techniques that make no assumptions about the distribution of the subjects and their associated results. For example, the identified flaws in the sample population meant that it was not possible to apply Factor Analysis to the weighted ratings, and so analysis had to be restricted to simply measuring the coefficient of correlation. Where alternative techniques (that is, techniques which did make some assumption about subject distribution) were used, the decision was explained.

5.5 Conclusions

Despite the potential limitations and associated caveats that must be considered in conjunction with the results presented in this chapter, there are a number of important points from which positive conclusions can be drawn.

The primary aim of this study was to show that SUIT is effective in terms of directing evaluator attention to issues of context of use. It also aimed to assess the viability and practicability of SUIT as a paper-based UIDT selection facility. The effectiveness of SUIT was principally investigated by means of analysis of the terminological references made in the submitted recommendation reports. SUIT was shown to significantly increase the number of references made to the majority of the terms analysed. Given that these terms were selected on the basis that they reflect the project-specific considerations and dimensions of measurement promoted by SUIT, it can be concluded that (at least within the scope of the sample population) SUIT was effective at directing evaluator attention to issues of context of use.

Consider now the secondary issue of the viability or practicability of SUIT as a paper-based UIDT selection facility. Fundamentally, only one person (other than the researcher) would need to be shown to complete a UIDT selection using SUIT to demonstrate that SUIT is viable. Given that fourteen people were shown to use SUIT to complete a UIDT evaluation, it can be concluded that SUIT is indeed viable as a paper-based UIDT selection facility. Although this claim cannot be generalised across all possible users, it does not detract from the fundamental observation that *SUIT has been shown to be viable as a*

paper-based evaluation facility for UIDTs. Hence, both of the fundamental aims of the study have been addressed and have led to positive conclusions.

Despite significantly increasing the mental effort required to perform an evaluation, SUIT did not significantly increase the time taken to complete an evaluation. Furthermore, it was shown that using SUIT did not significantly increase the perception of workload. Just in terms of dedication and general effort, SUIT and the Hix *et al* approach are similar.

SUIT was shown to place greatest demands on time during the preparatory and comparative evaluation activities, and to reduce the amount of time required to collect data from the UIDTs themselves. It was the manipulation and management of the SUIT frameworks during these tasks that generated most complaints from the SUIT subjects (section 5.3.4). From these observations, it can be concluded that there is scope to improve the overall time and effort requirements when using SUIT if these elements of a SUIT evaluation were to be automated. That is, the results of this study have highlighted the need for software support to assist, and thereby speed up, the process of tailoring frameworks, creating the ideal tool profile, managing the data once collected, and providing a means by which the data could more easily be compared once collected. Given the cognitively demanding nature of the SUIT approach, it is not anticipated that such support will lessen the mental effort inherent in SUIT evaluations. The nature of SUIT is such that it already assists in the reduction of the amount of time required to collect data from the UIDTs, and so the need for assistance for this aspect is less immediate. That said, chapter 8 will introduce and discuss some ideas for further assisting this aspect of UIDT selection.

In summary, therefore, SUIT has been shown to be effective both in terms of directing evaluator attention to issues of context of use, and to be a viable means of UIDT selection in its paper-based form. Further results and observations from the study have identified the scope and need for software support to assist in the preparatory, data management, and data comparison elements of a SUIT evaluation.

CHAPTER 6 : THE SUIT DATA VISUALISATION ENVIRONMENT

6.1 Introduction

The preceding empirical study concluded that SUIT is effective at directing evaluators' attention to issues of context of use and, in its paper-based format, is viable and practicable. That said, the study also illustrated the need for software support to assist the use of SUIT and thereby increase its practicability.

Motivated by the SUIT framework, methodology, and the observations of both the industrial survey (chapter 3) and the empirical study (chapter 5), two constituent components of software support have been established to assist in the performance of SUIT evaluations: automated means to tailor, use, and manage the SUIT framework tables; and a visualisation environment designed to facilitate the comparison and analysis of evaluation data.

For reasons that are discussed in later sections, this chapter primarily focuses on the SUIT visualisation environment. It begins by introducing the environment before outlining the requirements that guided its development and the development of automated support for the SUIT frameworks, and discussing the means by which these requirements have been met. Following a discussion of the visualisation environment design in terms of the data representation and associated means of manipulation, the chapter concludes with a summary of the automated support for performing SUIT evaluations.

6.2 Overview of the SUIT Data Visualisation Environment

The SUIT visualisation environment is, at present, a research environment that concentrates on the *visualisation* of SUIT-oriented data rather than the mechanisms of storing and entering that data (see section 6.3.1). The environment extracts the structure and data for visualisation from external databases (see section 6.3.1) and represents it using a novel combination of location and colour-coding to allow analysis and interrogation of the data (see sections 6.3.2.1 and 6.3.2.2). Figure 6.1 shows part of a visualisation created for a generic comparison of two fictitious tools that has been annotated to highlight the main features of the visualisation and environment.

The data visualisation represents all of the information contained within a SUIT framework (see section 6.3.2.1). That is, it represents the active components for any given evaluation (these may or may not constitute the entire framework depending on the genericity of the evaluation) and for each, represents the dimensional measurements proposed by SUIT. Aspects of functionality, interaction mechanisms, and interaction assistance are shown explicitly - as demonstrated in Figure 6.1 - and the remainder of the recorded information is available on demand (see section 6.3.2.2). Each visualisation has a dedicated electronic notepad to allow evaluators to record comparison and analysis comments and observations.

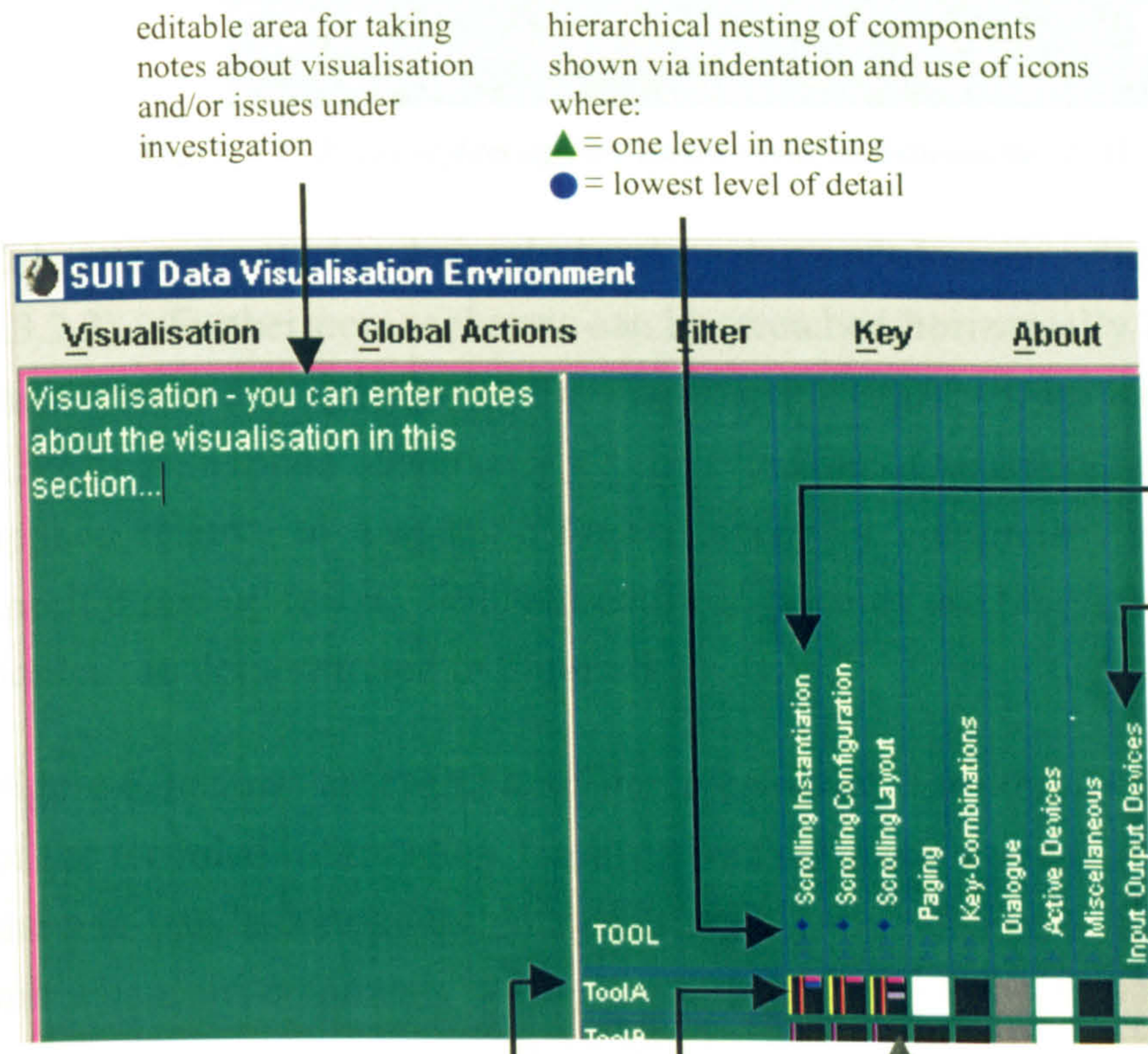


Figure 6.1 - *An example of a section of a SUIT visualisation*

The SUIT data set is spatially organised via a two-dimensional table in which a row represents a UIDT, and each of the SUIT component categories, sub-categories and

individual components is represented via a column (see section 6.3.2.1). Each component sub-division is also represented by a column. Columns can be examined by expanding their parent level column. Category-based data expansion and contraction facilities are applied in relation to a selected column (see section 6.3.2.2).

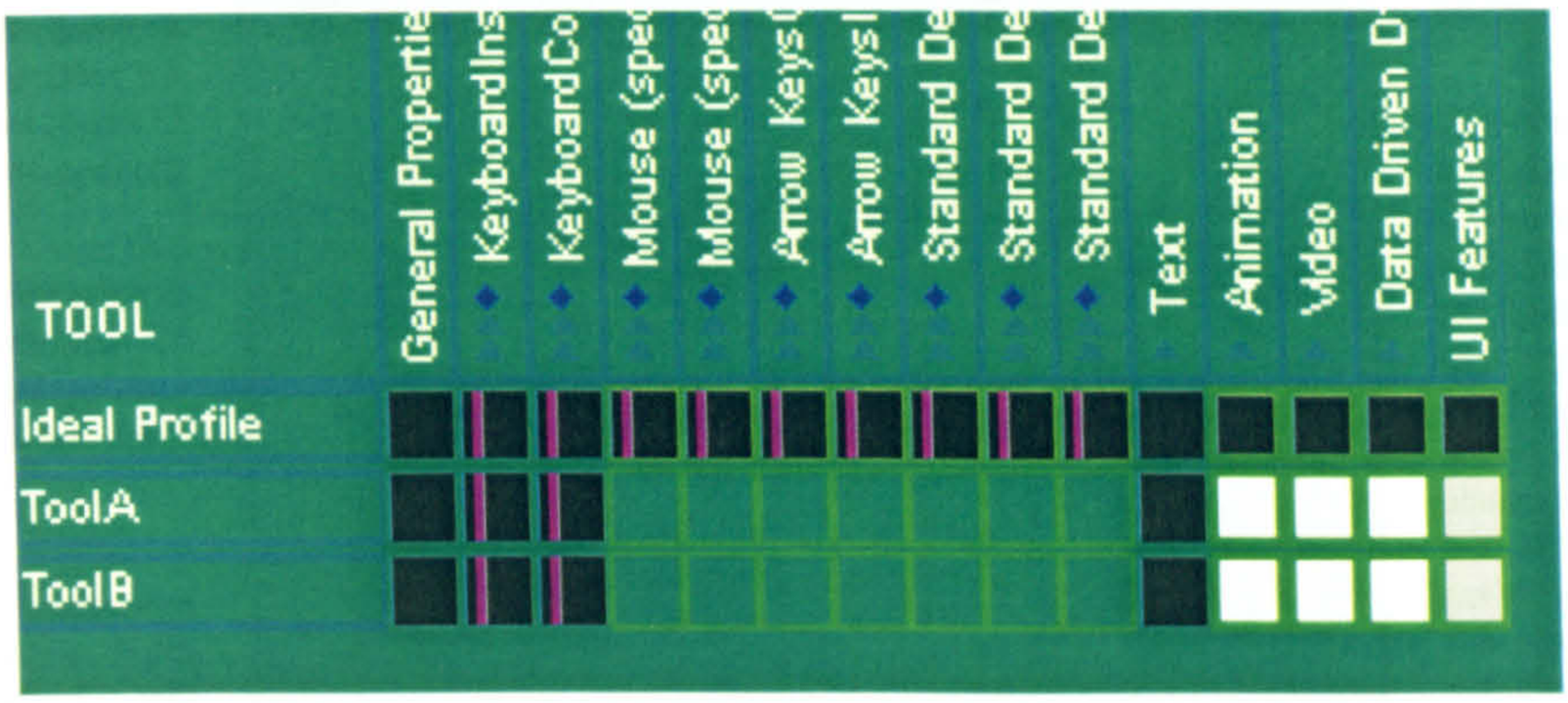


Figure 6.2 - *Result of filtering functionality matches between the UIDTs and the ideal tool profile*

Columns are re-ordered simply by dragging-and-dropping them into position (see section 6.3.2.2). Furthermore, columns can be stretched horizontally in order to enlarge or reduce their allocated proportion of the visualisation area. Data analysis facilities (see section 6.3.2.2) are applied universally via the selection of an option on a permanent menu and are applied relative to a specific (sub)category or component via an option on a column-specific pop-up menu. Filtered data components are highlighted via a bright surrounding border - as demonstrated in Figure 6.2.

Figure 6.3 shows an example of the information card that is used to (textually) present all of the recorded information for any given component relative to any UIDT, and especially the non-standard/auxiliary data such as cognitive demands, quality of feedback, and other miscellaneous comments.

Finally, there is a floating, re-locatable visualisation key to assist the evaluator in determining the colour-coding used throughout the environment.

UIDT evaluations are not a regular activity within most software development organisations (see chapter 3), and so it is unlikely that the visualisation environment will be subject to continuous use. It is therefore hypothesised that there will be an element of re-education in the use of the tool every time an evaluator returns to the environment to complete a new UIDT evaluation. It is not feasible to advocate that new users must undertake formal tuition of the environment, and so it is anticipated that hands-on experience will contribute (and ultimately equate) to learning the environment - 'learning by use'. A scenario-based user manual has therefore been developed to accompany the SUIT visualisation environment. Assessment of the effectiveness of the user manual is considered to be beyond the scope of this thesis, and so is discussed as future work in

chapter 8. A copy of the user manual is included as Appendix E and should be referred to for a more detailed overview of the environment.

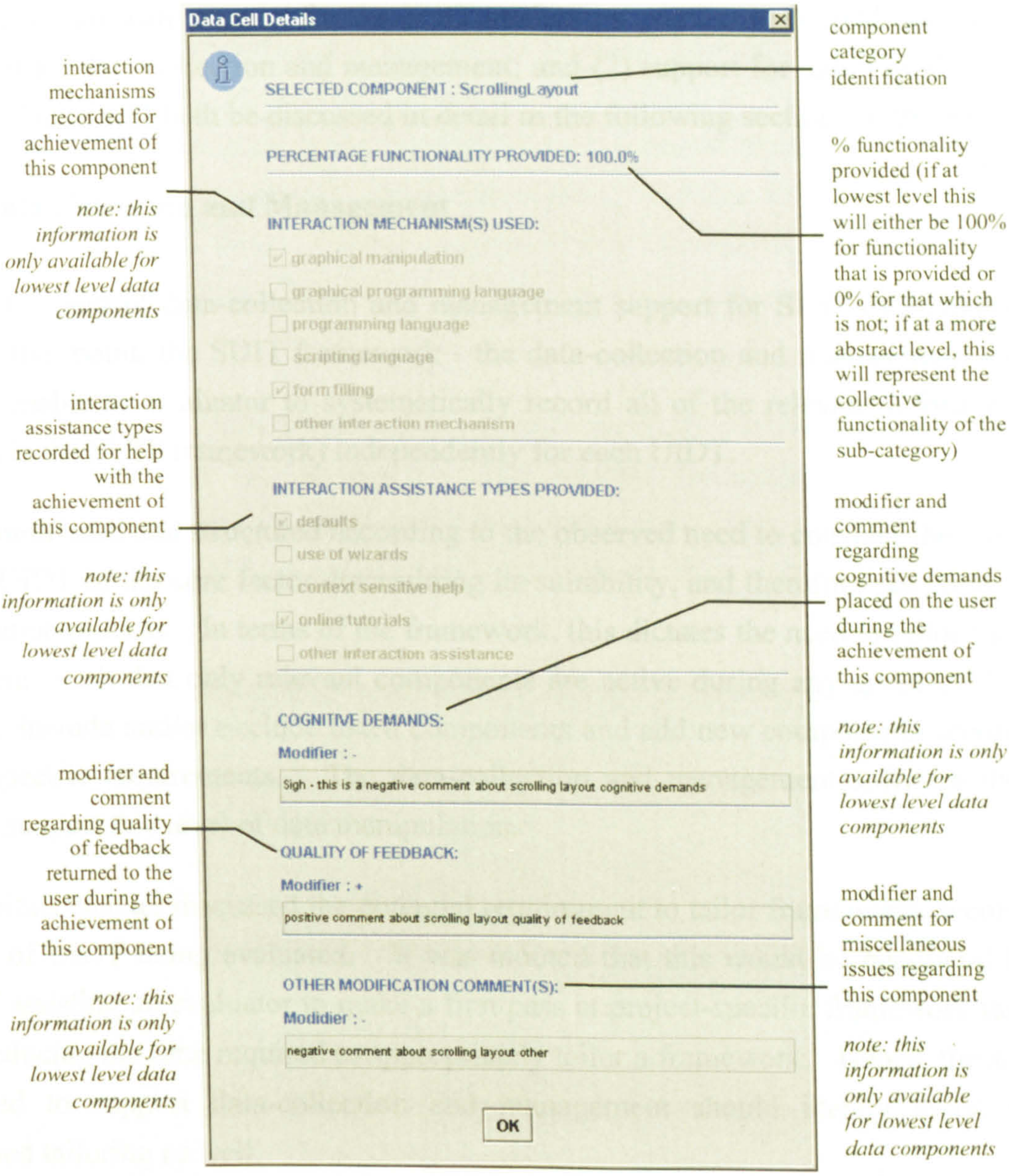


Figure 6.3 - *Annotated example of a data sheet*

The following sections of this chapter discuss the design and development considerations, decisions, and activities that led to the current prototype of the SUIT visualisation environment. This introduction to the environment precedes the forthcoming discussion in order to provide a reference point for the issues covered.

6.3 Software Requirements and Design

The empirical study illustrates the need for automated means by which to both store and view SUIT-based UIDT evaluation data. SUIT evaluation activities can be categorised as:

1. creating, tailoring, using, and managing instances of the SUIT framework;
2. visualising and analysing the collected data.

On this basis, software support for the SUIT evaluation process is divided into two parts: (1) support for data-collection and management; and (2) support for data visualisation and analysis. These will both be discussed in detail in the following sections of this chapter.

6.3.1 Data Collection and Management

The development of data-collection and management support for SUIT evaluations takes as its starting point, the SUIT framework - the data-collection and management software needs to enable an evaluator to systematically record all of the relevant information (as specified in the SUIT framework) independently for each UIDT.

SUIT is motivated and structured according to the observed need to consider the context of use of a UIDT as the core factor determining its suitability, and therefore selection, during an evaluation process. In terms of the framework, this dictates the need to tailor the list of components such that only relevant components are active during any given evaluation - that is, to include and/or exclude listed components and add new components according to project-specific requirements. The data-collection and management software therefore needs to support this level of data manipulation.

The previous chapter discussed the potential requirement to tailor frameworks according to the type of UIDT being evaluated. It was mooted that this would be beneficial both in terms of assisting an evaluator to make a first pass at project-specific framework tailoring, and in reducing the time required to appropriately tailor a framework. Hence, the software developed to support data-collection and management should ideally assist in this specialised tailoring as well.

Each completed SUIT framework has the potential to be at least thirty nine pages long. By handling electronic versions of the frameworks and their associated data, the supporting software should substantially ease framework and data management.

To summarise, the data-collection and management software needs to provide the means to record all of the information detailed in the SUIT framework and provide an evaluator with the ability to:

- create a new instance of a data-collection record for each UIDT;
- tailor a data-collection record on the basis of UIDT type (if required);
- tailor a data-collection record to activate/deactivate existing and add new components;

- systematically record appropriate data according to the tailored data-collection record;
- and to maintain and manage completed data-collection records.

In essence, this list of requirements mirrors the facilities provided within most database packages (for example, Microsoft® Access97). In light of the significant research challenge involved in developing the visualisation environment alone (see section 6.3.2) it was considered prudent to simply exploit pre-existing facilities for basic data-collection and management at this time. On the assumption that Microsoft® Access97 is a widely available database package, it is used to support basic data-collection, storage, and management for SUIT.

A database schema has been developed that mirrors the SUIT framework tables. A form-based user interface has been designed for the database that furthers that parallel and a blank template version of this database has been generated. To create a new instance of, what will now be referred to as an *'electronic framework'*, an evaluator need only copy the blank template.

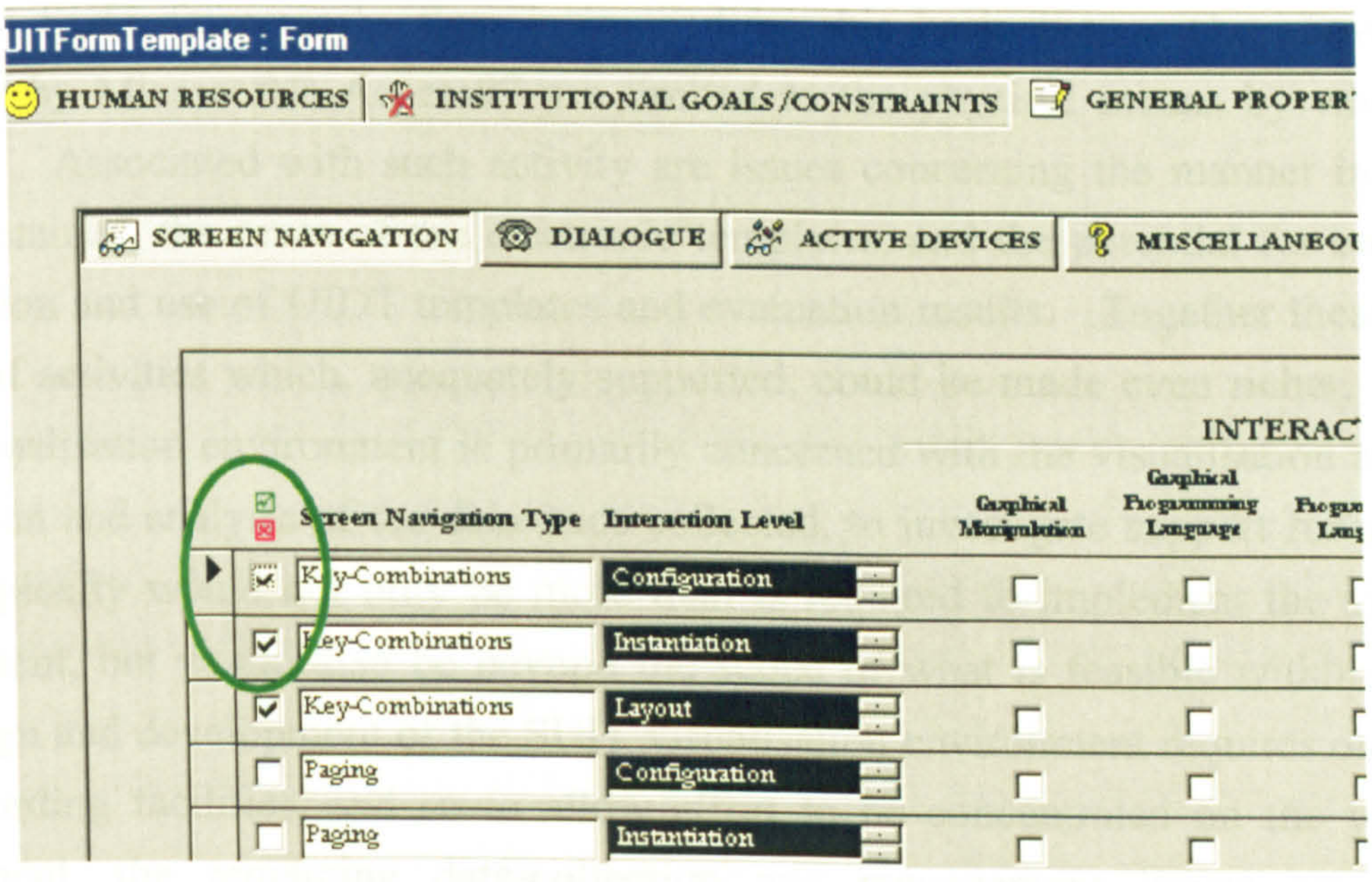


Figure 6.4 - *A section of an electronic framework page highlighting component activation/deactivation*

Figure 6.4 shows a section of one of the electronic framework pages. Each listed component is preceded by a small checkbox that represents the evaluation-specific activation/deactivation of the component. By default, all components are active - to deactivate any listed component, the evaluator need only deselect its corresponding checkbox. By default, Microsoft® Access97 database tables include the facility to add a new component and so additional components can be easily added to those which are listed in the SUIT framework.

The electronic framework is structured to prevent invalid data entry, and so encourages and enables systematic data entry for active framework components. To maintain and manage copies of the electronic framework the evaluator need only apply standard file management techniques.

By exploiting the standard Microsoft® Access97 facilities it is possible to implement the base identified data-collection and management requirements. However, the requirement for a facility to guide electronic framework tailoring according to UIDT type cannot be met solely by the functionality available in Microsoft® Access97. It will therefore be necessary to design and develop additional software that guides the manipulation of standard electronic frameworks. The appropriate means by which to affect this guidance requires further investigation and so it is considered to be beyond the scope of this thesis. An interesting element of future work, it is discussed further in chapter 8.

With the exception of guided means to tailor electronic frameworks according to UIDT type, the basic identified data-collection and management requirements are therefore provided. There is, however, more to supporting the generation, management, and use of completed SUIT frameworks than is implied by this basic list. The above facilities supported by Microsoft® Access97 are limited to the physical means by which data is recorded. Associated with such activity are issues concerning the manner in which the data is obtained, the reuse of the electronic templates, and the potential for collaborative construction and use of UIDT templates and evaluation results. Together these suggest a rich set of activities which, adequately supported, could be made even richer. Since the SUIT visualisation environment is primarily concerned with the visualisation and assisted comparison and analysis of the data once collected, to investigate support for activities of this complexity would not only be more than is required to implement the visualisation environment, but would also be beyond the scope of what is feasible within this thesis. The design and development of the SUIT visualisation environment requires only the base data recording facilities, and so to allow effort to be concentrated on the visualisation environment, the remaining data-collection and management issues are considered important aspects of further work and are discussed in detail in chapter 8.

6.3.2 Data Visualisation and Analysis

When designing the software support for data visualisation and analysis, there were two constituent components to consider: (1) the visual representation of the data; and (2) the facilities required to manipulate that data representation.

No tool currently exists that is designed specifically to support non-textual comparison and analysis of UIDT (or indeed CASE tool) data. As such, other than that gleaned from the

empirical study of the paper-based version of SUIT (see chapter 5) there is, at present, no corpus of knowledge *specifically* about the manner in which evaluators compare and analyse UIDT evaluation data given the support of such a tool. There is, however, generic information available regarding the manner in which people compare and analyse data to which the design of the SUIT visualisation environment turned for guidance in order to address the above points.

There is evidence that users' performance and subjective satisfaction is higher when they can view and control their search (Shneiderman, 1998). On this basis, Shneiderman suggests a four-phase framework for search strategies that places the user in control:

1. *formulation* - expressing the search in terms of appropriate parameters etc.;
2. *initiation of action* - either implicitly or explicitly launching the search;
3. *review of results* - observing and manipulating the results;
4. *refinement* - formulating the next step (Shneiderman, 1998).

He suggests that by applying this four-phase framework, designers can make a search process more visible, comprehensible, and controllable by users (Shneiderman, 1998).

Shneiderman proposes a visual information seeking 'mantra' that describes the order of scale at which users like to explore a data set (Shneiderman, 1998):

Overview first, zoom and filter, then details on demand

This rule relates to a series of tasks - overview, zoom, filter, and details on demand - to which he later added relate, history and extract tasks (Shneiderman, 1998). The overview task allows the user to gain an overview of the entire collection. The zoom task allows users to focus on particular areas of interest (Shneiderman claims the most satisfying way to zoom is to point to a location and issue a zooming command). Filter tasks allow users to filter out uninteresting items and quickly focus on those items that are of interest. Details on demand enables a user to obtain specific details about selected items as required. The relate task lets users view the relationships among items. The history task allows users to refine and retrace their actions - especially important given that a single user action rarely produces the desired result. Finally the extract task allows the user to extract items of interest once found. Shneiderman claims that very few prototypes support either the history or extract tasks (Shneiderman, 1998). There are examples of visualisation environments that reflect Shneiderman's principles - for example, Table Lens and FOCUS (Rao and Card, 1994, Spenke et al., 1996) - and closely resemble the facilities required for SUIT. Although these do not adequately support the SUIT data set, they provide inspiration and guidance for the development of the SUIT-specific UIDT

evaluation data visualisation environment. These examples are discussed in greater detail in the following sections.

Applying Shneiderman's principles, drawing on appropriate existing generic examples of visualisation environments, and observing the findings of the empirical study (see chapter 5) it will still only be possible to approach the optimal solution for SUIT via a process of iterative design and user testing. The lack of *specific* knowledge to inform its development and the likely complexity of the visualisation environment is potentially such that the time scale necessary to affect this process will far exceed that available to this research. It was therefore decided to develop the visualisation environment as an initial - albeit fairly comprehensive - prototype. Following evaluation, it can be used to inform and guide further development (see chapters 7 and 8). The design of the visual data representation and data manipulation facilities, guided by the above, are discussed below.

6.3.2.1 Visual Representation of the Data

To determine how best to visually represent SUIT data it was important to first consider the manner in which the data will be used. Although little is yet known about the specific comparison and analysis tasks evaluators are likely to perform using the environment, guided by the generic information that is available, it is possible to identify a core set of potential abstract tasks on the basis of both the methodology and framework structure.

The anticipated goal when using the visualisation environment is to identify aspects of the UIDTs upon which to base a selection decision, and so it is assumed that evaluators will want (or indeed have) to look at matches and mis-matches between the tools. Evaluators may want to identify matches and mis-matches according to low-level detail - for example, the interaction mechanisms used. Alternatively, they may want to identify the matches and mis-matches at an abstract level and then focus on specific highlighted elements, or groups of elements, for closer consideration. Adding data abstractions to the already complex collection of components and dimensions of measurement in a SUIT framework will result in a potentially extensive data set for each UIDT. The extent of the visualised data set will therefore increase substantially for each additional UIDT being compared. Given meaning and trade-offs that may be derived from the inter-relationships between the contributory measurement criteria, it is anticipated that evaluators will want to consider the criteria simultaneously. Until more information is available concerning the comparison and analysis strategies adopted by evaluators using the visualisation environment (see chapter 7), it would not be prudent to design separate visual representations of the data on the basis of individual criteria. The visual representation should therefore accommodate the simultaneous presentation of data for all measurement criteria per UIDT component. Furthermore, it should facilitate ease of identification of matches and mis-matches therein.

By presenting information visually and allowing dynamic user control over the organisation of that data, it is possible to enable users to traverse large information spaces with increased comprehension and reduced anxiety (Ahlberg and Shneiderman, 1994). Such is the enormous capacity for human visual information processing, within a few tenths of a second humans can recognise features in mega-pixel displays and can identify anomalies (Ahlberg and Shneiderman, 1994). Given the use, size and complexity of the data set to be visualised, a graphical representation was therefore the most appropriate. To design a graphical representation of the SUIT data, the following needed consideration: (1) the organisation of the data; and (2) the visual representation of each data component. The implications and resolution of each of these aspects with respect to the visual representation used in SUIT are discussed below.

Spatial Organisation of the Data

There were two principal concerns regarding the organisation of the data: the collective layout of the data components, and the manner in which an evaluator can manipulate that organisation. First consider the layout. A number of different spatial layouts were potential candidates for the SUIT visualisation environment - including graphs, parallel-co-ordinates, and scatterplots (Dawkes et al., 1996, Shneiderman, 1998). Each was considered during initial attempts at designing the environment but they were found to be unsympathetic to the particular complexities of the SUIT data and/or the manner in which it was likely to be manipulated. Two-dimensional point-based layouts (for example, graphs and parallel-co-ordinates) were found to be ill-suited to representing multiple complex SUIT data items at each of their points, where each point also had multiple relationships to other points in the representation. In particular, parallel co-ordinates became so overly complex that they could no longer be interpreted. Histograms and scatterplots, both 2-dimensional and *n*-dimensional, were also considered. Two-dimensional versions were inappropriate in terms of the complexity of data that could effectively be visualised at each point. The *n*-dimensional versions required so many dimensions to systematically visualise all the data that they not only became too complex, but they did not permit easy simultaneous comparison of all the data attributes and therefore pattern identification. The alternative, on the basis of the SUIT framework and the complexity of the data, was a table (Bertin, 1981, Spenke et al., 1996, Rao and Card, 1994, Shneiderman, 1998) and so this layout was further investigated.

Much has been written about the advantages of using tables as the basis for data organisation (Bertin, 1981, Gilmore, 1991, Spenke et al., 1996, Rao and Card, 1994), and their power and flexibility as a visual means of data layout or organisation is widely recognised (Bertin, 1981, Spenke et al., 1996, Rao and Card, 1994, Microsoft, 1997). In

particular, the regularity of the content of a table means the interrelated information along its rows and columns can be interpreted as a coherent set (Rao and Card, 1994), and their structure permits a clear and concise simultaneous representation of multiple attributes of data objects. Comparing tables to graphs, Joachim *et al* observed that tables proved best for user recollection of specific values, led to faster response times, and were shown to be at least as good, if not better, than graphs in terms of accuracy (Joachim et al., 1997). Shipman *et al* discovered that the most commonly used spatial structure during human problem solving was lists - both vertically and horizontally - and that such structures were primarily used to represent categorisation of features (Shipman et al., 1995). Lists are clearly embodied within the spatial layout of tables. To determine the benefit of using tables as opposed to standard lists for data comparison, Gilmore compared the use of each structure (Gilmore, 1991). For the same data set displayed in both list and table format, he found that for complex multi-dimensional data comparisons tables demonstrated greater accessibility and therefore speed of task completion. He also found that relationships between the data structures were more easily identified in the table structure than in the lists (Gilmore, 1991).

The above arguments, the characteristics of the SUIT data set - the categorisation of the data components and the complexity of attributes against which the components are measured - and the need to compare the data across UIDTs, establish the suitability of a table as the spatial layout for the visualisation.

Aside from standard spreadsheet packages, a number of tools have been developed that exploit the power of tables - in particular, Table Lens and FOCUS (Rao and Card, 1994, Spenke et al., 1996). Based on Shneiderman's principles, Table Lens presents a powerful style of direct manipulation exploratory data analysis. It combines symbolic and graphical representations into a single manipulable focus+context (or fisheye) display and provides a small set of interactive operations for data investigation (for example, sorting). Focus+context enables the user to interact with large information structures by dynamically distorting the spatial layout and thus varying the interest levels of the parts. This technique enables a user to view the entire data set at once whilst simultaneously focusing on specific items of interest. In so doing, it allows the user to search for patterns in the wider scope at the same time as investigating interesting details without losing the framing context (Rao and Card, 1994). Table Lens provides manipulation operations to zoom, adjust, and slide the focal area with the result that cells can be moved between three levels of focus - hidden, non-focal, and focal. Furthermore, it provides the facility to 'spotlight' specific cells on the basis of chosen criteria across either a specific column or globally - a feature that parallels the semantics of Magic Lens (Fishkin and Stone, 1995). Based on Table Lens, FOCUS is a generic interactive table for the manipulable display of

common object-attribute data. Unlike Table Lens, FOCUS presents mainly textual data and, via its dynamic query mechanism, enables 'output-is-input' progressive query refinement (Spenke et al., 1996).

Although either of these systems could be used to represent and query SUIT data, neither was considered wholly adequate in terms of their chosen data representations (within the table cells) and/or the way in which the data is manipulated. Since the SUIT tables are not identically structured, they would not have fitted the two-dimensional uniformity of these systems - in particular, FOCUS requires that the underlying data structure is flat. FOCUS concentrates on the manipulation and organisation of the data as opposed to its representation to the extent that it is not clear how matches and mis-matches could easily be identified in the data set. It also makes a number of inappropriate assumptions about the manner in which the users want to work. The genericity of manipulation within these systems means that it would not be possible to support the complex semantics of the SUIT data[67].

That said, much was learned from these systems - especially Table Lens - and was subsequently applied to the development of the SUIT visualisation environment (as discussed below). The main points for consideration in the design of the SUIT visualisation environment include:

- selective re-ordering of the table structure;
- the principles of focus+context (or fisheye) techniques and in particular
  - the facility to search an entire data set while investigating specific areas of interest;
  - the facility to zoom, adjust and slide the focal area;
- the facility to spotlight cells on the basis of a chosen criterion - either at the level of a single column or the entire data set.

In deciding to adopt a tabular layout, consideration had to be given to the manner in which the data - or more specifically, the table - could be manipulated to allow a user to effectively browse and query the contents. This manipulation can be considered at two levels: at an abstract level independent of the semantics of the data and task, and at a concrete domain-specific level. The former is discussed here, and the latter in section 6.3.2.2.

Bertin states that tables promote two types of question: questions introduced by the objects, and questions introduced by the characteristics of the objects. Furthermore, he suggests

[67] There are several differences between these tools and the requirements of SUIT but to prevent this discussion diverting into a visualisation environment evaluation, only a selection of the more significant ones are given here.

that any graphic construction which does not enable a user to find a visual reply to both types of question provides only one part of the information (Bertin, 1981). Bertin also suggests that useful information does not necessarily correspond to the general categories within a table, but to *new* groupings that are defined by the set of relationships constructed by the interplay of the data; as such, a visual construction that does not enable a user to define these groupings does not reach the overall information level of the entire data set. He therefore advocates the ability to re-order columns and rows in a table (Bertin, 1981). Table Lens and FOCUS adopt Bertin's suggestion of selective relocation of their rows and columns. Their exploratory focus+context techniques, when joined with a graphical mapping scheme and small set of manipulation operations, enables users to perform exploratory data analysis in a highly interactive and direct manner (Spenke et al., 1996, Rao and Card, 1994). The focus+context paradigm reflects the need to compare data at both the general and feature-specific levels within SUIT (see chapter 5), and column relocation continues the promotion of tailorability in the SUIT evaluation system.

The most significant claim made by Rao and Card, with respect to the development of the SUIT visualisation environment, regards the ability of Table Lens to enable the user to explore patterns in large data sets - they claim that most of the patterns that are easily found using Table Lens would be much harder or very unlikely to be detected using a traditional spreadsheet (Rao and Card, 1994). SUIT bases a project-specific UIDT evaluation on a process of matching the facilities of real UIDTs against the profile of a project-specific ideal tool. Furthermore, data matches and mis-matches were identified as core tasks likely to be performed using the environment. Since these activities amount to a process of pattern matching, it seemed essential that the SUIT visualisation environment supports users in terms of being able to match patterns among the tool data sources and ideal tool profile so that they may identify the UIDT best suited to a project. The structure and manipulation facilities in the likes of Table Lens embody the kind of activity it was anticipated that SUIT data comparison would require, making Table Lens a sensible source of guidance when developing a new table-based visualisation. For reasons previously outlined, it would have required considerable alteration to Table Lens to adapt it to the particular needs of SUIT. Informed by the design of Table Lens, FOCUS, and Shneiderman's design principles, a table-based data visualisation for SUIT was therefore independently investigated and implemented.

To summarise, in order to best cater to the complex requirements of the SUIT data and best support pattern-matching and browsing over that data, the visualisation should:

- adopt a table-based spatial layout to embody the categorisation within the SUIT framework;
- enable the user to overview the entire data set;

- enable the user to alter the focus across different components in the data set;
- enable the user to define groupings within the data - principally by reorganisation of the data categories - within the semantic restrictions imposed by SUIT;
- and enable the user to spotlight or filter data according to selected criteria at the level of a single category or the entire data set.

Centred around a table layout, the SUIT visualisation environment was therefore developed to include the listed manipulation facilities - albeit specific to the needs of SUIT and not identical to the likes of Table Lens.

When examining the use of tables, Gilmore identified some interesting issues of salience (Gilmore, 1991). In particular he noted that, where table rows were used to represent the salient information structure (that is, the principal focus of the comparison task), there was a natural strategy for comparison of the data set. In contrast, he found that when the salient structure does not match the macroscopic focus (the goal focus) subjects were at a loss as to how to perform the task unless they received instruction regarding the mapping between the macroscopic and microscopic (action) foci. Since SUIT will not be provided with tuition (other than the user manual) it is important to ensure that a natural comparison strategy is inherent in the visualisation. Therefore, according to Gilmore's findings, the rows in the SUIT table should represent the salient information structures - the tools - and the columns should represent the microscopic foci - the component and measurement information.

It was anticipated that the number of evaluated components would always out-number (by as much as one thousand times) the total number of UIDTs being evaluated, and as such, there would be greater demand for component relocation as opposed to tool relocation. Investigation of implementational issues regarding the table layout highlighted that column relocation was better supported than row relocation. Together, these factors determined that the SUIT visualisation environment should use rows to represent the salient tool structure and columns to represent the components, each of which should be labelled according to the associated label within the SUIT framework. Thus, a single cell within the table represents the data for either a single component or a component hierarchy (see later) for a single UIDT.

Visual Representation of the Data Components

Having determined the spatial layout of the visualisation, attention was focussed on the visual representation of the table data. Each component sub-division (instantiation, configuration and layout) requires representation of six interaction mechanisms, five

interaction assistance types, and a modifier and comment for each of: cognitive demands; quality of feedback; and miscellaneous comments. A means of condensing and collectively representing the data in each cell had therefore to be developed.

Since graphical representations demonstrate a natural economy in terms of showing values (Rao and Card, 1994), occupy a higher bandwidth of information presentation than any other media reaching any of the other senses (Shneiderman, 1998), and have been shown to be effective in terms of enabling the identification of patterns and features in table data (Rao and Card, 1994, Shneiderman, 1998), a graphical representation was adopted for the data in each table cell. The amount of data that is recorded for each cell would inevitably lead to clutter and therefore lack of clarity if it was all represented graphically. The ancillary data (see chapter 4) was therefore excluded from the graphical representation and was instead included on an information card for each cell. Available on demand, these information cards exploit the principle of 'details-on-demand' (Ahlberg and Shneiderman, 1994. The remaining primary data - that is interaction mechanisms and interaction assistance - then became the focus of attention when designing the visual representation of cell data.

Initial designs of the graphical data representation tried to exploit the work done by Bertin in the use of circles to represent data components, sized according to some associated attribute (Bertin, 1981). Due to the complexity of the data that each circle was required to represent in SUIT, it proved to be an inappropriate representation - most significantly because it was very difficult to pattern match data across the table. The Bertin-inspired circles where therefore abandoned. Experience with the circles highlighted the need to focus on the development of a visual representation of the data that would most effectively facilitate pattern matching across the table cells. Most importantly, since SUIT measures learnability according to the match between ideal profile interaction mechanisms and UIDT interaction mechanisms, the graphical representation had to be developed to assist interaction mechanism pattern matching as far as possible. Chambers *et al* claim that the human eye is better able to perceive simple patterns than complex ones and, in particular, is most able to determine straight line configurations (Chambers et al., 1983). Based on this, and the experience with the circles, it was decided to base the visual rendering on linear representations of each of the interaction mechanisms. Since each listed mechanism needed to be independently identifiable, a colour-coding was mapped onto each. Further, on the basis of Bertin's observation that varying the position of components in a nominal data set imposes an order that is immediately perceptible, the colour-coded lines were also given a positional or proximity code (relative to the bounds of the data cell). The resulting graphical representation of the interaction mechanisms for each data component was a fixed-position, colour-coded, vertical bar arrangement such that vertical linear patterns

could be determined for each component across the rows (that is, across the UIDTs) in the table[68]. Interaction assistance data was similarly represented but was given a horizontal orientation to differentiate interaction assistance swatches from those representing interaction mechanisms in each cell. On the basis of the SUIT methodology, it was anticipated that interaction mechanisms would be the primary or initial focus of user attention. Therefore, although interaction assistance data is explicitly graphically represented within each appropriate table cell, to avoid clutter and thus simplify the initial visualisation, it is only made visible on demand.

Discussion of the anticipated core activities identified the potential requirement for evaluators to consider data abstractly - that is, at the level of component categories. It was therefore decided that each level in the SUIT hierarchy would be represented in the table via a dedicated column, and that the principles of focus+context would be used to navigate between these levels. Barring the lowest level of detail which corresponds to component sub-division, applying focus to a given column or level replaces it with the columns representing the components at the level immediately below it in the SUIT hierarchy. Thus, in contrast to using a distortion mechanism as in Table Lens and FOCUS, SUIT uses a parent-child replacement mechanism to apply or remove focus from any given component. Each level of the hierarchy therefore represents an abstraction over the data in the sub-tree of which it is root - for the purposes of the initial prototype the abstraction was restricted to functionality. Therefore, each abstract cell represents the percentage of required functionality - with respect to the component hierarchy of which the cell is root - that is actually provided by the corresponding UIDT. According to Tufte, varying shades of grey have a natural visual hierarchy and so represent varying quantities better than colour (Tufte, 1983). Functionality percentages from 0% to 100% were therefore mapped onto greyscale values which were then used to represent the abstraction over functionality in the SUIT visualisation environment - 0% is represented by white regions, and 100% by black regions. When focus is applied to a white (0%) region such that it is replaced by the most detailed level components - that is, the level at which interaction mechanisms are shown - the zero data is represented by a 'blank' cell which simply shows the background colour of the table.

The remaining aspect of the visual representation that required design consideration was the means by which to 'spotlight' or highlight cells on the basis of a user selected criteria. This was implemented via colour highlighting of the cell boundaries.

[68] This is similar to the coloured, positioned, swatch representations used successfully within Table Lens to represent category values.

As the most effective visual display property (Ahlberg and Shneiderman, 1994, Rao and Card, 1994, Bertin, 1981, Shneiderman, 1998), colour was used extensively in the SUIT visualisation environment. However, designing the characteristics of the graphical data representation highlighted the large number of colours that were required. In consideration of the proportion of the potential user population who are colour-blind, colour choice was guided and informed by Kelly's set of twenty-two colours of maximum contrast (Kelly, 1966).

6.3.2.2 Facilities for Manipulation or Analysis of the Data

Having decided upon the abstract data manipulation and the data representation, it was important to consider the way in which the evaluator might want to analyse that data, and therefore, concretely manipulate the representation. Given the improbability that the first visualisation environment prototype would capture all necessary data analysis facilities (see section 6.3.2), the approach adopted was to design a set of data manipulation facilities informed by the abstract table manipulation principles, the results of the empirical study (see chapter 5), and the specific concepts of SUIT - that is, data manipulation facilities that enhance the SUIT perspective on UIDT evaluation.

The design of the data representation concluded that data central to the SUIT methodology - that is, active components measured according to their interaction mechanisms and interaction assistance - would be explicitly graphically represented within each low-level table cell. This immediately facilitates the visual investigation of functionality and, given the visual representation of a project-specific ideal tool profile, the assessment of *prima facie* learnability promoted by SUIT (see chapter 4).

The results of the empirical study illustrate the following requirements for data manipulation and analysis:

1. the ability to automatically generate a project-specific ideal tool profile;
2. the ability to compare a project-specific ideal tool profile with the data collected for the various UIDTs under evaluation;
3. the ability to examine and compare the data at the level of individual active components;
4. and the ability to view the data for the non-standard aspects of UIDT evaluation - for example, the quality of feedback.

On the basis of the generic principles discussed in the preceding section, and the requirements listed above, the following data manipulation and analysis facilities were included in the SUIT visualisation environment.

- *SUIT automatically generates the project-specific ideal tool profile*

Context of use is embodied in the project-specific ideal tool profile. For the first visualisation environment prototype, it was decided to completely automate ideal tool profile generation but, for simplicity, to restrict it to the human factors information recorded in a tailored template. Therefore, unlike the paper-based frameworks in which an evaluator manually correlates suitable interaction mechanisms derived from the human factors *and* task characteristics with the tailored list of functional requirements, the initial prototype restricts this correlation to just mechanisms on the basis of the human resources.

Using the visualisation environment the evaluator need only record the human resource information in the appropriate table in one electronic framework which is allocated as the template for the visualisation. The visualisation environment tallies the interaction mechanisms listed in this table and calculates the most appropriate interaction mechanism(s) on the basis of the mechanism(s) with the highest tally. It then generates an ideal tool profile in which the interaction mechanism(s) are correlated with the functional requirements. Thus, albeit with some restrictions, the requirement identified in the empirical study to automate the process of ideal tool profile generation has been met.

- *SUIT assists pattern matching between the tools and the ideal tool profile*

For project-specific evaluations the ideal tool profile is visualised as if it were another UIDT so that it can be compared to the real tools represented. To assist this comparison - or pattern matching - the visualisation environment includes facilities to highlight matches and mis-matches between the project-specific ideal tool profile and the UIDTs according to the provision of functionality and to the use of interaction mechanisms. Thus, not only can the degree to which the UIDTs provide the required functionality be compared, but also the learnability of each UIDT can be assessed.

Since the project-specific ideal tool profile only includes information about the required functionality correlated with interaction mechanisms, there was no further requirement for data manipulation or analysis facilities that were particular to project-specific UIDT evaluation. The remainder of the implemented data manipulation and analysis facilities are therefore applicable for both generic and project-specific evaluations.

- *SUIT presents information at abstract and component-specific levels*

SUIT data can be visualised abstractly on a category by category basis, and at the level of individual component sub-division in terms of the interaction mechanisms used and assistance provided. Assuming that an evaluator would not necessarily want to universally view all data at the same level of abstraction or detail (see section 6.3.2), the

visualisation environment has been developed to allow an evaluator to expand or contract the level of detail both universally, or relative to a single component or category of components (at any level in the categorisation hierarchy). The evaluator therefore has absolute control over exposure to detail, and the principles of focus+context are observed.

- *SUIT supports the re-ordering of data components*

The visualisation environment supports the relocation of components and categories of components. It does, however, place restrictions on the movement of certain columns to observe the internal semantics of the component hierarchy. Like Table Lens, SUIT provides the facility to hide any given component or category of components. An evaluator therefore has complete control over the left-to-right spatial ordering of the entire active data set. Thus, the facilities not only enable an evaluator to group components for comparative purposes, but they also enable an evaluator to instigate prioritisation among the active data components if necessary.

- *SUIT allows the evaluator to determine visibility of interaction assistance data*

Interaction assistance data is visualised and hidden on demand by the evaluator. Since interaction assistance information is recorded at the level of individual component sub-division, the facility to hide/show interaction assistance data has been implemented so that it can be applied universally - that is, across all components at once - or at the level of a single selected component sub-division.

- *SUIT supports selection of components on the basis of their match/mismatch across all tools*

Assuming the analysis of tool matches and mis-matches to be a core activity, the visualisation environment allows an evaluator to select to view only those components for which all visualised UIDTs match, or conversely mismatch. It was not sensible to suppose the criteria upon which such matches or mis-matches would be based, and so for the purpose of the initial visualisation environment prototype it was decided to facilitate only the most gross match or mismatch - the combination of functionality, interaction mechanisms, and interaction assistance. These selection options have been implemented such that an evaluator can apply them universally or at the level of an entire category of components. The initial prototype will subsequently allow these decisions to be investigated (see chapter 7).

- *SUIT supports selection of components on the basis of 0% or 100% functionality provision across all tools*

On the hypothesis that evaluators would appreciate assistance to identify functionality that was missing from all evaluated tools, the visualisation environment provides the facility to select all components for which no visualised UIDT has registered the functionality. For consistency, and to allow an evaluator to equally eliminate components that are unsupported across all visualised UIDTs, the facility to select only those components for which at least one UIDT provides the required functionality has also been implemented. The evaluator is again given the option to apply these selections both universally or at the level of category or sub-category.

- *SUIT includes the ability to view non-standard data per component*

Information cards can be opened on demand and present all the information for a selected UIDT-specific data component - including the data for cognitive demands, quality of feedback, and other modification issues.

- *SUIT supports 'spotlighting' of data components on basis of selected criteria*

Although an information card can be opened for any given cell in the data visualisation, it was assumed that means of quickly locating cells for which a specific modifier had been recorded for any of the non-standard information would be helpful. A series of highlight (or 'spotlight') options was therefore implemented for cognitive demands, quality of feedback, and other noted modifiers. For each, the evaluator can determine whether the highlight is to apply to comments with a positive, neutral, or negative modifier.

In implementing the above highlight options for non-standard data, it was realised that a similar facility for the standard data might be potentially useful. Highlight options were therefore included for interaction mechanisms and interaction assistance so that an evaluator can identify which interaction mechanism(s) or interaction assistance types are of interest, and highlight their location within the data visualisation.

- *SUIT supports query generation based on combined spotlight and selection results*

It was recognised that the above spotlights - known as filters in the SUIT environment - could be combined to enable an evaluator to establish and build up a query over the data set. Obviously there are a number of boolean operators that can be used to combine the constituent parts of a query - for example, AND, OR, NOT. However, the first visualisation environment prototype restricts query combination to AND on the basis of its simplicity and the fact that it is likely to narrow a selection as opposed to the ability for OR

to continually expand it. A filter history is maintained in order to remind the evaluator of the query criteria at any given time (see section 6.3.2).

- *SUIT allows evaluators to record the results of data manipulation*

Dawkes *et al* suggest that the facility to store a query allows a historical record of users' investigation to be maintained and also allows the users' to return to and continue an old query (Dawkes et al., 1996). A facility is therefore provided to allow evaluators to snapshot the result of applied analysis or manipulation and thereby maintain a record of their investigation. As manipulable visualisations in their own right, snapshots allow an evaluator to either return to and extend a previous investigation, or to effectively run several investigations simultaneously (see Shneiderman's history and extract tasks). Both the original visualisation and each subsequent snapshot are supported by an electronic notepad that lets an evaluator record notes or comments regarding the associated data visualisation. Finally, facilities to clear applied highlights and to restore the visualisation back to its initial state are provided to enable an evaluator to clear the results of an investigation in preparation for further analysis.

Dawkes *et al* observe that curiosity is the driving force behind users' interactions during the initial steps of data exploration - users play with data in an instinctive manner to satisfy their appetites for 'what if' type questions (Dawkes et al., 1996). Although it is anticipated that evaluators using SUIT will be more directed than the users of the general browsing facilities referred to by Dawkes *et al*, it is hoped that the data representation and general functionality provided within SUIT will enable evaluators to satisfy exploratory curiosity. Conversely, it is also hoped that the design of the visualisation environment will assist evaluators to perform specific directed investigation of important issues when comparing UIDTs.

Combined with the facilities to tailor, use, and manage electronic frameworks (as discussed in section 6.3.1), the visualisation environment contributes significantly to the automated support for the activities performed during a SUIT evaluation.

6.4 Summary

Based on observations of the use of the paper-based version of SUIT, together with intrinsic characteristics of the SUIT methodology itself, this chapter started by outlining a series of requirements for a new SUIT data visualisation environment. In abstract terms, those requirements were to provide the ability to: (1) create, tailor, use, and manage instances of the SUIT framework; and (2) visualise and analyse the collected data.

Given the research challenge presented by the latter, the basic set of the former requirements was met via exploitation of the facilities in Microsoft® Access97. Support for the more complex requirements regarding the use of the frameworks was not considered feasible within the scope of this research, but is discussed in chapter 8.

A tabular spatial layout was adopted for the visualisation environment given the structure and complexity of SUIT data. Existing tabular visualisation environments were considered for use with SUIT but were found to be inadequate - although they could have been used they would have required significant adaptation, and so it was decided to implement a unique visualisation environment catering specifically to the needs of SUIT data. Abstract interaction principles that informed the manipulation facilities within the environment were inspired by Shneiderman's principles and existing environments - for example, focus+context. The concrete manipulation and analysis facilities included in the environment are based on the abstract techniques and those that are hypothesised to be required to support the specific analytical activities according to the SUIT methodology.

A graphical representation of SUIT data was designed to exploit, and thereby assist, the human capacity for pattern matching. Graphical representation is restricted to the primary data - that is, interaction mechanisms and assistance - with the remaining data available on demand to reduce clutter. A combination of colour-coding, positional coding, orientation, and straight line configurations has been used to graphically represent the interaction mechanism and assistance data. A greyscale has been used to encode abstractions over functional provision. Furthermore, colour highlighting has been used to spotlight individual cells on the basis of selected criteria. Given the number of colours required within the graphical data representations used in SUIT, colour choice was guided and informed by the set of twenty-two colours of maximum contrast.

The SUIT data visualisation environment has been implemented as an initial - albeit comprehensive - prototype, given that the optimal solution will only ever be achieved via a process of iterative design and user testing. The following chapter describes and presents the findings of a qualitative study that constitutes the first step towards iterative refinement of the SUIT visualisation environment. Future iterations are discussed in chapter 8.

CHAPTER 7 : QUALITATIVE STUDIES OF THE USE OF THE SUIT DATA VISUALISATION ENVIRONMENT

7.1 Introduction

This chapter describes a study of the use of the SUIT data visualisation environment, the principal aim of which was to investigate the manner in which evaluators used the environment to compare and analyse UIDT evaluation data. The study also identified usability issues related to the user interface to the visualisation environment.

This chapter begins by discussing the motivation behind the study, its design, and the manner in which it was performed. Following a summary discussion of observed usability issues and defects, the chapter considers the strategies adopted by the subjects to perform a project-specific data comparison and analysis task. After a brief examination of the experimental validity of the study, the chapter concludes with a discussion of the implications of the observed usability issues, and considers the manner in which the observed data comparison and analysis strategies can be used to develop a strategy map to guide future UIDT evaluators through these aspects of SUIT-specific UIDT evaluation.

7.2 Motivation

As discussed in chapter 6, there is generic information available regarding the manner in which people explore and compare data. For example, Shneiderman's visual information seeking 'mantra' -

Overview first, zoom and filter, then details on demand

- suggests the order of scale at which people like to explore information (Shneiderman, 1998). Similarly, his four-phase framework for search strategies - (1) formulation; (2) initiation of action; (3) review of results; and (4) refinement - defines the manner in which people organise a search over a data set (Shneiderman, 1998). There are examples of tabular visualisation environments that are based on the principles of Shneiderman's mantra and framework (Rao and Card, 1994, Spenke et al., 1996) from which the SUIT visualisation environment drew inspiration and guidance (see chapter 6).

However, given that there is no alternative tool currently available that is designed to support the non-textual comparison and analysis of UIDT evaluation data, there is at present no corpus of knowledge specifically about the manner in which evaluators perform the task of comparing and analysing UIDT evaluation data when supported by such a tool. In particular, there is no knowledge about the strategies adopted by evaluators when supported during the comparison and analysis of UIDT data according to the project-specific evaluation concepts promoted by SUIT.

The motivation behind this study is therefore to establish an initial corpus of information regarding the strategies for comparison and analysis adopted by evaluators when using the SUIT visualisation environment to complete project-specific UIDT selections according to the SUIT methodology. Not only will this knowledge inform future development of the SUIT visualisation environment, but it will also contribute to the creation of a data comparison and analysis strategy model for use with the environment by future evaluators - that is, an optional guide to possible data comparison and analysis strategies. More generally it will contribute to an improved understanding of the way in which UIDT data might be compared and analysed in any future UIDT or CASE tool visualisation environments.

7.3 Designing the Study

Given the above motivation, the study was regarded as an investigative process designed to provide *qualitative* data about the strategies of UIDT data comparison and analysis adopted by the subjects. This section discusses the primary concerns that had to be addressed when designing the study - including the structure of the evaluated task and the means of data elicitation and recording - and outlines the manner in which the study was performed.

7.3.1 Knowledge Elicitation and Observation Methods

The investigative nature of the study determined that appropriate and effective techniques for knowledge elicitation and observation had to be selected. To fully investigate their adopted data comparison and analysis strategies, it was important to record both the physical actions taken by the subjects and the subjects' accounts of their actions. Before discussing the techniques for knowledge elicitation selected to record the subjects' accounts of their actions, consider first the observation technique chosen to record their physical actions.

7.3.1.1 A Method and Technique of Observation

Real-time observation is the most powerful investigative tool for describing, understanding, and asking questions about what users are doing. It helps elucidate the sequence in which users perform different tasks, and how tasks can be divided into sub-tasks in a suitable manner (Lindgaard, 1994). Placing emphasis on capturing the goals of activities rather than the precise methods of performance, passive observation enables researchers to identify user activities in situations where the specific nature of the user tasks is not entirely known prior to the study (Newman and Lamming, 1995).

The strengths of direct observation made it ideal for elicitation of the physical activities performed by the subjects during their UIDT data comparison and analysis tasks - not only does it promote the facility to record the actual tasks and their decomposition, but also the sequencing between tasks and their sub-tasks. The activities of each of the subjects were therefore videotaped during their evaluation session.

As a mechanical audio-visual fixation of an event, video records produce data that is much closer to the event itself than other means of data recording. They provide a more accurate account of what *actually* happened than is permitted by users' and observers' often flawed recollections - they overcome the gaps between what users say they did and what they actually did. The permanency of video records also means that researchers can return to them time and time again and, in the course of repeated viewing, previously invisible phenomena often become apparent and increasingly deeper orders of regularity in users' behaviour are revealed (O'Neill, 1998). Although subjects may be conscious of, or anxious about, the video camera to start with, within minutes they normally focus on their task and ignore the videotaping (Shneiderman, 1998). Video records are criticised for the fact that they only record events from a particular camera viewpoint and miss what is not in shot. Furthermore, no matter the extent to which the immediate situation is recorded, prior events outside that situation are not recorded (O'Neill, 1998). For the purpose of this study, however, these disadvantages are limited by the fact that the observation is only concerned with the immediate computer task upon which the camera can exclusively focus (see section 7.4). Events over a long period prior to the session are likely to have influenced each subject's experience and attitude toward UIDT evaluation and data visualisation environments - for each subject this was determined during their post-session interview (see section 7.4).

7.3.1.2 Knowledge Elicitation

Preece *et al* warn that using direct observation, evaluators tend to see only what they want to see and are given only one pass at data-collection (Preece et al., 1994). However, when combined with verbal protocols, it is possible to develop a picture of the users' physical

activity supported by insight into their underlying cognitive activity - that is, is it possible to obtain a wide range of information (for example, the way a user has planned to perform a task and reactions when things go wrong) including clues as to the users' subjective feelings about the activities (Lindgaard, 1994, Preece et al., 1994).

Simply observing the subjects' actions would not, therefore, be sufficient to determine a detailed picture of their adopted UIDT data comparison and analysis strategies - and in particular their underlying rationale. Direct observation was therefore supplemented with protocol analysis/think alouds and question asking protocols (see below) during each interactive session. An interview was held with each subject at the end of their interactive period with the visualisation environment to clarify actions taken by the subjects and obtain some further idea of their reaction to the environment. In order to ensure that subjects were asked the same subset of questions during the course of their individual interviews, and that the questions were phrased in the same manner for each, a minimal set of core questions was drawn up for use in semi-structured interviews with each subject (Lindgaard, 1994, Preece et al., 1994). The nature of these questions is outlined in section 7.4.

Protocol Analysis/Think Alouds

A protocol is a verbal account given by the people who perform the tasks (Lindgaard, 1994). To conduct concurrent protocol analysis[69]/think alouds, subjects need to be convinced to verbalise their thoughts, ideas, known facts, plans, beliefs, expectations, and doubts while performing their task (Lindgaard, 1994). The technique of protocol analysis/think aloud allows a researcher to gain an appreciation of the task and system from the users' perspective. It does, however, place added strain on the user who is then required to do two things at once. This often results in silences from the user when required to concentrate intently on the immediate task. However, when combined with the use of question asking protocols (see below) such silences can be avoided - as well as prompting the users to maintain their verbal commentary, the researcher can ask investigative questions as necessary. In combination with question asking protocols, think alouds were adopted as an appropriate technique to elicit users' response and rationale during the course of their tasks, and thus elucidate the reasoning behind their adopted data comparison and analysis strategies.

[69] On the basis of a video record, it is possible to conduct retrospective protocols whereby subjects review and comment on the video-taped record of their actions. Although this places less strain on the subjects than concurrent analysis, the resulting protocols are normally less accurate - they tend to demonstrate the benefit of hindsight and rely on the users' ability to accurately reconstruct the past.

Question Asking Protocols

This participatory evaluation technique is normally based on contextual inquiry whereby the researcher 'interviews' the subject as the subject is working. By asking typical questions such as 'what are you doing?', 'why are you doing that?', and 'is that system response what you expected?', the researcher's understanding of the situation is shared with the subject so as to identify misunderstandings and misinterpretations. The questions asked using this technique are unconstrained and are relevant to the situation as it occurs - context is preserved, and much can be clarified as it is happening (Lindgaard, 1994). However, it is recommended that question asking protocols should be supplemented by other methods since the knowledge elicited using this technique is likely to become distorted (see section 7.6) (Lindgaard, 1994). The strength of question asking protocols - in terms of determining what the user is doing - and their compatibility with direct observation and protocol analysis (described above) made it suitable for use during the investigative study.

7.3.2 Defining and Piloting the Study Task

During the course of the development of the SUIT visualisation environment, discussion took place with a senior software engineering representative from a software development company based in Glasgow, Buchanan International Ltd. The organisation was selected on account of accessibility and the expertise of its software developers with respect to widely accepted UIDTs (the company train other developers in the use of tools such as Borland's Delphi™ as well as making extensive use of the tools themselves for in-house software development). The senior representative was initially asked to complete a generic tool profile for as many of their UIDTs as possible. He subsequently completed a template for each of Borland's Delphi™ and Visual Basic ™ and expressed keen interest in being further involved in the evaluation of the SUIT system, especially the visualisation environment. It was therefore arranged that an initial pilot for the observational study would be performed by a selected software engineer from Buchanan International Ltd.

Given the representative nature of the pilot subject, it was decided that the characteristics of the task to be performed by the subject should be equally representative and, in order to provide realistic motivation, the outcome of the task should ultimately be of benefit to Buchanan International Ltd. After discussion with the company's senior representative it was decided that the selected software engineer would be asked to use SUIT to make a

justified recommendation for either Delphi™ or Visual Basic™ on the basis of the needs of a project team within one of their client organisations[70].

The pilot study was structured as follows. Three weeks prior to the study, the SUIT visualisation environment was installed on a PC within Buchanan International Ltd. and the subject was given a brief tutorial on, and user manual for, the environment. The subject was encouraged to familiarise himself with the environment in the intervening period between the tutorial session and the study session and, to that end, was provided with journal pages so that he could record comments or problems as necessary. Unfortunately pressures of work did not permit the subject to familiarise himself with the environment before the start of the session and so no data was returned via the journal pages.

The study session began with a brief recap of the SUIT visualisation environment at the end of which the subject was asked to use the environment to answer a series of questions regarding a small sample data set. These questions were designed to ensure that the subject had achieved a basic level of understanding of the concepts and environment he was going to be asked to manage during the main study task. A video camera was placed behind and slightly to the side of the subject and was focussed on the monitor and surrounding area such that it could record screen activity and his actions whilst remaining unobtrusive to the subject. The session, which was video- (and therefore audio-) taped, lasted approximately 20 minutes.

Following the introductory session, the evaluation task was discussed with the subject who had previously been briefed by the senior representative. The subject was then advised that the session was going to be videotaped and was asked to think aloud as he performed his data comparison and analysis task. Throughout the duration of the task the researcher prompted the subject to maintain his verbal commentary, asked investigative questions where necessary, and provided assistance where required in order to allow the subject to continue his task. This session lasted approximately 2½ hours.

Immediately following the interactive session, the subject was asked to participate in a brief interview session. The video camera was left in situ in order to exploit its audio channel to record the interview, and to capture any potential gestured reference to the SUIT visualisation environment made by the subject during the interview. The subject was then asked a series of pre-defined questions (see Appendix F) and additional questions that were

[70] These tools were selected on the basis that Buchanan International Ltd. had supplied a complete SUIT template for both tools which were each used and supported extensively by Buchanan International Ltd.

intended to expand on, or clarify, issues that had been observed during the interactive session. The interview session lasted approximately 45 minutes.

7.3.3 Reacting to the Pilot Study

The pilot study returned valuable results and feedback from the software engineer (see section 7.5), but it also highlighted issues that needed to be addressed prior to performing the evaluation with other subjects.

The length of time required to complete the study including the interview - nearly four hours - was not considered a feasible demand to place on subsequent subjects. While determined to maintain the reality of the study task - that is, its use of real data and the motivation for the task - it was recognised that the total time required to complete the evaluation study would have to be reduced if it was to represent a feasible demand on subjects' time. To address this, the ideal tool profile depicting the requirements of the client organisation was provided for the subsequent evaluations (as opposed to asking the subjects themselves to define a profile prior to comparing and analysing the data). Furthermore, to reduce the size of the very large data set manipulated during the pilot study, the pre-defined ideal tool profile reduced the set of required attributes. Finally, it was decided that future subjects should be presented with the data visualisation in a state where it was ready to manipulate rather than have to enter the correct parameters to generate the visualisation before starting their data comparison and analysis task.

Experience with the pilot suggested that there was no real benefit in videotaping the tutorial session. It was therefore decided that the tutorial session in future studies would not be taped but that the video camera would be in place so that subjects could become accustomed to its presence prior to the main evaluation session.

Towards the end of the pilot study, the subject commented that he was beginning to find it harder to distinguish details in the visualisation which he attributed to visual fatigue on account of the intricate nature of the data representation. It was therefore decided to inject a minimum interval of 1½ hours between the tutorial session and the actual (reduced) evaluation session.

The pilot study uncovered a restriction on the use of the SUIT visualisation environment. During the course of the study it became apparent that the functionality of the software was restricted to generic UIDT comparisons when installed on any PC other than the development PC[71]. Subsequent sessions had, therefore, to be run on the (under-

[71] This was later attributed to an unidentified software dependency that was not a direct result of the coding.

resourced) development PC - removing the possibility of using screen-grabbing software - and observational records of each were restricted to the videotapes.

7.4 Performing the Study

Five subjects were selected on the basis of their abilities, interests, and availability - all five were software developers, one was an expert in the field of CASE tools, and one was an expert in the development and evaluation of a range of software tools. Four subjects are researchers within the Department of Computing Science at the University of Glasgow, and one comes from an external organisation.

Prior to their evaluation session, each subject was given a short paper outlining the SUIT methodology and framework with which to familiarise themselves with the concepts behind the system, and the sort of task they would be given to perform (Lumsden and Gray, 2000). Each subject was given a short tutorial on the SUIT visualisation environment and, provided with a small data set, was given as long as required to investigate and familiarise themselves with the environment before being asked to answer a series of questions designed to ensure that they had acquired the same basic understanding of the environment prior to using it for the evaluation task (see Appendix F). Each subject was then asked to return in 1½ hours (with the exception of one subject who could only return after 4½ hours).

On their return, the subjects were each given 5 minutes to re-familiarise themselves with the environment. Each subject was then asked to read an outline of the evaluation scenario and their task whilst the researcher generated the required visualisation and started the video camera. After clarifying any questions about what they were being asked to do, the researcher asked each subject to outline their proposed strategy for tackling the data comparison and analysis task. After describing their anticipated strategy, each subject was provided with a pen and paper to allow them to take written notes should they wish, and asked to begin their task. They were reminded to think aloud as they performed their task and were advised that the researcher might ask questions to clarify their comments or actions if required. They were also advised that they were free to ask the researcher for assistance if they were unable to continue with their activity and needed guidance.

At the end of their interactive session, each subject was asked to participate in a short interview session during which they were asked the series of questions outlined in Appendix F combined with subject-specific questions that arose as a result of observing their activities during the interactive session. Together, the interactive session and interview session lasted between 2 and 3 hours depending on the strategies adopted by

each subject and the extent to which they expanded on answers during the interview session.

7.5 Results of the Evaluations

The videotape of each subject (including the pilot) was analysed to: (1) generate a content log for the interactive session; and (2) generate a transcript of the interview session (O'Neill, 1998). Each content log describes the activities performed by the user and provides a synopsis of any discussion that took place during the session between the subject and the researcher. The interview transcriptions are a verbatim account of the discussions that took place during the interview sessions.

In light of Preece *et als'* claim that, when using direct observation techniques researchers have a tendency to see what they want to see, each content log was verified to ensure that they accurately represent the videotaped record for each subject. To do this, a five minute clip from each video record was selected from a point fifteen minutes into each interactive session, and the corresponding section from each content log was identified. Given each content log extract, an independent assessor was asked to watch the corresponding video clip and determine whether the textual record in the content log accurately captured the activities performed (including errors resulting from usability defects) and the sequence of the activities, and whether it represented an accurate synopsis of any subject-researcher dialog. After viewing each video clip and correlating it with the associated content log extract, the independent assessor concluded that the content logs were representative and that she was convinced that they were an accurate synopsis of the course of the interactive session.

Given the principal aim of this study, analysis of the content logs primarily focussed on examining the sequence of activities performed by each subject. The results of this analysis are therefore the primary consideration during discussion in this section. However the content logs and interview sessions identified a number of interesting usability defects within the environment. These will be presented briefly before considering the data comparison and analysis strategies in detail.

7.5.1 Evaluation of Usability Defects

A number of usability defects/issues were highlighted during each interactive session and interview. While no statistical significance can or will be attributed to these observations, they are worth discussion not least because it is important to address the identified issues in future versions of the SUIT visualisation environment prototype (see chapter 8).

Illustrations of usability defects are interspersed with descriptions of user activity within each content log and with other discussion in the interview transcripts. It was therefore decided to extrapolate and categorise the defects for analysis. Lindgaard suggests that usability defects tend to fall into the following categories:

- *navigation* - this refers to the ease with which a user is able to move around the system. It covers issues such as the layout and understandability of menu options; the ease with which a user can identify current, previous, and future positions within a sequence of dialogs; and the means by which a system assists a user to recover from error - for example, error messages, help information, and hardcopy documentation that can either assist or hinder smooth navigation;
- *screen design and layout* - this is concerned with the way the information is presented on the screen to the user. Problems of usability tend to surface in this respect when too much has been crammed into the screen space or when screen objects are incorrectly/badly aligned and as a result hinder easy scanning. Other issues within this category include the legibility of characters, and the identifiability of fields;
- *terminology* - this refers to the use of words, sentences, and abbreviations. Problems in this respect can arise if the terminology ('jargon') used is either inappropriate for the user population or absent where required, or where the vocabulary used is either ambiguous or lacks meaning (for example, field captions, codes, prompts, commands, or the introduction of concepts);
- *feedback* - this is concerned with the way the system communicates the results of user actions or the state of the system to the user. Examples of such communication include error messages, confirmation messages, highlighting, and regularity of response times;
- *consistency* - this reflects the degree to which the system performs in a predictable or standard fashion;
- *modality* - this refers to the various states of the system operation that the user must understand - how many different modes the user must learn, and how easily a user will be able to identify any given mode and move between modes;
- *redundancies* - this is principally concerned with repetitious or unnecessary aspects of the user interface - for example, elements of the user interface that are never used or irrelevant messages and prompts;
- *user control* - this is concerned with the users' feeling of being in control of the system. Usability issues that arise in this respect include situations where actions are initiated or controlled by the system or where the user is not informed about background processing. This relates to the degree of confidence the user has in the system;
- *match with user tasks* - this category is principally concerned with the degree to which the system matches the users' requirements. It is a basic assessment of whether or not the system supports the tasks the user wants or needs to perform when using it (Lindgaard, 1994).

The nine categories comprehensively cover the types of usability issue it was anticipated would be identified during the study, and so this categorisation was adopted as the core structure for usability defect analysis. The last of the listed categories (match with user tasks) is the most complementary to the primary aim of the study - where the physical actions listed in the content logs illustrate what each subject did do, this type of usability defect highlights what each subject wanted to do but was not able to, given the available system functionality.

Prior to each interactive session, the subjects were advised as to the technological constraints imposed upon the study (see section 7.3.3). Although they accepted the situation, they inevitably made some comments regarding the usability of the system directly related to response times. These are noted in the content logs and transcripts but are not explicitly included in the usability analysis (in particular the feedback category) to avoid obscuring unknown usability defects.

Given that the visualisation environment is an initial prototype it was anticipated that bugs in the system, previously missed during in-house testing, would be identified during the studies. An additional category - *bugs* - was therefore added to the categorisation to capture identified coding defects. Although coding bugs are functionality rather than usability defects, the identification of a bug during an interactive session could potentially influence the usability of the system. This category is therefore included to record coding defects so that they may be corrected in future development of the environment. They will not, however, be discussed in detail in this chapter.

The content logs and interview transcriptions were examined, and identified usability issues assigned to one or more of the above ten categories as appropriate. When analysing the content logs every observation of a usability issue was counted, with repetitions of the same issue or defect being considered as independent occurrences. Although defect repetitions could have been collated into a single observation which was then assigned to the appropriate category, it was felt that this may have obscured the severity of the issue; they were therefore analysed as separate occurrences to identify the frequency with which particular types of usability defect arose. Conversely, usability defects identified in the transcripts of the interview sessions were only counted as independent occurrences if they did not relate to an issue that was observed directly during the associated interactive session - otherwise they were considered explanations of previously identified defects. This approach reflects the extent to which usability issues were identified during each study session without applying bias on the basis of the subject's discussion.

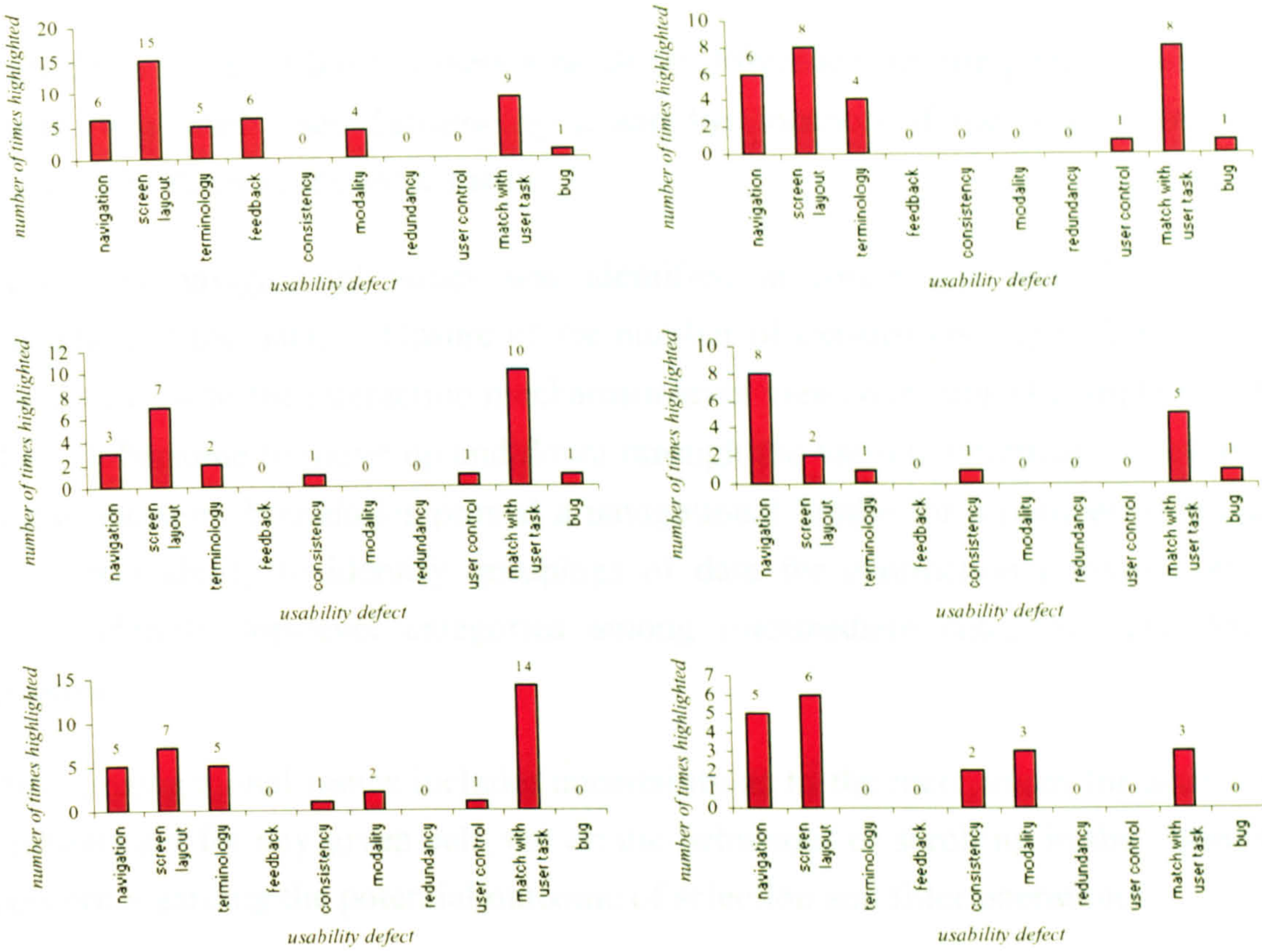


Figure 7.1 - *categorisation of usability defects for each of the 6 subjects*

Figure 7.1 shows the categorisation of usability defects for each of the six subjects. Although the subjects are not being compared in terms of the usability defects they identified, the collection of charts clearly illustrates that usability defects are generally concentrated in three categories: navigation, screen layout and design, and match with users' tasks. Barring *bugs* and with the exception of *redundancies* - for which no defects were identified - the following discussion considers the illustrated usability defects according to each of the remaining categories, placing greater emphasis on the three predominant categories outlined here.

Navigation

The primary navigational issues, identified by four out of the six subjects, centred on the contraction operations provided in the environment. Not only did subjects express confusion regarding the potential outcome of applying a contraction operation - that is, the extent of the contraction - in any given situation, but they were also unsure as to the applicability of the operations and as to the appropriate component on which to apply them.

The limited width of the columns made it difficult for some subjects to achieve accurate mouse clicks and selections. This was especially true for activating the column-specific pop-up menu, and resizing columns. Two subjects had difficulty locating menu options - in particular, the means by which to show interaction assistance data. It was later

determined that these difficulties were a result of: differences in interpretation of the menu and menu item labels (see Terminology); and the location of the required menu item relative to other items in the same menu.

A selection of navigational issues was identified in connection with the hierarchical representation of the data. Unsure of the number of expansions required to move from the top-level view to the interaction mechanism level view, one subject complained that he found it cumbersome to move up and down through the various intermediate levels. The absence of category boundaries proved a navigational hurdle for a number of subjects in terms of their ability to identify groupings of data for contraction purposes, and their ability to identify top-level categories among intermediate categories and low-level components.

Additional navigational issues include: uncertainty as to the mechanism for activating the information card for any given cell; the erratic behaviour of scrolling in the visualisation; and concern regarding the potential outcome of selection and filter operations.

Screen Design and Layout

Identified usability defects regarding the design and layout of the screen were dominated by two issues: the design of the component labels, and the absence of category and component boundaries. Four out of the six subjects complained about the orientation and font size of the component labels. While attempting self-analysis as to the reason why the orientation of the labels encumbered his ability to read them, one subject suggested that it was because he was practised reading from left-to-right and top-to-bottom but was far less familiar with reading from bottom-to-top (the orientation of the labels). He did conclude, however, that it was something that users would learn to do, and as his interactive session progressed he became quite adept at identifying the components and his associated complaints diminished accordingly. This was true for all bar one of the subjects who indicated that he would not be willing to use a tool with vertical component labelling[72]. In light of the problems with the component labels, a number of subjects suggested that they might prefer a tree-based visualisation structure. There is considerable scope to investigate this alternative orientation of the visualisation, although initial consideration of this approach suggests that it may not be as simple as these subjects indicate (see section 7.7.1). One subject commented that the table layout did not mirror the paper-based version of the SUIT framework tables, and speculated that this might potentially cause problems if an evaluator was working between the two.

[72] It should be noted that the subject in question failed to pay adequate attention to what he had been asked to do and chose instead to use the session to conduct a critical analysis of the environment - a number of his criticisms therefore lack the justification of real task use that supports the defects outlined by the remaining subjects.

The majority of subjects identified the absence of component and category boundaries as a deterrent to their interpretation of the hierarchical relationships between the components, especially when more than one top-level category was fully expanded. In particular, one subject commented that the absence of grouping boundaries around component sub-division (that is, instantiation, configuration, and layout) prevented her from quickly identifying and comparing these groups across the various tools.

Not only did the restricted width of the columns hinder accurate mouse clicks, but it also made the first colour swatch representing graphical manipulation difficult to identify when cells were highlighted. Furthermore, on account of the narrow columns and small font size, one subject found it difficult to trace vertical alignment among the colour swatches, and to correlate cells with their column headers.

A selection of complaints were targeted at the tool tips. In particular they were criticised for obscuring data of interest when they appeared, and for not remaining visible long enough to read their contents.

A number of additional usability defects were identified by individual subjects in relation to the design and layout of the screen. For brevity, only a selection of the more major of these will be discussed here.

When discussing the issue of visualising the interaction assistance information, one subject raised a number of interesting points, as highlighted in the transcription extract shown below. It should be noted that in this, and all subsequent extracts from the interview transcriptions, [s] refers to the subject and [r] to the researcher.

[s] I think the…interaction assistance…should be shown from the outset and let the user hide that…

[r] …so in other words, the opposite way to the way that I have done it…?

[s] yes…and I am also aware that you are trying to minimise the amount of information that is there already…there is enough there already…eh…but…if you keep it the way it is then somehow draw the user's attention to interaction…interaction assistance…in some other way…perhaps…I don't know…start off with the key [demonstrating with the SUIT visualisation environment's key panel] or…maybe when the user first boots up the application they get the key straight away so that they say 'oh…this is interaction assistance so let's find out about that' …

[r] right…right…

[s] …because if it is not there…and it's not shown initially…where it's on the menus…well…the assistance is important…

This response, although likely to be a reflection of his confusion concerning the modality and terminology surrounding access to interaction assistance data (discussed below), suggests that the design of the visualisation environment may not encourage an evaluator to examine interaction assistance on the basis that its presence is not obvious.

During the interactive sessions, the majority of subjects preferred paper and pencil note taking. Reasons varied from the fact that it allowed subjects to introduce their own notations to the fact that spelling was not perceived to be as important in scribbled notes (compared to electronic ones). One subject suggested that the electronic notepad should therefore be resizable in order that maximum screen space can be allocated to the data visualisation by evaluators who do not wish to use the note talking facility.

Despite the considerable use of colour in the visualisation, with minor exceptions, this was not a source of usability defects. One subject, who initially expressed concern about his ability to differentiate the colours used to represent interaction mechanisms and assistance, later decided that the positional coding in the representation and visualisation key meant that it did not deter or prevent him from using the system. The only subject to use the greyscale slider sampler on the visualisation key expressed concern that, because the background of the sampler was not the same as the background of the greyscale cells in the visualisation, he thought the key was perhaps giving him misleading results:

> [s] …I think there is a slight problem with judging the depth of the greyscale [pointing to the key] because of the background…I don't know whether my…own perception of greyscale is good enough to really make…em…strong judgements…on the other hand, I think the…importance of the greyscale is the relativity not necessarily the…the absoluteness…

Using the key the subject was, however, very accurate in his estimation of the percentage functionality and, as the above extract highlights, he finally determined that he was more concerned with the relativity of the greyscale than the actual values it represents.

Tool identification was noted to be missing from the information card for each cell. It was also noted that the tool identification labels (that is, row labels) did not remain visible during scrolling which made it difficult to correlate observations with their associated tool.

Finally, as mentioned in section 7.3.3, the level of detail and complexity of the data representation caused visual fatigue for the pilot subject, whose tutorial and interactive sessions were considerably longer than those of the subsequent subjects.

Terminology

Use of certain terminology was noted to confuse a number of the subjects. In particular, half of the subjects found it initially difficult to apportion meaning to the term 'other interaction mechanism', and some had similar problems with the scope of 'miscellaneous comments'. A number of subjects commented that they found some of the component identifiers ambiguous. More serious, was the confusion surrounding the use of the terms 'filter', 'highlight', and 'select'. Aggravated by the fact that the system uses the terms 'filter' and 'highlight' synonymously, the subjects' interpretation of the difference between these three terms often did not conform to the use of the terms in the visualisation environment. This often led to anxiety when the returned results of selected operations did not match with those anticipated by the subjects. Similar terminological ambiguity was illustrated for 'show' versus 'highlight' in terms of interaction assistance data, and 'save' versus 'snapshot' for the visualisation as a whole.

Feedback

Feedback was not observed to be a primary contributor to usability defects during the interactive sessions. The most noticeable defect in this respect was the lack of explicit feedback following a filter or selection operation that returned an empty set - subjects were left confused as to the lack of highlights or components respectively. Similarly, after applying a selection operation, subjects were unable to confirm the semantics of the returned data set on the grounds that there is no explicit identification of the currently active selection. Lack of feedback of this nature noticeably diminished subjects' confidence in their ability to correctly apply filter and selection operations.

Other minor identified issues include the lack of visual feedback during the drag-and-drop action to relocate columns - this caused concern for subjects who were unsure as to the effectiveness of their actions, and prevented them easily identifying their current position relative to their intended target position. Some subjects did not consider the display speed of the tool tips to be acceptable. One subject suggested that the system should adopt a status bar that is continuously updated with the identification of the component that is located beneath the mouse cursor, such that components could be identified more quickly with less intrusion into the active part of the display.

Consistency

Usability issues concerning consistency were focussed entirely on the behaviour of the contraction operations. In particular, one third of the subjects did not appreciate the consistent behaviour of the global contract-all operation. The semantics of a contract

operation are such that the selected node and all its siblings are contracted (hidden) and replaced by their parent. To remain consistent in this respect, the global contract-all operation - designed to contract all components across all levels of expansion - was implemented such that the result was the parent of all components, the root of the component hierarchy (that is, the actual tool identification column). Although this is semantically consistent, the fact that the top-level categories are hidden irritated the subjects. As a result, they suggested that the behaviour of the system would be more intuitive if the effect of the global contraction operation was to return the visualisation to the top-level categories. It also surprised a number of the subjects that the contract operation is not a direct counterpart of the expand operation.

Modality

In general, subjects did not encounter many usability problems directly related to the modality of the system. Two subjects became confused regarding the difference in modality between highlighting interaction assistance data and making it visible. This is likely to be a direct consequence of the terminological issues previously identified. One subject became concerned as to the state of the system when he applied a selection that returned an empty set, and another was unsure of his operational mode after applying a series of filters and selections over the data set. Finally, one subject was unsure of the operational relationship between the ideal tool profile and the real tool profiles - that is, he was not certain as to the manner in which the ideal tool data was incorporated in the various selection and filter operations.

User Control

Like many of the identified defects, the contraction operation was the root cause of subjects' perceived lack of control - one subject shied away from using the operation on the grounds that she was uncertain about what the system would do if she was to apply it, and she did not want to lose the results of her previous analysis. Although principally a result of the operational speed of the system, one subject avoided using the electronic notepad on the grounds that he did not know how detrimental its use would be to the function of the system as a whole.

Match with Users' Tasks

This category is particularly interesting since it addresses one of the core experimental aspects of the visualisation environment - that is, determining the functionality that should be provided in a tool of this nature for the specific task of comparing and analysing UIDT evaluation data according to the SUIT methodology. The subjects identified a series of

extensions and additions to the provided functionality, many of which proved to be a common request.

The most requested addition was the introduction of priority information to the visualisation. Four out of the five subjects who completed their task[73] commented that the absence of component prioritisation made it difficult for them to determine the relative significance of their observations with respect to the project for which they were selecting a UIDT, and as a result they were less confident in their decision. It was interesting to note that the subject for whom this was not a problem was the pilot subject who generated the ideal tool profile and therefore had background knowledge of the project and its requirements. This suggests that the need for priority information may have been the result of lack of detailed prior, or background, knowledge of the project. One subject suggested that a facility to toggle between the component-based view and a priority-based view would be very useful whilst avoiding over complicating the visualisation.

Half of the subjects requested the facility to universally expand and contract components by one level of abstraction - many suggested that they might have been more inclined to consider intermediate levels of data had this been available. Furthermore, it was suggested that there should be a one step facility to fully contract all low-level components belonging to the same top-level category. Similarly, half of the subjects suggested that they should be able to select multiple components over which to apply the available analysis operations. One subject went further to suggest that he wanted to be able to mark several components for consideration, and then hide all non-marked components so that he was left with only those components upon which he wished to focus.

One third of the subjects wanted greater control over the filters. This ranged from being able to specify the manner in which the filters are joined to being able to edit the filter history and thereby affect filter application. One subject wanted to be able to apply several filters independently in parallel over the same data set such that she could simultaneously compare the results; another wanted to be able to restrict filter application to a select group of components. One subject requested the addition of filters to identify the location of recorded comments (for example, cognitive demand comments) - both generally, and independently for each of the modifiers.

Two subjects suggested that the inclusion of the ideal tool profile in selection operations should be optional, and furthermore, that evaluators should be able to specify which of the tools to include in any selection operation. Similarly, two subjects requested that the greyscale cells be extended to include representation of the overlap in percentage

[73] Subject 5 did not complete his comparison and analysis task.

functionality provision between different tools. There was also a request to more easily identify the actual percentage value for any cell without having to open its associated information card - the status bar or tool tips were highlighted as acceptable solutions.

Two subjects requested more sophisticated note taking/report generation facilities. In particular, one requested that the electronic notepad include a transposition of the table structure (that is, the row and column identifiers) such that he could more easily record comments against the appropriate component. The other subject suggested that the tool could provide the facility to semi-automatically generate reports on the basis of an automated comparison of the data components. On the grounds that it is not a primary requirement for data comparison and analysis (the goal of the visualisation environment), the automation of report generation is not something that was initially considered for the first SUIT visualisation environment prototype. It does, however, present an interesting avenue for future development of the environment.

In response to problems he encountered related to the size of the columns, one subject suggested that the environment should include a facility to globally resize all columns.

Other minor requests for additional functionality included: undo and online help facilities; a numerical tally per tool for filter highlights; the ability to switch tool tips on and off; and a suggestion that the visualisation key should be visible by default at start up.

7.5.2 Observation of Comparison Process

The content logs were examined to identify and extract user actions - the activities or tasks to which their actions correspond, and the sequence in which they were performed. In general, subjects spent a period of time at the beginning of each session exploring the visualisation and testing some of the available features relative to the data set. For most subjects, a noticeable strategy only became apparent after this initial investigative period. Although subjects' exploratory actions are discussed in the following sections, attempts to model the strategies focus on the more systematic use of the system[74].

Section 7.7.2 combines the individual strategies into a single route map or guide for UIDT data comparison and analysis using the SUIT data visualisation environment.

7.5.2.1 The Strategy Model Notation

Initial attempts to model the strategies adopted by the subjects highlighted their complexity and the need for a tailored notation to clearly present the activities and, in particular, their

[74] Initial attempts to model *all* user actions, including exploratory activity, resulted in overly complicated diagrams.

sequencing and flexibility. Observation of the subjects' strategies illustrated that they tended to adopt (often repetitive) patterns of activity (as demonstrated in the remainder of this section), and it was therefore considered important to represent their strategies in such a way as to make these patterns identifiable. There are limited task notations available that could have been used to model the identified strategies (Gray et al., 1994, Harston and Gray, 1992). However, their structure did not manageably accommodate the observed complexity of task activity (for example, task tables become too large for practical use (Gray et al., 1994, Harston and Gray, 1992)) and they did not allow activity sequencing to be represented in such a manner as to make the patterns and flexibility of activity salient (Gray et al., 1994, Harston and Gray, 1992). It was therefore necessary to devise a suitable notation specifically for the representation of the data comparison and analysis strategies.

Given the benefits of graphical representation identified in chapter 6 – not least their facility for pattern identification – (Ahlberg and Shneiderman, 1994, Rao and Card, 1994, Chambers et al., 1983) and its use in other modelling notations (for example, Weaver, 1993, Booch, 1994), it was considered the most appropriate means by which to represent the activities and their sequencing. Early attempts at modelling the strategies identified the need for notational constructs to represent: low-level atomic user actions; black-boxed combinations of user actions – and a means to identify the corresponding expansion of the activities internal to the black-box; both bounded and unbounded repetition of collections of one or more activity; choice of action based on satisfaction of given criteria; and parallelism of task performance. In particular it became necessary to black-box user actions to simplify the diagrams and therefore increase their readability. The resulting notation is used to represent each of the strategy diagrams in the remainder of this chapter.

7.5.2.2 Observed Data Comparison and Analysis Strategies

Pilot/Study 1

Before considering the strategy adopted by this subject, it should be noted that as a result of the technical restrictions uncovered during the pilot study (see section 7.3.3) this subject was confined to using generic comparison facilities whilst bearing in mind the interaction mechanisms defined in the ideal tool profile. This therefore meant that he did not have access to the project-specific functionality that was available to subsequent subjects.

This subject began with an exploratory investigation of the data set by fully expanding all categories and examining the data at the lowest level of detail. He also explored the use of the selection options in terms of tool match and mismatch and in terms of registered

functionality. The result of this exploration led the subject to determine that he would judge the tools on the basis of the mismatch between them.

With this abstract strategy in mind, he explored the use of the interaction mechanism highlights. After applying a filter on the basis of the two required interaction mechanisms, he determined that neither tool provided any components in which both mechanisms were employed together. He therefore adapted this approach to examine one of the required interaction mechanisms at a time, and it was at this point that his strategy became more focussed and systematic.

He began by applying a filter to identify the use of one of the required interaction mechanisms. He then selected and fully expanded the first of the top-level categories. Amongst the interaction mechanism levels components, he identified points at which the tools differed in terms of functionality and in terms of the interaction mechanisms used. He also identified components for which both tools were identical. When asked whether he attributed greater importance to functionality or interaction mechanisms, the subject suggested that the provision of the functionality was the more important of the two and he therefore afforded it primary consideration. Having exhausted his examination of that category of components, he returned the visualisation to the view of the top-level categories. He then hid the top-level category he had just finished examining before fully expanding the next of the top-level categories. He repeated the above process for each of the top-level categories, hiding each of the completed categories before expanding and examining the detail of the next one.

For one of the top-level categories, the subject noted that the cells were represented in white which he correctly interpreted as indicating that neither tool provided the functionality. Although he commented that he did not really need to further expand and examine this category, he nevertheless did treat it the same way as the other categories in order to confirm his analysis.

At specific points during the examination of the low-level details, the subject took electronic notes regarding his observations. He also referred extensively to the visualisation key in order to correctly interpret the colour-coding. At one point the subject attempted to include interaction assistance in his comparison using a filter but the requested filter returned an empty set. He subsequently determined that he should have made the data visible and was then able to include comparison on this level.

After completing the detailed systematic category-by-category comparison and analysis of the data, the subject applied the selection of first mis-matches and then matches between

the two tools in order to obtain a broader perspective on the data he had been examining at a microscopic scale.

When asked to summarise the strategy he had adopted during his comparison and analysis task, the subject responded:

> [s] I began by getting used to the environment itself - I think that was a big part of it initially. I was getting used to what the menus did, making mistakes (which you drew my attention to), and learning about the SUIT environment itself. Then, having done that, the next step was trying to look at and compare the different categories and in doing so started off in the simplest way by expanding all the categories and looking at them serially just one by one and making a comparison by eye, then getting involved in filters and selecting where things differed or were the same so that eased the comparison quite a bit because we were filtering out what we didn't want to see anymore...and gaining confidence in using SUIT itself and also the fact that I had a clearer...a much clearer idea of what it was actually doing...eh...although I had some understanding of what it was doing it was not really until you use something that you can sort of get that idea...

He also explained the issues upon which he focussed during the task:

> [r] ...can you sum up what your main interests...areas of interest were when you were doing your comparison...? What were you actually focussing on most to do your comparison?
>
> [s] well...just the sort of eh...where features were offered in one tool and not in another...where the interaction mechanism in one tool was different from the one in the other...whether or not that difference meant...had a bearing on the project members...the team members that were going to be involved with working on the project...also...the...the...it's very much the differences and the things in common really...trying to compare each of the tools that way...

As already mentioned, initial attempts to exhaustively model each subject's activities (including their exploratory investigative actions) resulted in diagrams that were so intricate that their meaning and potential usefulness became obscured by their complexity. Figure 7.2 therefore represents a simplified model of this subject's actions, the aim of which is to illustrate strategy as opposed to specific action selection. For clarity, sections of the model are represented as black-boxes and their internal structure is modelled separately.

This subject adopted a relatively straightforward approach to comparing data. He differentiated between manually comparing the data at the microscopic level and comparing the entire data set at a more macroscopic level. Bearing in mind that this subject was working under falsified conditions (see above) he chose to remind himself of

the required interaction mechanisms before adopting a methodical category-by-category strategy for examination of the UIDT data. During the interview session, the subject indicated that he placed the importance of functionality over that of the interaction mechanisms used and so, per category, examined the functionality matches and then interaction mechanism matches at the lowest level of detail. He was methodical in hiding his completed categories to avoid confusion and to simplify the visualisation.

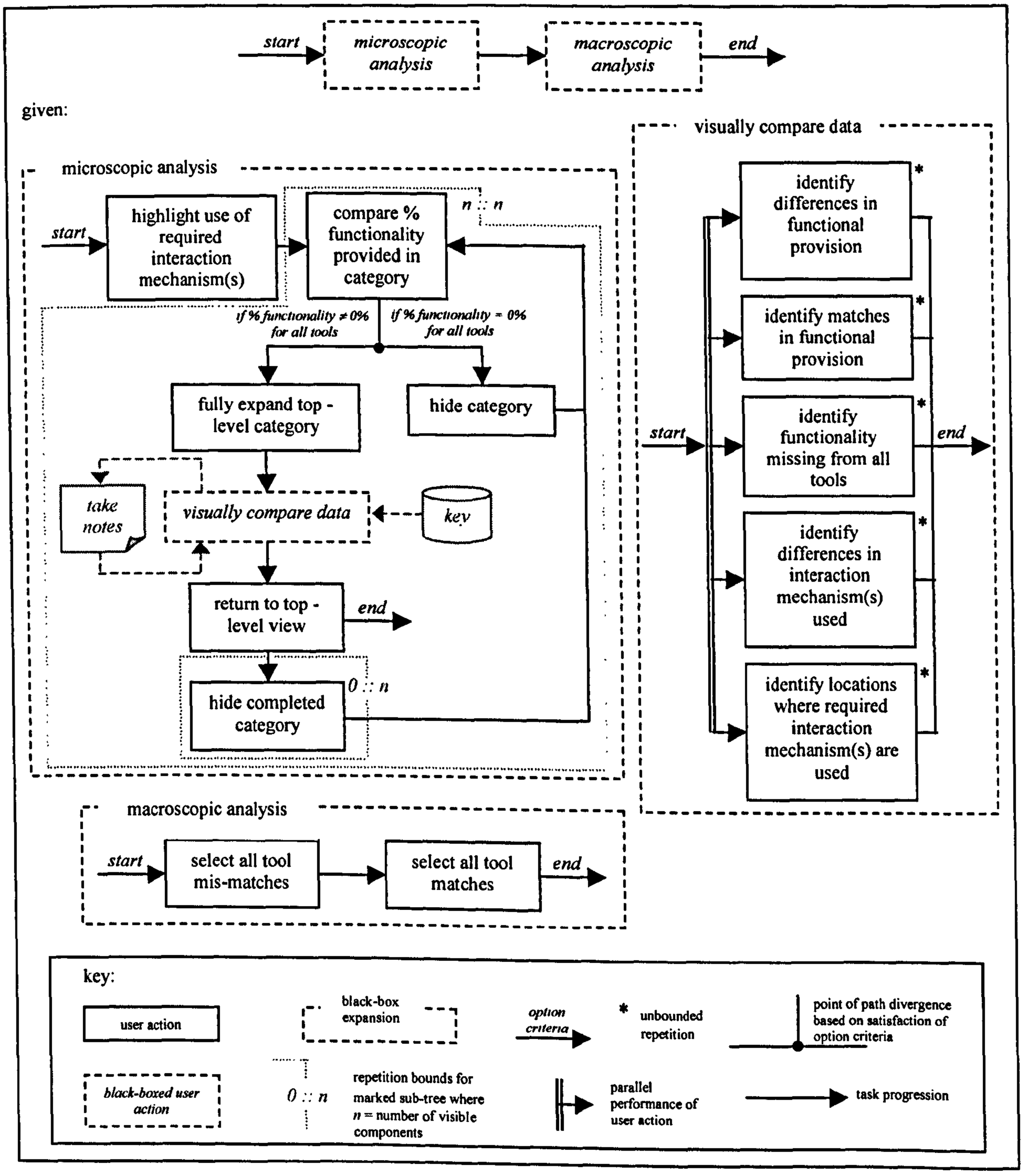


Figure 7.2 - *model of comparison and analysis strategy adopted by subject 1*

When asked to comment on his confidence in his own performance and in the results he obtained he replied:

[r] …so you have now worked with the tool for a few hours…how would you rate your confidence in your performance with it?

[s] confidence…eh…much higher than it was at the start…because I know what I am doing I think now…eh…performance…similarly quite high…I started off quite unproductive if you like because you were going through each of the categories but now when we know what to look for you can sort of narrow things down quickly and make a judgement call quite quickly…so…

[r] …so how confident are you with the results you obtained ? …would you feel confident at the end of using that that you could go and write a reasoned rationale for the selection of one tool over the others ?

[s] as confident as I am in the templates themselves…

His response suggests that the strategy outlined in Figure 7.2 was the result of increasing confidence in his use of the visualisation environment. Furthermore, the subject suggests that his productivity increased with extended use of the environment. The issue he raises at the end of the above extract is an interesting one, and is discussed in detail in chapter 8.

Study 2

After the pilot study, subjects were asked to outline their anticipated strategy prior to commencing their data comparison and analysis task. The following extract shows the response of the second subject when presented with the initial visualisation and asked to describe her intended strategy:

[r] Do you have any strategy in mind as to how you are going to compare this data?

[s] not immediately…thought that what I might try to do to start off with is look at the…umm…use the filters - the highlighting to see about the functionality because obviously what we don't really want is to produce…is a tool that doesn't have the required functionality and as the ideal profile is going to tell me what functionality is required that's going to help…now, I notice already [pointing to the scenario visualisation] that none of the them have the same amount of functionality so obviously what I am going to try and do is I am going to try and look at…to see to what extent there is a difference between…on the ones that are the darkest…[pointing to specific cells in the visualisation] I wouldn't mind seeing whether there was a…the percentage difference between those two grey ones there…so obviously I can see immediately that I am not going to be able to get the ideal OK…which is a bit disappointing since it would be nice if I could…um…but none of them match…so I suppose what I need to do is I need to find out…um…yeah…what sort of other…the interaction facilities that are available where those matches are…So I

> think my first strategy is...mmm...I think there's two ways I might go about it: I might just expand everything just to get an idea about what is there and what's not there...just to get an overall idea of the scope of the categories...just to get a context; and then I'll start looking at...highlighting different things with respect to matches and mis-matches between the tools and the profile. OK.

On a cursory examination of the visualised data, the subject correctly concluded that neither of the tools being evaluated were an exact match with the ideal tool profile. After a brief comparison of the greyscale representation of percentage functionality provision within the top-level cells, she fully expanded all of the categories to determine the extent of the data set. From the ideal tool profile cells, the subject identified the required interaction mechanisms and then superficially located the existence of these mechanisms in the visual representation of the actual UIDT data. Similarly, she observed where the two tools matched and did not match in terms of functionality and interaction mechanisms. Her final exploratory activity was to first select all matches between the tools and then all mis-matches, at which point she realised that the ideal tool profile was included in the selection process and therefore considered the selection operations as of little use (see section 7.5.1).

After restoring the visualisation to its initial state, this subject clearly identified her comparison agenda - to compare categories of components in order to eliminate from consideration those that contained no (or at least no significant) differences such that she could focus her analysis on the categories for which there were substantial and therefore decision informing differences. She began with an examination of the greyscale representation of percentage functionality in the top-level cells. After ranking the top-level categories on the basis of the degree to which the greyscale representation differed across their cells, the subject systematically analysed each top-level category - starting from the one with least difference and concluding with the one with the greatest difference.

Taking each top-level category in turn, the subject applied a full expansion to see the interaction mechanism level detail for each component. Based on the lowest, sub-component level representation she identified differences in functionality between the two tools, functionality missing from both tools, differences in the use of interaction mechanisms, and differences in the use of the required interaction mechanisms. At various intervals, she took written notes of her observations. Having completed her examination of a category, she restored the visualisation and relocated the completed category to the front of the list to 'mark' it as compared, before moving onto the next in the ranked list.

Once she had completed the above process for each of the top-level categories, she identified the two categories that represented substantial differences. Fully expanding the

first of these, she began hiding the low-level components for which there was no difference between the two tools. That done, she took a snapshot of the manipulated visualisation before restoring the original visualisation and repeating the process for the other selected top-level category. Having isolated the most significant differences between the two tools, the subject - presented with all the important information simultaneously and manageably - was then able to make her final comparison and analysis.

During the course of the session, the subject was asked to rate the importance of functionality against the interaction mechanisms and interaction assistance information. She replied:

> [s] …once I realised that there was that much difference between them I realised that I had to pay a little bit more attention to them than I had first anticipated…my first idea was functionality is what's important…is what's important…the tool has to be able to do what we want it to…but once I'd realised that there was quite a distinctive mismatch between what the ideal tool wanted and the others I felt that then I had to start thinking a little bit more towards well…which sort of interaction mechanisms…which tool is giving the most appropriate interaction mechanisms for the ideal…

Like the previous subject, she rated functionality as the most important determinant for the selection of UIDTs. However, when she realised that the tools did not match the ideal tool profile, she expanded her consideration to more fully analyse the manner in which the provided functionality was achieved (the interaction mechanisms).

Figure 7.3 maps the comparison and analysis strategy adopted by this subject. Although more extended and perhaps complex, there is much in common between this strategy and that for the previous subject, shown in Figure 7.2. Both subjects adopted a category-by-category approach to examining the data, and focussed on the differences between the tools. Both performed a period of microscopic and macroscopic comparison, albeit in opposite sequence. However, this subject extended her analysis by revisiting and isolating the categories for which she had identified the most substantial differences - this is noted as summation analysis in Figure 7.3.

When asked to determine whether her suggested strategy had been the most appropriate given the benefit of hindsight, she commented:

> [s] …um…I don't ever feel…there was no time where I felt I had done things in the wrong order…
>
> [r] …right…
>
> [s] …and I thought I should have gone back and done it differently…I felt from the way that I wanted to look at the system it was an appropriate way of going about it…

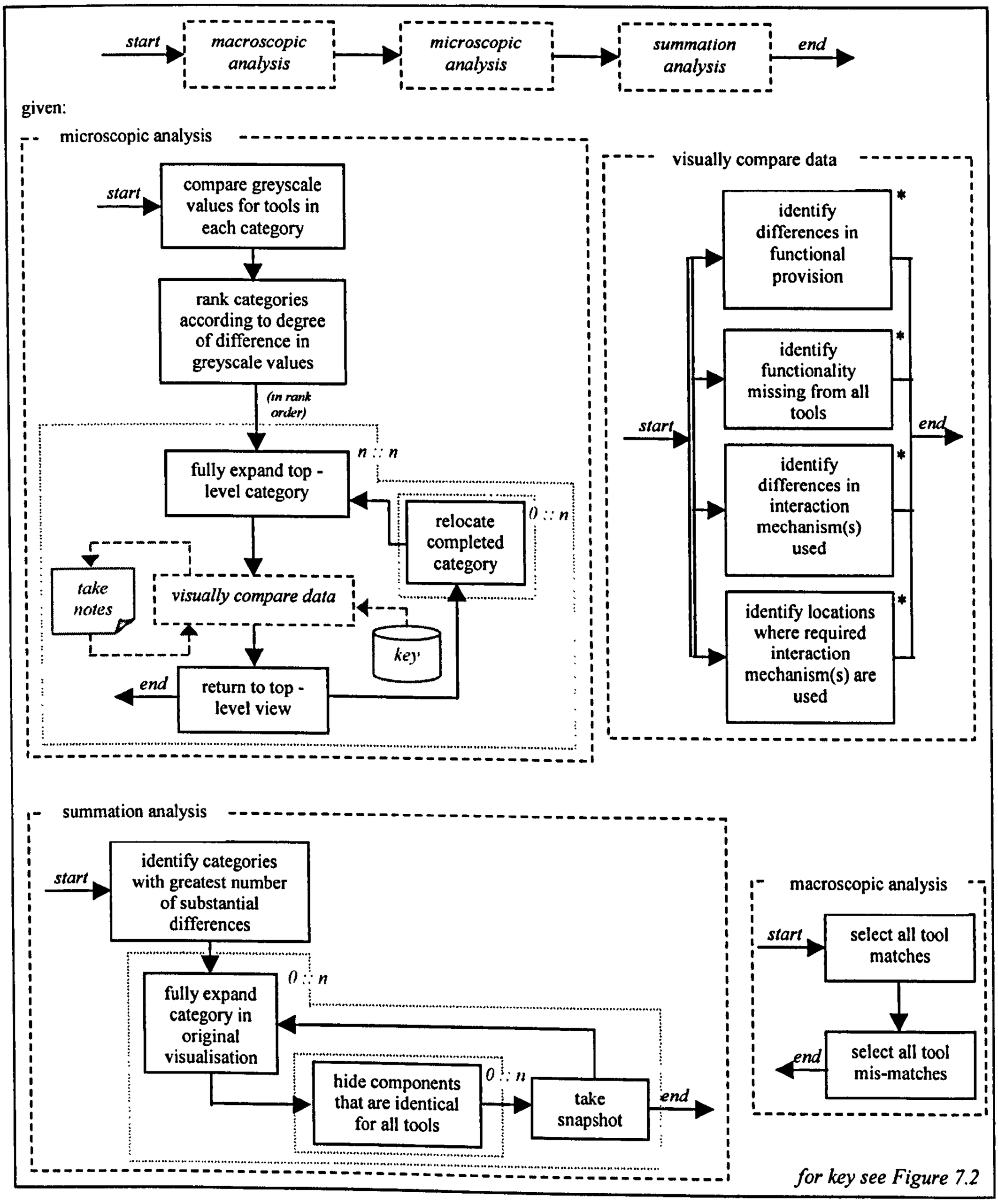


Figure 7.3 - *model of comparison and analysis strategy adopted by subject 2*

The subject was asked to comment on her confidence in her own performance and in the results she obtained given her adopted strategy. The corresponding conversation is outlined in the following extract:

```
[r]  ...how would you rate your confidence in your performance?
[s]  ...my confidence...in my strategy ?...
```

[r] …yes…your performance…

[s] …I thought I did pretty well…

[r] …yeah…

[s] …I think that I am pretty confident in the way I did the strategy…I think…I mean it obviously took a while for me to really work out what I was going to do and why I was going to do it…but once I realised that eliminating the categories that were the same and the mechanism of doing it…I was pretty sure that this was the way to do it…

[r] …uh huh…

[s] …and that having a visualisation of the categories that were the most different showing exactly where the differences were… I was pretty confident that was how…the best way to make the decisions…

[r] …so you are quite happy that you have actually identified the differences…?

[s] …yes…

[r] …and the points of interest ?…

[s] …most definitely…

The subject was not only confident that she had tackled her comparison and analysis task in an appropriate manner, but she was also confident that she had identified the points upon which to base her selection decision - that is, the points of substantial difference between the two tools.

Study 3

Before commencing his comparison and analysis task, this subject discussed his intended strategy:

[r] Do you have any strategy in mind as to how you are going to compare this data ?

[s] OK…um…I was initially…eh…recalling the existence of the searching for functionality that it does and doesn't cover and the interaction mechanisms that it does and doesn't cover, I was going to start just by generally using those two to look and see if there are any major places where…eh…where one of them is obviously better…um…other than that, no that's my immediate goals and I will then see where I go from there.

Unlike the previous subject, he did not have a well defined strategy in mind prior to starting his task. Instead, he decided to adopt a macroscopic exploratory technique in order to get a better feel for the data set and thereby clarify his strategy. The subject started by filtering the data set on the basis of the tools' percentage functionality match with the ideal tool profile. When this returned an empty set, he concluded that the converse filter would return everything and so determined that his current line of

investigation was inappropriate given the nature of the data set. He then decided to investigate the data on the basis of the match between the interaction mechanisms identified in the ideal tool profile and those recorded for each tool. Once again this returned an empty set which confirmed the subject's interpretation that he was not going to be able to approach the comparison on the basis of filtering out the mis-matches and matches between the tools and the ideal tool profile.

Like the previous subjects, he fully expanded all categories to get an impression of the size of the data set - at which point he concluded that there was too much data to manipulate at once and so decided to compare the data on a category-by-category basis.

After returning the visualisation to its initial state, the subject fully expanded the first top-level category. He examined the interaction mechanism level components in order to determine differences in functionality provision between the tools and functionality that was missing from all tools, before restoring the visualisation to the view of the top-level categories. He repeated this pattern for each of the top-level categories. At one point he returned to a top-level category and, out of curiosity, only expanded it down one level. After considering the data at that level and levels below, he determined that there was not much to be gained from viewing the data at the intermediate levels and so decided to continue with his one step top-to-bottom expansion strategy.

Once he had completed his functionality-centred comparison, he focussed on interaction mechanism and assistance data. He visualised interaction assistance data across the entire data set. He then expanded the first of the top-level categories and compared the low-level components on the basis of their use of interaction mechanisms, provision of interaction assistance, and the match between the interaction mechanisms used and those identified in the ideal tool profile. He returned the visualisation to the top-level category view and repeated the process for each of the remaining categories. Sporadically, while examining issues concerning the use of interaction mechanisms and assistance, he re-examined differences in the functionality provided by each tool.

Having completed this stage of his comparison, the subject focussed on the ancillary data recorded for each component - namely the cognitive demand and quality of feedback. After filtering the data on the basis of positive cognitive demand modifiers, he began a category-by-category examination of the component highlights. For each category, he tallied the number of highlights and examined random cells for recorded comments.

Once he had completed his examination of positive cognitive demand modifiers, the subject cleared the highlights and filtered the data on the basis of negative cognitive demand modifiers. At this point, he decided to fully expand all the categories at once

rather than continue his systematic category-by-category analysis stating that his former approach was becoming tedious. He repeated this process (clearing highlights in between) for both positive and negative quality of feedback modifiers.

When tallying the highlights for filtered modifiers, the subject was noted to selectively exclude or include highlights on components that were provided by only one of the two tools. When asked to explain his strategy he commented that in the case of negative modifiers, if one tool did provide the functionality and the other tool did not, then the functionality was the overriding issue and so the negative modifier could be ignored. However, in the case of positive modifiers, if one tool provided the functionality and the other did not and furthermore provided it well (that is, had a positive modifier), then that was the equivalent to two points in favour of the first tool. Like the previous subjects, his strategy placed greater emphasis on the provision of functionality.

Figure 7.4 maps the strategy adopted by the third subject. To avoid unnecessary repetition, the black-box expansion for quality of feedback analysis has been omitted - it is essentially the same as that for cognitive demand analysis, with the exception that the filters applied relate to quality of feedback.

Of the subjects thus far, this subject made the most extensive use of the range of facilities and data provided in the visualisation environment. He adopted a methodical - albeit perhaps inefficient - approach to the comparison and analysis of the core elements (functionality, interaction mechanisms, and interaction assistance) whereby he examined each in turn on a category-by-category basis. He then turned to the ancillary data for additional criteria on which to base his selection. He abandoned the category-by-category approach at this stage and globally compared the filter results.

At the start of the interview session, the subject was asked whether he felt the tool had supported him in his intended strategy:

[r] ...do you think you were able to follow the strategy you wanted to follow ?

[s] roughly yes...

[r] yeah ?...

[s] I went down to each of the bottom layers and then back up...down and up...down and up...down and up...and then did a similar thing for the interaction mechanisms...so...eh...give or take...actually the very initial strategy...the very initial strategy I was going to try which was the trying to find out any obvious places where they satisfy or don't satisfy failed...so after that the second strategy I tried I basically followed yes...

[r] right...OK...em...how would you say that your strategy developed as you went along?

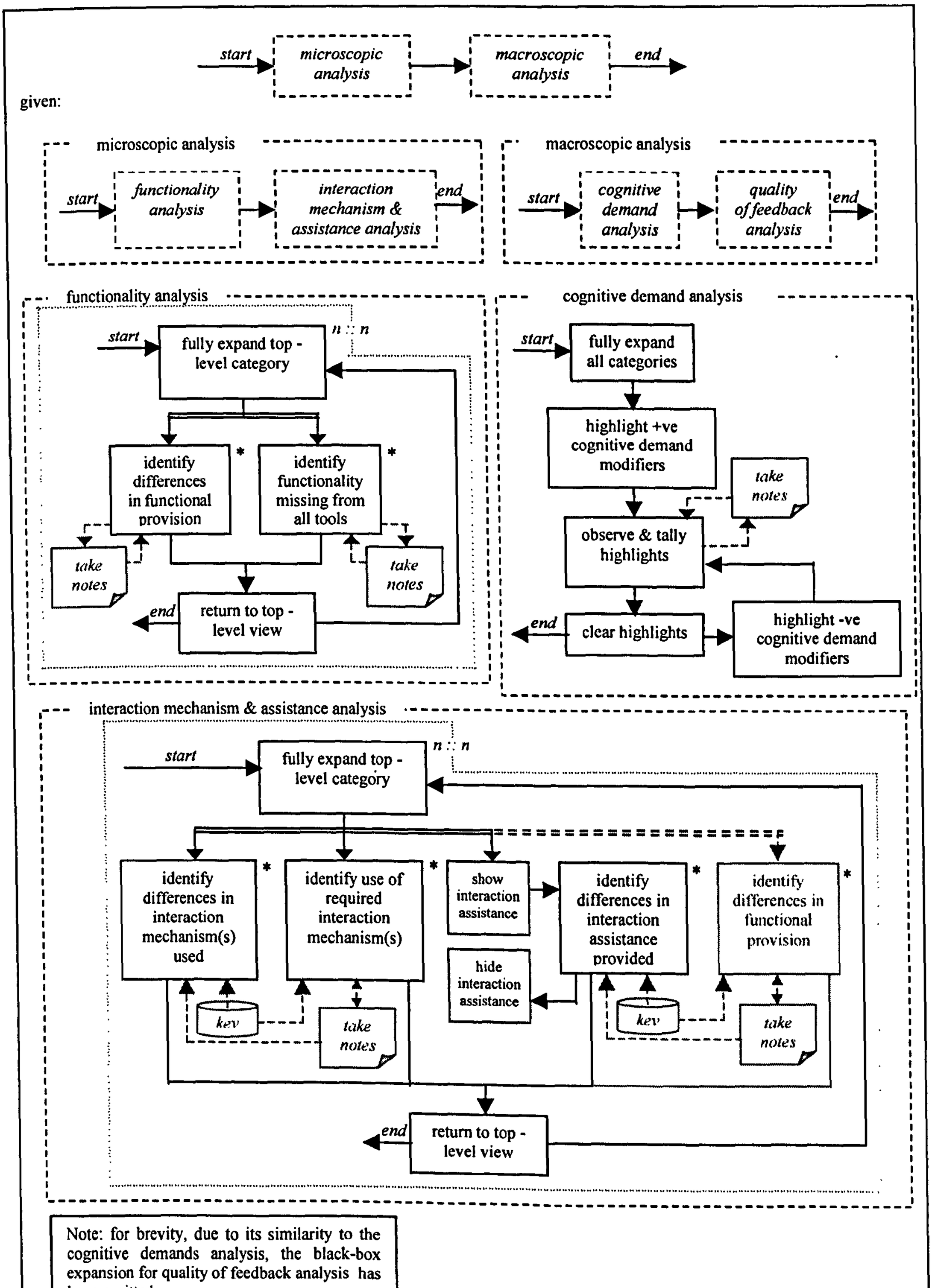


Figure 7.4 - *model of comparison and analysis strategy adopted by subject 3*

[s] I started off...I think I started off thinking a bit more about the functionality and not about the extra comments around them...

[r] ...uh huh...

[s] ...I started to think...eh...as I went along I started to think about the subtleties of what providing that functionality actually really meant...in terms of whether they supported the interaction mechanisms, the quality of feedback and so on...

[r] ...uh huh...

[s] ...um...which I wasn't thinking about so clearly to start with...

[r] ...uh huh...

[s] ...so being able to see it in front of me was helping me to start...to start really thinking about it in that format...

This suggests that the subject was able to follow his intended strategy - at least after he had determined that, on account of the data, his initial strategy had failed. This feedback also suggests that the visualisation environment was effective in terms of focussing the subject's attention on important project-specific context-sensitive issues. Chapter 5 showed that the SUIT methodology is effective in terms of directing evaluator's attention to issues of context when comparing UIDTs for a specific project. Given that the subjects involved in this study had not been directly exposed to the methodology the above feedback implies that, for this subject at least, the structure of the visualisation and associated functionality was also effective in terms of directing his attention to more than just a functional comparison.

Finally, as with the other subjects, this subject was asked to rate his confidence in his performance and in the observations he had made:

[r] how well would you rate your confidence in your performance with the tool?

[s] ...reasonable...

[r] ...uh huh...and where would you say you weren't confident?

[s] ...um...I think it's all coming back down to the importance of the...the but I'm not confident about is the importance of whether the things I have chosen are actually really significant or not...

[r] ...so it's your priority...you're not confident on whether you've based it on the right priority?

[s] ...yeah...

[r] ...what about the results you've obtained? ...again would that be based on the priority?

[s] ...yeah...

[r] ...are you confident that you have picked up the important data and been able to identify the differences between the tools? ...assuming they are of the same priority...

[s] ...yeah...yeah...assuming they are...yeah...I'm fairly sure I have...

[r] ...yeah...

[s] ...I wouldn't say 100% but probably 80% - 90%...

The absence of priority information in the environment made the subject question his performance and results to some extent (although he still rates them quite highly). After future development of the environment has addressed the prioritisation issue, it would be interesting to evaluate any change in the subject's confidence levels (see chapter 8).

Study 4

The following extract illustrates the subject's initial thoughts on the strategy she would adopt:

[r] Do you have any strategy in mind as to how you are going to compare this data ?

[s] I would probably go and look at the ones that differ radically...[pointing to the first two top-level components in the visualisation] I wouldn't bother too much about those two because that looks pretty equal...so I'd first look at the big differences and then if I still thought I needed to decide, I'd go and look at the ones where they appeared to be the same and expand them.

Like previous subjects, her initial intention was to focus on the differences between the two tools. In particular, she decided to identify the areas for which the difference was greatest and concentrate her attention on those. Unlike the other subjects, however, she did not adopt an obviously systematic approach to her comparison and analysis task. Nevertheless, she made extensive use of the various operations available within the visualisation environment and thereby considered all aspects of the data as opposed to just functionality and interaction mechanisms.

The subject began by examining the greyscale representation within the top-level category cells. Selecting the category for which the greyscale values were most different across the tools, she expanded the category by one level. After examining the greyscale values for the cells at that level, she again selected the category for which the greyscale values were most obviously different and fully expanded it. To simplify her visualisation, she hid the top-level categories that remained untouched.

Focussing on the components within her selected sub-category, the subject filtered the data on the basis of functionality mis-matches with the ideal tool profile. Selecting specific components and displaying the interaction assistance information for each, she investigated the data available to her at the interaction mechanism level. She then identified the differences in functional provision amongst the various components - periodically

visualising the interaction assistance information - and identified the differences in interaction mechanisms used.

After clearing the highlights and returning the visualisation to its initial state, the subject identified another of the top-level categories for which the tools were substantially different in terms of their greyscale representation. She again expanded that category by one level and examined the sub-categories for marked differences in greyscale - fully expanding the sub-category with greatest visual greyscale disparity. At the interaction mechanism level, she identified the components for which both tools provided the functionality, the interaction mechanisms used, where the interaction mechanisms differed, and after visualising the interaction assistance data across all components, examined the provision of interaction assistance. There then followed a period during which the subject highlighted negative cognitive demand modifiers, positive quality of feedback modifiers, and positive modifiers for miscellaneous comments. After applying each filter, she compared the components on the basis of the highlights, took notes about her observations, and cleared the existing highlights before applying the next filter. After contracting the expanded sub-category back to its root, she selected and fully expanded another of the noticeably different sub-categories and repeated the above process of examination for its constituent components.

The subject filtered the data on the basis of functionality matches with the ideal tool profile before using the highlights to identify differences in functionality between the two tools, and the components for which both tools provided the required functionality. Furthermore, she identified functionality that was missing from both tools, and differences in the interaction mechanisms used.

The subject then adopted a different strategy - after returning the visualisation to the top-level view she fully expanded two of the top-level categories. She identified differences in the functionality and interaction assistance provided by the two tools before clearing her highlights and again returning the visualisation to the top-level view.

Having previously selected top-level categories for which the greyscale values differed most significantly between the tools, she then selected and fully expanded a top-level category for which there was no distinguishable difference in the greyscale representation of the cells. She identified differences in the functionality provided and interaction mechanisms used before filtering the data on the basis of positive cognitive demand modifiers. Using the highlights as a guide, she identified differences in cognitive demand before examining differences in the provision of interaction assistance. She cleared the existing highlights before filtering out the positive quality of feedback modifiers, and

identifying differences between the tools on the basis of their associated quality of feedback. She then returned the visualisation to the initial top-level state.

Selecting another top-level category, she fully expanded it before identifying differences in the functionality provided, identifying the use of the required interaction mechanisms, and comparing and identifying differences in the provision of interaction assistance. The subject returned the visualisation to its initial state before independently applying filters for functionality matches with the ideal tool profile, and instances of use of the required interaction mechanisms. After fully expanding the entire data set, she identified functionality that was missing from both tools before examining the difference in cognitive demands between the tools by independently applying filters on the basis of positive cognitive demand modifiers and then negative cognitive demand modifiers.

This synopsis of the subject's activities illustrates that she did not adopt a noticeably consistent or regimented approach to comparing the data. Unlike previous subjects, she did not work globally and then more locally (or vice versa) but flitted between the various scopes available to her in an opportunistic manner. Similarly, she did not examine the attributes of the data in a set order - she mixed examination of all aspects of the data, not always comprehensively, and returned to certain attributes despite having filtered the data on completely different attributes.

Despite the apparent randomness of her approach at times, the following extract suggests that her approach nevertheless led the subject to feel confident that she had learned the tool sufficiently to consider it easy-to-use, and that she was confident that she had identified the important aspects of the tools:

[r] How you rate your confidence in your performance with SUIT?

[s] I think if it was faster I would have rated it higher…but I wasn't always sure whether I had got my input or not…

[r] …uh huh…

[s] …but that's not your fault…

[r] …that's not really your fault that's the system…

[s] …yeah…so…yeah I…I think I could use it quite easily now…

[r] …right…good…em…and how confident are you with the results you've obtained ?

[s] I think it was a very difficult one to do because these two tools are very alike…

[r] …uh huh…

[s] …um…

[r] …but are you quite happy that you've identified what you were looking to identify ?

[s] I think so yes…

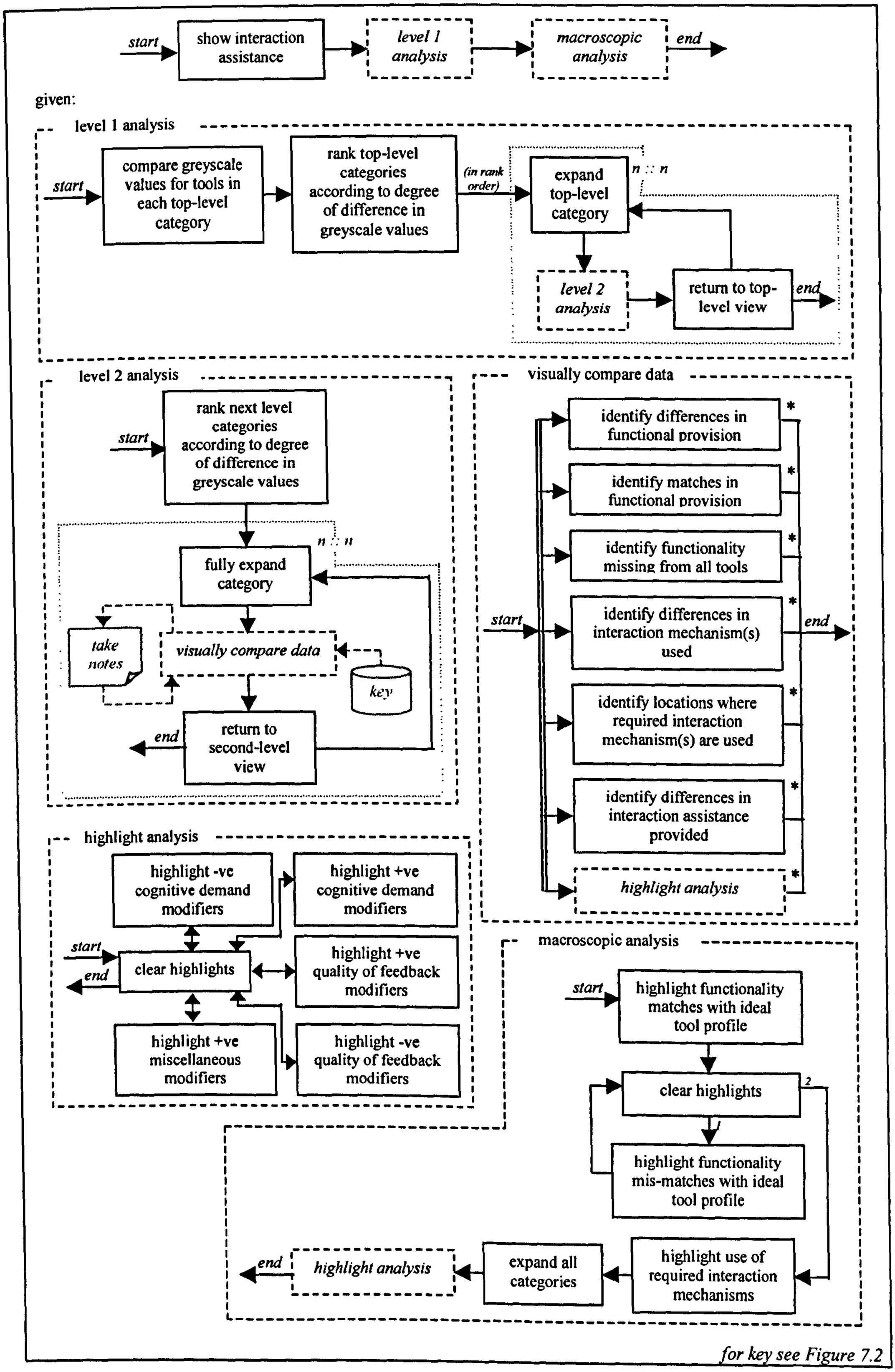


Figure 7.5 - *model of comparison and analysis strategy adopted by subject 4*

Although the general response time of the system (see section 7.3.3) reduced the subject's confidence in some of her actions, she recognised the root of this problem and indicated that it would not deter her from using it again. Furthermore, the environment has obviously been successful in terms of making her appreciate how similar the two tools actually are and that the issues upon which to base a decision were not going to be easily identified with a cursory look at the data.

This subject's use of the notepad was particularly interesting. Unlike other subjects who made minimal use of it to record the results of their analysis, this subject also used it to record a checklist of things she identified as requiring her attention - essentially an aide memoir during the course of her comparison task. Her 'to do' list is evidence that the environment needs to consider support for higher level task and information management aspects of UIDT data comparison and analysis. This is discussed further in chapter 8.

Figure 7.5 outlines a model of the subject's comparison and analysis strategy. Although every effort has been made to accurately represent the manner in which the subject tackled her task, it was sometimes necessary to abstract over her observed actions in order to generate a comprehensible model.

Study 5

Although this subject provided valuable feedback in terms of his critique of the visualisation environment, he failed to focus on, and therefore complete, the task he was asked to perform. Additionally, extraneous technical interruptions were encountered during this study[75] (see section 7.6.1). The content log for this subject does not therefore truly reflect real use of the environment. As such, although the activities of the subject are discussed, a strategy map has not been outlined on the grounds that the subject did not progress beyond cursory investigative activity[76]. Rather than actually perform his analysis task, the subject often discussed how he might approach the task. Often interesting, such discussion has therefore been included in the following synopsis of the subject's activities.

Initially, the subject spent some time examining the greyscale representation of the percentage functionality provision in the top-level view. He noted a relative value for each cell (high - medium - low) and the degree to which the tools differed in this respect for each category. As with the exploratory actions of many of the subjects, he fully

[75] To the extent that they prevented the subject continuing interaction with the environment.

[76] Activities that have been excluded from the strategy maps of the other subjects.

expanded all categories to get an overview of the data set before restoring the visualisation to its initial state.

For each of the top-level categories in turn, the subject expanded the category by one level, gave the associated sub-categories a cursory examination, and then returned the visualisation to the top-level view. After viewing each of the top-level categories in this manner, the subject expanded all of the top-level categories by one level. For each sub-category, he identified differences in the greyscale value represented in each cell, before selecting all matches between the tools. This returned an empty set - since the ideal tool profile is treated as a tool in its own right there were no components for which both tools exactly matched the ideal tool profile. He therefore selected all mis-matches between the tools and was presented with the entire data set. After a brief examination of the components, he selected all categories for which there was no registered functionality before returning the visualisation to its original state.

The subject fully expanded the first of the top-level categories and identified functionality that was missing from both tools, differences in the functionality provided by the two tools, the use of the required interaction mechanisms, and differences in the use of interaction mechanisms. During this process, he commented that he was tending to ignore where the two tools were the same, and was concentrating on where there was some sort of difference between them. He returned the visualisation to the top-level view before fully expanding another of the top-level categories. This time he focussed on the functionality that was missing from both tools and on differences in the interaction mechanisms used. This was the limit of the subject's progress during the interactive session.

The subject did not progress further than a basic initial exploration of the data set and the effects of some of the filters and selection operations. However, when he had fully expanded one of the top-level categories and was comparing the data at the level of the interaction mechanism representation, the subject succinctly identified the three principal circumstances to consider when examining the data set at that level: (1) the areas where neither tool provides the functionality - which he suggested were not worthy of further investigation; (2) the areas where one tool does not provide the functionality whereas the other tool does - in which case he suggested that the latter tool should be considered superior even if it does not use the correct interaction mechanisms and therefore such areas did not require further investigation; and (3) areas where the tools both provide the functionality but use different interaction mechanisms - such areas he deemed to be the most important in terms of further investigation. Furthermore, he suggested that a comparison would consist of finding all functionality for which the tools differed and then investigating which tool the difference favoured. It is clear that the other subjects were

also using this conceptual justification but none of them summarised the situation as concisely.

Since his suggested logic showed the potential to develop into a systematic and comprehensive strategy for comparison and analysis, it is unfortunate that this subject was not able or minded to complete the comparison and analysis task. However, he was able to provide useful and informative feedback both in terms of conceptual means of tackling the task, and usability defects within the environment itself (see section 7.5.1).

Study 6

This subject was the most experienced software developer with extensive knowledge of CASE tools. His expertise in CASE tools led to an appreciation of the complexity of UIDTs that was reflected in his approach to the comparison and analysis task whereby, after an initial investigative foray into the lowest level of detail, he tackled his analysis from a top-down perspective.

He fully expanded a top-level category to identify functionality that was missing from both tools, differences in the functionality provided, differences in the use of interaction mechanisms, and in the use of the required interaction mechanisms before restoring the visualisation to its original state with the declaration that he was getting too 'bogged down' in detail and that he intended to get a 'broader view' of the data.

Deciding to adopt a 'negative stance', the subject filtered the data on the basis of functionality mis-matches with the ideal tool profile. After identifying differences in functionality match on the basis of the highlights within the scope of the top-level categories, the subject selected one of the top-level categories and fully expanded it. He identified differences in functionality and interaction mechanisms (including the use of the required interaction mechanisms) before returning the visualisation to the top-level view.

After identifying the percentage functionality provision for the two tools within the next top-level category, he expanded that category by one level of abstraction. He examined the sub-categories' cells to identify sub-categories for which both tools provided either 0% or 100% of the required functionality. On the grounds that 0% or 100% functionality provision meant that, at least in terms of functionality, the tools were identical, he expressed his intention to focus on the remaining sub-categories - that is, the ones with grey cells. He selected one of the sub-categories for which one or both of the cells were grey and expanded it down one further level. Momentarily diverting his attention to the level above, he decided to hide the 0% and 100% sub-categories to simplify the visualisation. He identified sub-categories of his current expansion that were different in

terms of provided functionality before contracting the visualisation by one level - that is, to his previous series of sub-categories. He then repeated this process for another of the initially identified 'grey' cell sub-categories. Having completed the comparison of its sub-categories and the categories at the first level of abstraction, the subject returned the visualisation to the top-level view. After hiding the completed top-level category, he repeated the whole of the above process for another two of the top-level categories.

On the occasion of one expansion of a 'grey' sub-category, he decided to fully expand it (rather than just expand it by one level). He identified differences in the functionality provided by each tool and in their use of the required interaction mechanisms. He also identified functionality that was missing from both tools. It would appear that the subject broke from his routine on this occasion out of curiosity rather than necessity since he concluded that he did not need to view the data at the lowest level in order to make his desired observations. He returned the visualisation to one level of expansion for the current top-level category and resumed his previous systematic top-down approach.

Another momentary break from routine saw the subject first highlight functionality matches with the ideal tool profile and then functionality mis-matches with the ideal tool profile. He took some written notes regarding his observation of the highlights before resuming his established strategy.

Having completed his top-down analysis of each top-level category, the subject restored the visualisation (which clears existing highlights). He then adopted a different strategy for his remaining comparison. In particular, he opted for a more macroscopic comparison approach whereby he first highlighted the use of one of the required interaction mechanisms before globally expanding all categories and, aided by the highlights, identifying differences in the use of the required interaction mechanism.

Returning the visualisation to the top-level view, in turn he expanded each of the top-level categories by one level and examined the enclosed highlights. He restored the visualisation and filtered the data on the basis of interaction mechanism mis-matches with the ideal tool profile. After expanding two of the top-level categories by one level of abstraction, he resorted to a global expansion of all categories. Guided by the highlights, he identified differences in the functionality provision for components where the required interaction mechanisms were used.

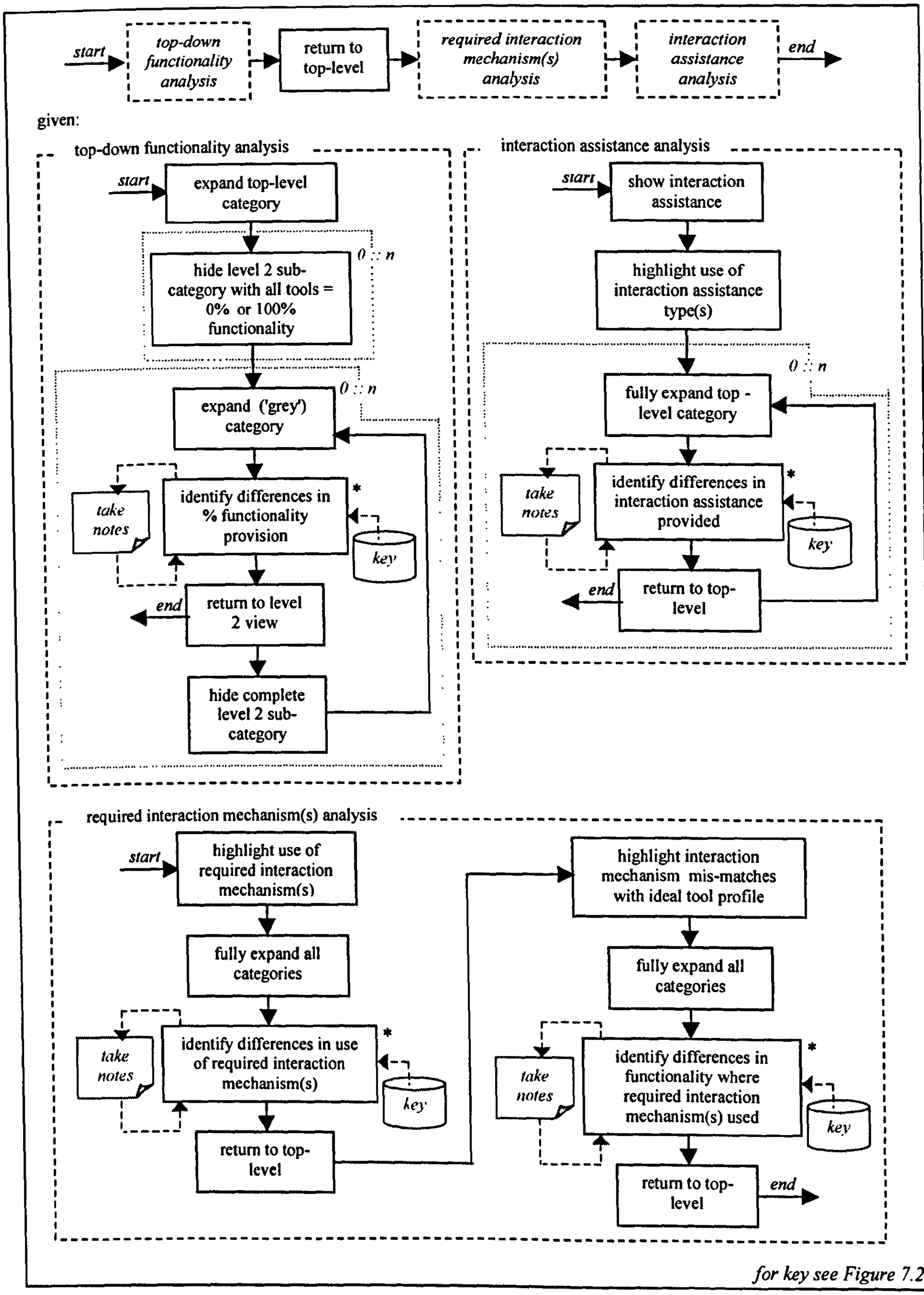


Figure 7.6 - *model of comparison and analysis strategy adopted by subject 6*

He restored the visualisation before filtering the data on the basis of interaction mechanism matches with the ideal tool profile and then, after clearing the highlights, according to a specific interaction assistance type. After fully expanding the first top-level category, he

visualised interaction assistance data and identified differences on this basis between the two tools. After returning the visualisation to the top-level view he repeated this examination for the next top-level category. At this point, the subject declared that he felt he had extracted as much useful information as possible from the visualisation and concluded his comparison and analysis task.

Before commencing his task, he outlined his initial thoughts regarding the strategy he was going to adopt:

[r] Do you have any strategy in mind as to how you are going to compare these tools ?

[s] Well yes, I was thinking about that. I'm not sure whether to start with the…the weakest feature or the strongest feature…I think I'm going to start with the weakest to try and identify the weakest areas first. So I'm going to work out where the weakest areas of match are and then see if I get anywhere that way - if I don't, I'll switch and I'll start working with the strongest areas. [pointing to the visualisation - and specifically the second of the top-level categories] So there's obviously weakness in both tools in one area; [pointing to the visualisation - specifically the darkest of the greyscale cells in the top-level categories] there's a strength down here…em…obviously in the Delphi tool…em…actually I think I'm going to change my mind and start with the strongest because that [again pointing to the darkest of the greyscale cells] seems to be the thing that stands out most in the whole…in the whole array, the thing that seems to stand out most is that 'UI Features' box there for Delphi [again pointing to the same cell] so I think I'm going to start by expanding that out and having a look to see what the…the situation is there.

From the outset the subject decided to investigate the data set from what he later called the 'negative stance' but demonstrated flexibility and willingness to alter his strategy if it was found to be inappropriate once he had examined the data.

Figure 7.6 maps the strategy adopted by the subject and demonstrates the systematic approach he followed when comparing and analysing his data set.

The subject was asked to comment as to whether the above strategy was appropriate and whether he felt the system had enabled him to follow the strategy he wanted to follow. The following extract is taken from the beginning of the interview session:

[r] Before you started, you mentioned that you had a sort of…quite a vague…strategy in mind. Do you think that you were able to follow that strategy when you started using the tool ?

[s] em…I think the general idea of looking for the…the…the biggest mis-matches was probably right. I didn't…in fact…follow the strategy properly…eh…I…I think I wasted

probably the first five minutes or so trying to work at too fine a level of detail…

[r] …uh huh…

[s] …and…if I was going to start again…if you gave me another session with two different tools or…or…you know a different…eh…task…

[r] …uh huh…

[s] …I think I'd have a much better idea of how to attack the problem. I…I would…I would take this top-down approach which I eventually…

[r] …uh huh…

[s] …reached because in fact, a lot of the gross…eh…either differences or non-differences - matches - show up almost immediately at one level down - you don't need to go to three levels down to see the differences…

[r] …uh huh…

[s] …and so, I think it's much more natural to do it in a…in a top-down manner…so you start with each category, look at the…expand by one level, look for the real differences, if there are any then go a bit further down if you want to…

[r] …uh huh…

[s] … so I think my…perhaps my global strategy was right but my…my first attempt to implement it was actually wrong…

The subject was able to distinguish the mechanisms available to him and which he used, from the more abstract strategy he was trying to follow. He was able to identify the fact that his technique was not working initially and so adopted a different technique that he found more effective. Furthermore, the subject suggests that after one interactive session with the environment he felt sufficiently comfortable with it that he considered himself far better equipped to approach a similar problem in a more efficient manner.

When asked about the level of his confidence in terms of his performance with the environment and the results he obtained, he commented that although he thought he had started off using the environment rather 'haphazardly', he felt that by the end of the session he was using it reasonably well. However, regarding his confidence in the observations he had made and the conclusions he had drawn, the subject was less confident on the grounds that component priority information that would have made his decision more definite was missing from the visualisation. Although the subject's confidence was compromised by the absence of priority information, it was not lacking on the grounds that he felt that he had failed to identify the differences between the tools, but rather that, having identified these differences he did not know the relative priority to assign to them in order to conclude a relevant recommendation. This suggests that the environment is effective at assisting an evaluator to identify the differences between UIDTs but needs to consider assistance in prioritisation of these differences (see chapter 8).

7.6 Experimental Validity

Having presented the results, it is important to consider the validity of the qualitative study regarding the manner of execution and the way in which the results have been analysed - not only to qualify or temper the resulting claims, but also to highlight the lessons that may be learned from mistakes made during this study.

7.6.1 Technical Problems Encountered During Study Execution and its Impact on the Study Results

Section 7.3.3 outlined technical restrictions imposed upon the execution of the study. In general, the impact of the retarded operational speed on the usability of the system was restricted to feedback delays – more substantial usability defects as a result of the technical restrictions were discussed in section 7.5.1. Nevertheless, the impact of the operational speed is likely to have influenced the manner in which the subjects reacted to, and therefore interacted with, the environment. It is anticipated that fewer usability defects may have been identified had the system response been less handicapped.

During the course of the interactive session of the fifth study, it became apparent that the departmental system administrators were carrying out remote system updates on the departmental network. These had not been forewarned, and as a result interrupted and handicapped the interactive session of the study. Had the administrative operations been known in advance, the study would obviously not have been scheduled when it was. However, this experience highlights an important lesson - that is, to isolate the evaluation equipment if at all possible.

7.6.2 The Quality of the Results

The focus of this investigation was to identify, and thereby generate an initial corpus of qualitative information regarding the strategies adopted by subjects when asked to compare and analyse a complex data set using the SUIT visualisation environment.

Lindgaard warns that although data is easy to collect during protocol analysis, the value of the data is directly attributable to the skill of the evaluator in terms of the evaluator's ability to extract interesting or salient issues (Lindgaard, 1994). Given that this was the first series of evaluative studies of the environment, the data collected is unlikely to be comprehensive in terms of having elicited all possible information. The results from this series of studies should therefore be used to inform future studies of the environment once the observations made here have been addressed (see chapter 8).

Since the subjects were being observed when performing their tasks and were unavoidably aware of the researcher's agenda, the Hawthorne Effect is likely to have influenced the results obtained. Most of the subjects had considerable research and development experience and so there is no reason to believe that the Hawthorne Effect will have influenced their adopted *strategies*. It may, however, have influenced the *thoroughness* with which they completed their comparison and analysis tasks. Although the Hawthorne Effect may have influenced the subjects' judgement of their confidence, since their ratings were not overly high it is likely that they would only be marginally reduced if the subjects were not monitored.

Often subjects' first use of a software application is not representative of how they will use it as their level of experience increases. In the case of the SUIT data visualisation environment, however, it is anticipated that use will be casual and infrequent and so expert use is not relevant. Lacking expertise, the subjects' use of the SUIT data visualisation environment can therefore be considered representative of its typical use.

It was not the intention of the study to produce quantitative data to which statistical significance could be attributed, and so neither the strategy analysis nor the usability defect analysis should be considered formal analyses of their respective topics. Steps were taken, however, to reduce researcher bias and ensure accuracy of representation. In particular, the content logs generated for each interactive session were validated by an independent assessor (see section 7.5) to confirm that they were an accurate representation of the videotaped activities and discussions. Furthermore, the process by which the usability defect analysis was conducted is well documented, uses a clearly defined set of categories, and is repeatable by an independent assessor.

Consider now the comparison strategy analysis. Given the nature of the investigation and lack of randomness of the subject population, no statistical significance is being attributed to the results presented in this chapter - the strategies identified are only attributable to the subjects who were part of the study and no attempt is being made to generalise the results beyond the subject population.

The size and potential unrepresentativeness of the subject population means that the identified strategies cannot be considered conclusive or exhaustive. Although there is no evidence to suggest that the identified strategies are representative – there might be numerous others – the results of this study provide evidence that there are *several* different practical approaches to comparing and analysing data using the SUIT visualisation environment. The intention of developing the independent strategic models (and the collective model discussed in section 7.7.2) was to illustrate how evaluators *have been shown* to tackle the data comparison and analysis stages of a SUIT selection process to

provide at least a basic level of guidance for future evaluators. The strategy maps are not being presented as the only manner in which UIDT data comparison and analysis can be structured. During the interview sessions a number of the subjects commented that, having used the environment to complete a complex task, they considered themselves better equipped to tackle future evaluations in a more efficient and structured manner - they had successfully learned to use the environment through extensive use of its visualisation and associated operations. It is therefore proposed that the strategy maps generated as a result of this study will serve as an optional introduction to the use of the visualisation environment such that first time use of the environment for future evaluators is perhaps more efficient and less overwhelming than it would be with no such guidance. It is fully anticipated that after initial guided use of the environment, future evaluators will develop their own preferred strategy.

Consider, finally, the usability defect analysis. As with the strategy analysis, the usability defects identified in the study are not attributed any statistical significance. Similarly, they are not considered conclusive or exhaustive - most require further investigation and it is fully anticipated that others will be uncovered during future evaluation studies. Furthermore, they are likely to have been influenced by researcher bias given the nature and phrasing of the questions asked. That said, subject feedback regarding the strengths and weaknesses of the environment that was elicited during the interview sessions highlights a number of interesting issues and defects to be addressed in future development of the visualisation environment (see chapter 8).

7.7 Discussion

The preceding discussion in this chapter has presented, in detail, the findings of the qualitative study. The following concluding sections start with a brief summary discussion of the identified usability issues and defects. The comparison and analysis strategies adopted by the various subjects are then discussed and the individual strategy maps are combined into a comprehensive flexible strategy map covering the majority of the noted activities and their sequencing.

7.7.1 Usability Defects

Section 7.5.1 discussed the observed and categorised usability defects. Although each subject identified defects that were specific to their own study, the discussion highlighted defects that were common to several subjects.

Not least of these issues were those centred around the contraction operations. Although the contraction operations were implemented to affect what was considered logical behaviour, they have not been shown to be successful in this respect and as such will need to be investigated in order to improve their learnability.

Many subjects commented on the design and layout of the environment. These comments were dominated by issues surrounding component (column) labelling. In particular, subjects found the orientation of the text on the component labels hindered their ability to read them. The point raised by the pilot subject - that is, his familiarity with reading from left-to-right and top-to-bottom but not bottom-to-top - suggests that the difficulties experienced by the subjects were perhaps not just a factor of the orientation of the labels, but also a result of the reading direction.

Several subjects suggested that the visualisation should be turned on its side - essentially transposed into a tree structure. Although they suggest this for reasons of label orientation and consistency with the paper version of SUIT tables, this raises an interesting question. In chapter 6, the discussion of the design of the visual data representation for the visualisation environment introduced Gilmore's findings regarding the salience of data in tables used for data comparison (Gilmore, 1991). In particular, he discovered that where the salient issue in such tables (that is, the principal focus of the comparison task) is represented by the rows, there is a natural inherent strategy for comparison of the data set. Conversely, he found that subjects are at a loss as to how to compare the data in tables where this mapping is not present (Gilmore, 1991). For this reason, the SUIT data visualisation environment selected the rows to represent the UIDTs - that is, the goal focus of the comparison task. Hence, the subjects' request to re-orientate the rows and columns in the SUIT visualisation environment apparently contradicts Gilmore's findings[77] - albeit each subject appeared to benefit from, and thereby confirm, the presence of natural strategy. This is a significant issue and should be addressed during further research and work on the environment (see chapter 8).

Subjects expressed concern regarding the columns in the visualisation table - specifically, that the narrow width of the columns made accurate mouse clicks difficult and inhibited identification of the first interaction mechanism colour swatch when a cell was highlighted. Subjects also commented on the absence of grouping boundaries when viewing lower level components. When designing the visualisation it was realised that a large amount of data needed to be displayed in a restricted space and hence the columns were designed to be as narrow as possible. Although consideration had been given to the implementation of

[77] It should be noted that the paper-based framework tables do not contravene Gilmore's principle of salience since: (1) the data within each table is not being compared; and (2) the salient issue in *each table* is the component.

category boundaries and groupings during the initial design stages for the environment, they were abandoned (at least from the first prototype) on the grounds of complexity. Basically, since columns can be visually relocated outwith the conceptual bounds of their category, it was thought that the representation of groupings would become so overly complex that it would deter the user from successful navigation around the system. However, the response from the subjects in the various studies would suggest that some form of category boundary is required and this should therefore be the focus of further work on the visualisation design.

Identified terminological confusion - especially concerning interpretation of the terms 'filter', 'highlight' and 'select' - should be addressed as a matter of priority in future development of the environment. Subjects were also found to lack confidence in, and a means of interpreting, the term 'other interaction mechanism'. Had the subjects developed the ideal tool profile and/or tool profiles they would have been exposed to the context in which the term 'other' has meaning, and it is anticipated that the use of this terminology in the visualisation environment would not therefore have been identified as a usability defect. However, it cannot be assumed that all users will necessarily have been involved with the evaluation process at that stage, and so the use of this and similar terminology should be examined further.

Future development of the environment will have to tackle some of the modality issues highlighted by a number of subjects during the studies. In particular, the lack of feedback when a selection or filter returns an empty set will have to be addressed in order to increase user confidence in the correctness of their actions, and the difference between highlighting and showing interaction assistance will have to be clarified - perhaps by adopting the pilot subject's suggestion of showing rather than hiding interaction assistance data by default.

While observation of the subjects identified their adopted data comparison and analysis strategies, it also highlighted missing functionality that prevented the subjects interacting with the data set as they would have liked. This mismatch with the users' tasks illustrates the functionality that is required within the environment if it is to meet evaluator needs. The requests for additional functionality are summarised below:

1. the ability to select and/or mark multiple columns and thereafter apply operations to that selection;
2. increased sophistication in terms of note taking facilities - including the option to hide the notepad, to transpose the component hierarchy information into a list structure within the notepad, and the associated facility to semi-automatically generate reports on the basis of the notepad and associated data visualisation;

3. the representation and manipulation of priority information for components in the visualisation - perhaps via the ability to toggle between a priority-based and category-based view of the data;
4. increased control over filters - namely, greater control over the join mechanism between applied filters and the ability to independently apply non-joined filters simultaneously in order to compare their results, as well as the provision of a row-based numerical tally to indicate the number of highlights generated as a result of applying a filter;
5. increased sophistication in terms of contraction and expansion of component categories - in particular, the ability to globally expand or contract all components by one level at a time;
6. addition of options to filter on the basis of the existence of modifier comments both generically and topic-specifically;
7. the facility to decide whether or not to include the ideal tool profile in selection operations;
8. representation of percentage functionality intersection between tools - and similarly the ability to identify the actual percentage value without having to refer to the data sheet;
9. the ability to switch the tool tip facility on and off;
10. the ability to globally resize all columns at once;
11. the provision of both help and undo facilities.

Although some of the above facilities are relatively easily remedied via the extension of existing functionality, others will require more substantial research in order to best identify the means to address the functionality. In particular, the issue of representing priority information will require considerable thought not least in terms of how to represent it, but also in terms of how to record the priority information in the first place. As was discussed in chapter 2, the introduction of complex means by which to record and represent prioritisation of components was found to over-complicate and burden evaluators. This study has highlighted the demand for such information. Hence it will have to be considered in order to match the needs of the users, but should be treated carefully and with caution to avoid the problems faced by the developers of other UIDT evaluation facilities. Similarly, it is unclear whether the demand for the representation of the intersection of percentage functionality is indeed a true requirement or has instead arisen as a consequence of missing analytical tools (for example, the inability to exclude the ideal tool profile from selection operations) and this should be further investigated before adapting the basic graphical representation.

Although the identified usability issues may have affected the manner in which the subjects interacted with the visualisation environment - an interesting issue for further

investigation after addressing the defects - none prevented the subjects successfully completing their data comparison and analysis task.

7.7.2 Comparison Processes

Section 7.5.2 presented a series of five strategy maps modelling the approach taken by each of five subjects to UIDT data comparison and analysis using the SUIT visualisation environment. Examination of the corpus of mapped strategies highlights a number of summary points, discussed below.

Each subject spent a short initial period investigating the data set in order to 'get a feel' for the most appropriate strategy given the nature of the data. It was normally during this period, as a result of their exploratory analysis, that any predetermined strategies were either confirmed or altered.

With the exception of subject 6, each of the subjects sub-divided their comparison and analysis into a period of *global* analysis and a period of *category-by-category* analysis. Although the subjects' strategies for each type of analysis differed in terms of detail, there were a number of common points.

During their category-by-category analysis, a number of the subjects ranked the top-level categories according to the degree of difference in the greyscale values across the different tools. Some thereafter started with the categories with least difference in the hope to be able to eliminate categories from consideration, and others focussed primarily on the categories with the greatest degree of difference considering them to be the categories in which the decision criteria would be found. Whether or not the ranking activity was performed, the subjects selected each category in turn and (normally) expanded the category to the lowest level of detail. After examining the data, they all returned the visualisation to the top-level category view.

Without exception, the subjects focussed their comparison and analysis on the differences between the tools - as stated by subject 6, they were adopting the 'negative stance'. Given that the two tools are in fact very similar in terms of functionality provided and interaction mechanisms used, this proved to be an effective strategy. During his interview session, one subject questioned whether this approach would be the most appropriate if the tools being considered were radically different in most respects. Although this would require further investigation, it is likely that subjects would still adopt the negative approach since it is the differences rather than the similarities that set tools apart and enable an evaluator to select one over the others.

In general, the subjects relied most heavily on 'manual' comparison of the data components at the lowest level of detail when conducting the category-by-category analysis - this process was commonly referred to by the subjects as 'eyeballing' the data. Little use was made of filters in this respect. Although the reason for this is not known, the highlighted usability defects regarding confusion surrounding the semantics of some of the filter options may have been a contributory factor. However, even when filters were applied to the data set, the subjects often forgot about them and performed manual comparison without taking advantage of the highlights presented to them.

With the exception of subject 6, who focussed his comparison and analysis strategy on the intermediate levels of abstraction, the remaining subjects made no real use of these intermediate levels. In general they worked from the top-level to the bottom and back up to the top without considering the data abstractions in between. The usability defect analysis did identify a mismatch with the subjects' tasks in this respect given that a number of them requested the ability to globally contract every component/category by one level at a time and vice versa, to globally expand all categories by one level at a time. Had this functionality been included in the visualisation environment, the use of intermediate levels of abstraction may have been greater. However, the impact of this - as with many of the issues identified during the usability defect analysis - will require further investigation in light of future changes to the visualisation environment (see chapter 8).

Without exception, the subjects rated the importance of functional provision over that of the interaction mechanisms used. In general, they suggested that it was more important to have the functionality than it was to worry about the interaction mechanisms employed to facilitate that functionality. Hence, if one tool was found to provide functionality that was missing from other tools, the former was automatically rated more favourably.

After functionality, in order of priority, the subjects placed the interaction mechanisms used by the various tools. During the course of each interactive session it was interesting to observe that, had the ideal tool profile not provided the information concerning interaction mechanisms, most subjects would have intuitively identified the mechanism with which they were most familiar or favoured (in the case of the subject population, this tended to be Programming Language). This is illustrated in the following quotation taken from one of the interactive session records (the subject is answering the researcher's question as to whether he thought that in the absence of the ideal tool profile, he felt he might have been inclined to compare the tools on the basis of his own preferences):

```
[s] yes…definitely, because I think everybody, although they
    would like to take into account how other people see things
    and how other people like to work, I think initially - more
    than initially actually - if you're looking at some
    project, some problem, and you…you find you approach it
```

> from what you know and if what you know is programming language rather than some of the other interaction mechanisms, then that would be your baseline - that would be where you were judging them from...

The ideal tool profile therefore successfully maintained subjects' focus on the required interaction mechanisms. The subjects placed interaction assistance provision after interaction mechanisms in their prioritisation of the data, and assigned the issues of cognitive demand and quality of feedback equal weighting. It is interesting to observe that, with the exception of one subject, miscellaneous comments were ignored.

Although most subjects compared the tool data according to the prioritisation order discussed above, some performed all the comparisons at the same time (that is, during a single full expansion of a top-level category) whilst others repeated the category-by-category expansion for each of functionality provision, interaction mechanism use, and then the remaining ancillary data.

For the subjects who performed global analysis after category-by-category analysis, the former appeared to constitute a confirmation exercise. That is, they had in general identified the points upon which to base their recommendation but used the global analysis to 'double check' their observations. In some cases, it was at this point that subjects compared the data set on the basis of cognitive demands and quality of feedback to determine whether there was any other dimension, over and above their functionality and interaction mechanism observations, upon which to base their recommendation.

Only one of the subjects exploited the snapshot facility. It is conjectured that this is a direct result of the speed at which the environment was running - the subjects did not want to slow it down further - and as such is not a true reflection on the use of this facility. Furthermore, given that their tasks were for evaluation purposes, the subjects were perhaps less inclined to make extensive records of their observations since they would have no need of them after the session closed.

Although some of the subjects intimated desire for greater control over the manner in which the filters could be combined in the environment, very rarely did the subjects *intentionally* combine them. Instead, they tended to work methodically through their required filters, clearing the effects of one before applying the next. The semantics of the filters were shown to have made the subjects wary of them, and hence this may be further reflected in their approach to filter combination.

Given the diversity in the detail of the manner in which the subjects approached their evaluation tasks, it was not possible to combine the various strategy maps into a single comprehensive strategy map that included every activity illustrated in each of the

individual maps. However, a single strategy map has been developed that, via flexibility of choice, captures the majority of the approaches outlined. It was infeasible to include some of the unique (and perhaps more sophisticated) activities performed by subject 6 and the slight twist on category-by-category expansion shown by subject 1. That said, given that the strategy map is only a proposed means of initial guidance for future evaluators, they can introduce additional activities and/or abandon the strategy map altogether.

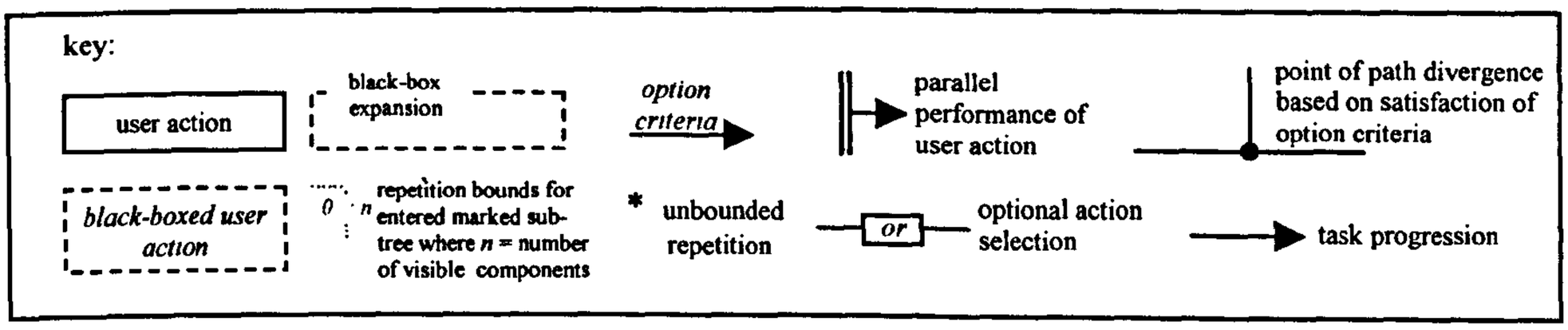


Figure 7.7 - *key for strategy map outlined in Figure 7.8*

Figure 7.7 shows the key for the collective strategy map outlined in Figure 7.8 which extends over the following four pages.

The strategy outline is a comprehensive summary and combination of the various data comparison and analysis strategies adopted by the different subjects. It is proposed as an *optional* and *flexible* guideline for future evaluators using the SUIT data visualisation environment such that if they choose to follow it, they may benefit from the strategies attempted by previous evaluators. It is therefore principally intended as a means by which to introduce future evaluators to the use of the environment in a way that reflects real practical use of the environment.

It should be noted that the strategy map is not promoted as conclusive nor exhaustive and is only applicable for project-specific user UIDT evaluations. No claims are being made as to the effectiveness of the strategies other than that which is demonstrated through the participating subjects' responses.

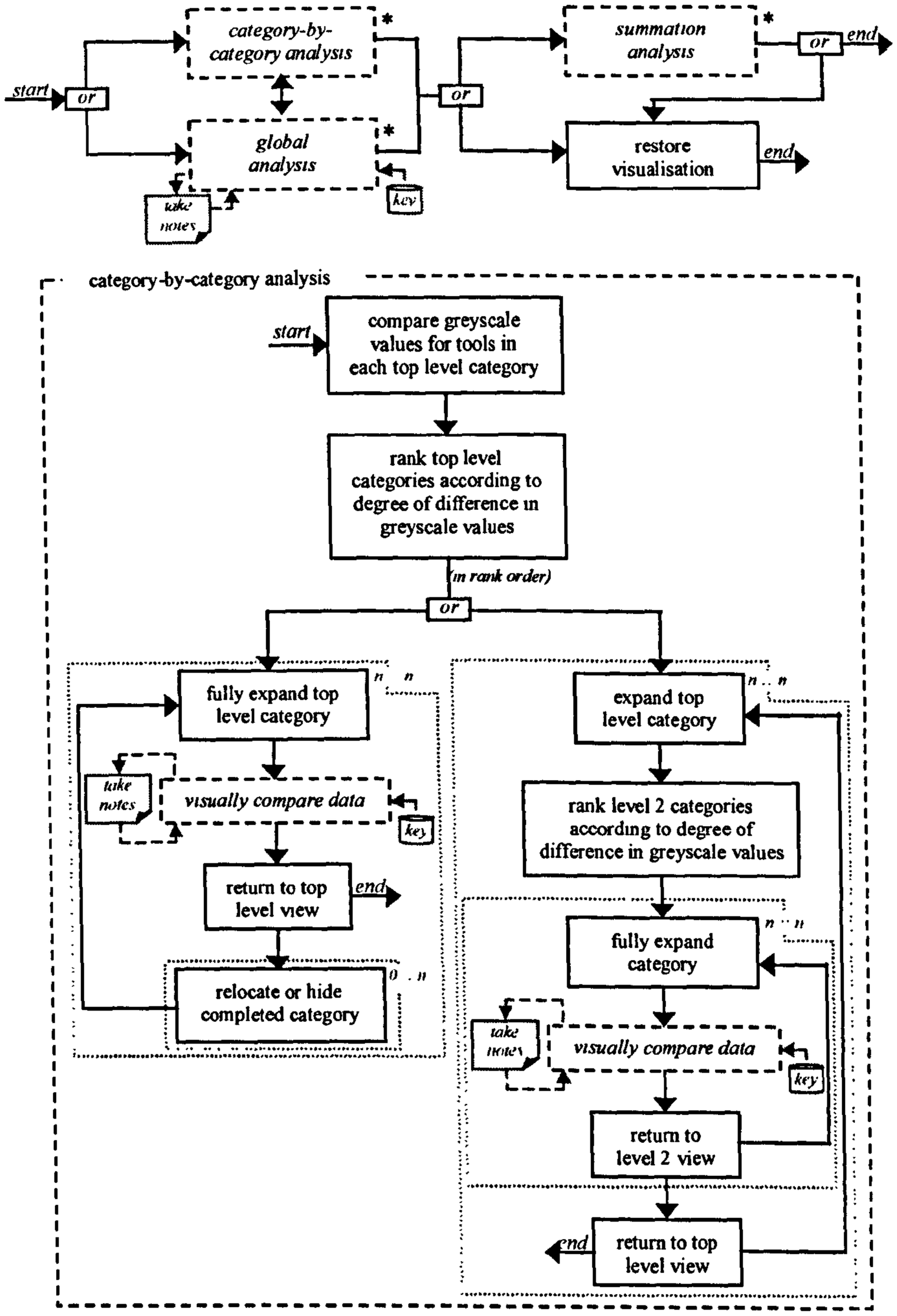


Figure 7.8 - *combined strategy map for project-specific UIDT data comparison using SUIT*

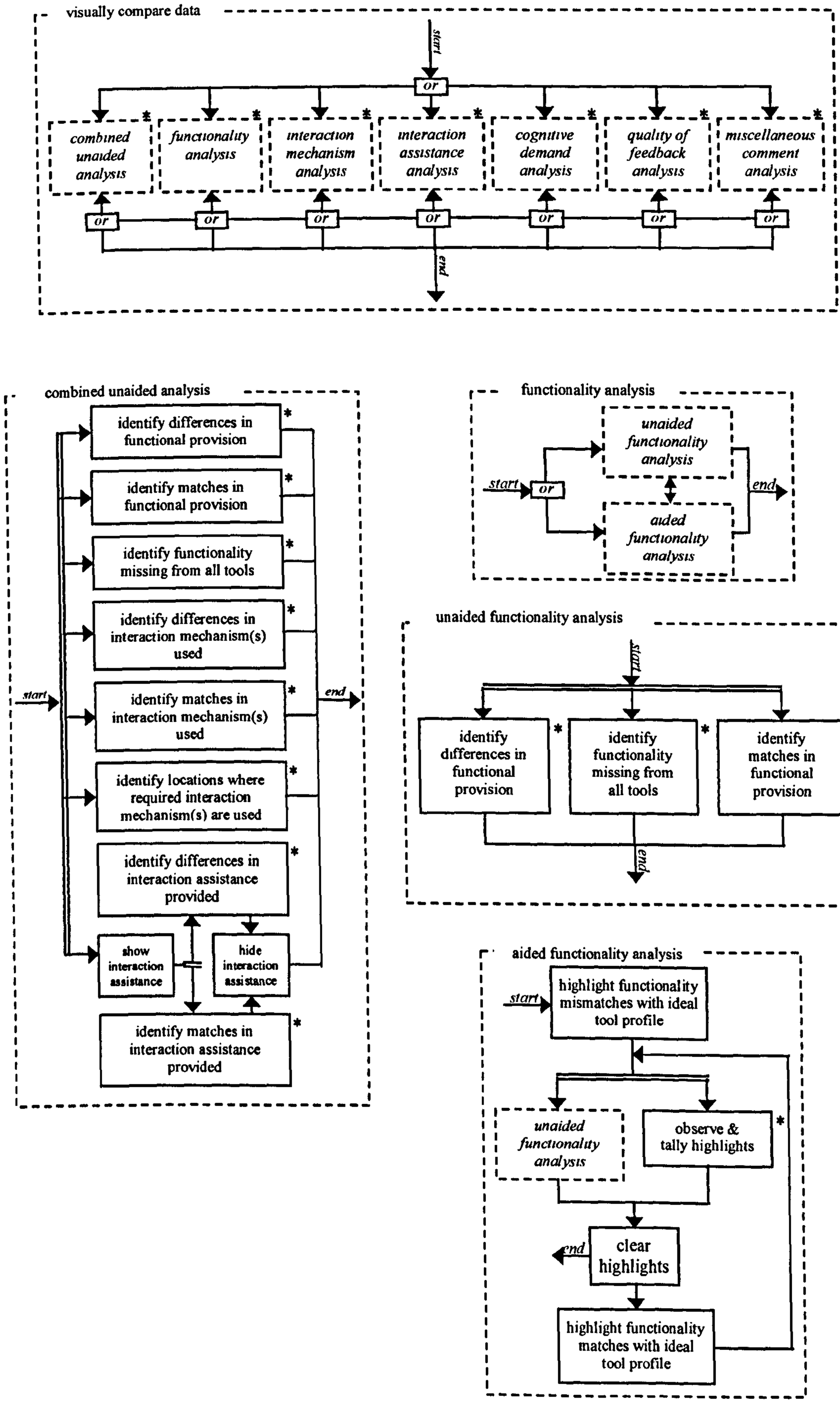


Figure 7.8 contd. - *combined strategy map for project-specific UIDT data comparison using SUIT*

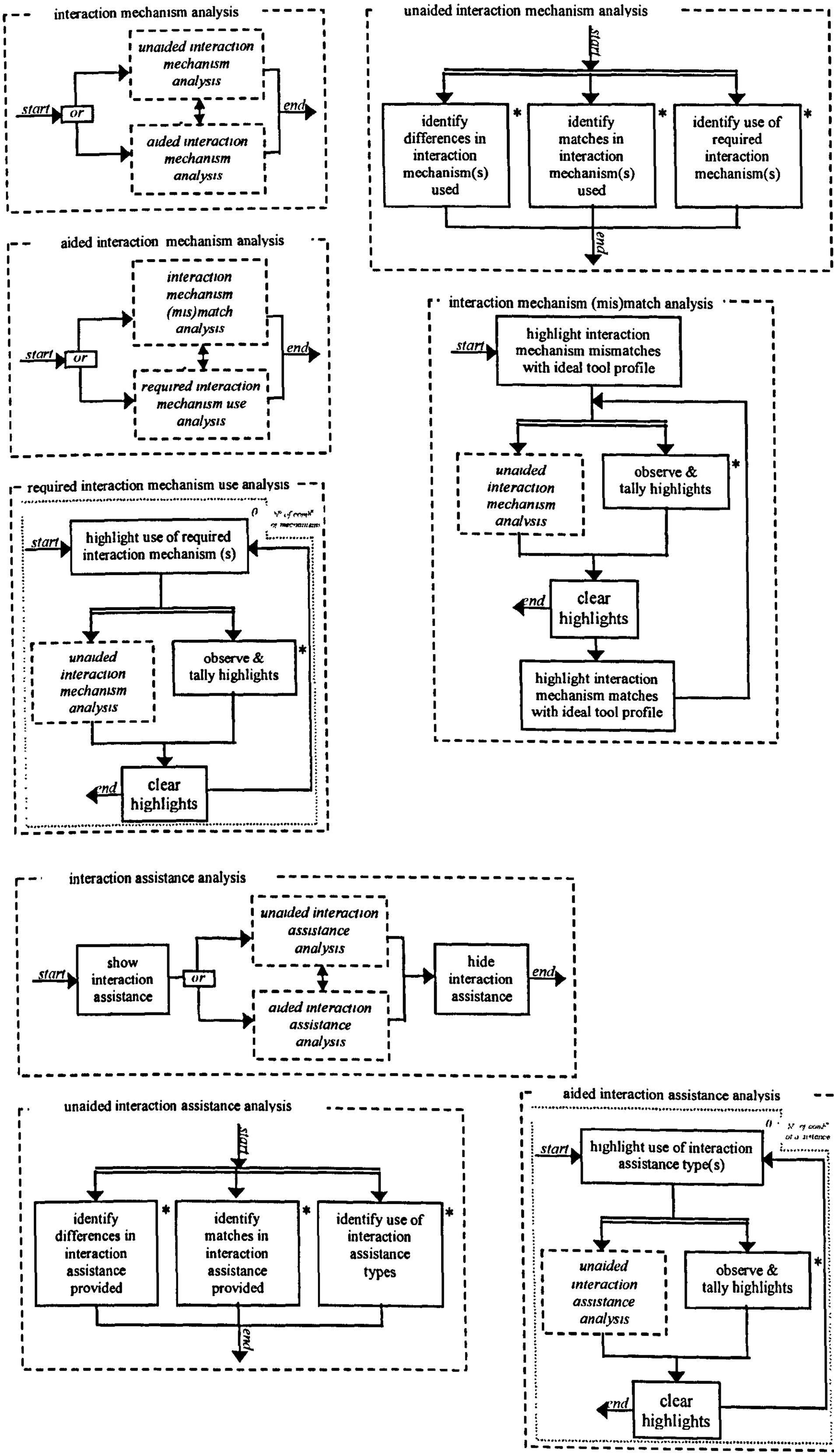


Figure 7.8 contd. - *combined strategy map for project-specific UIDT data comparison using SUIT*

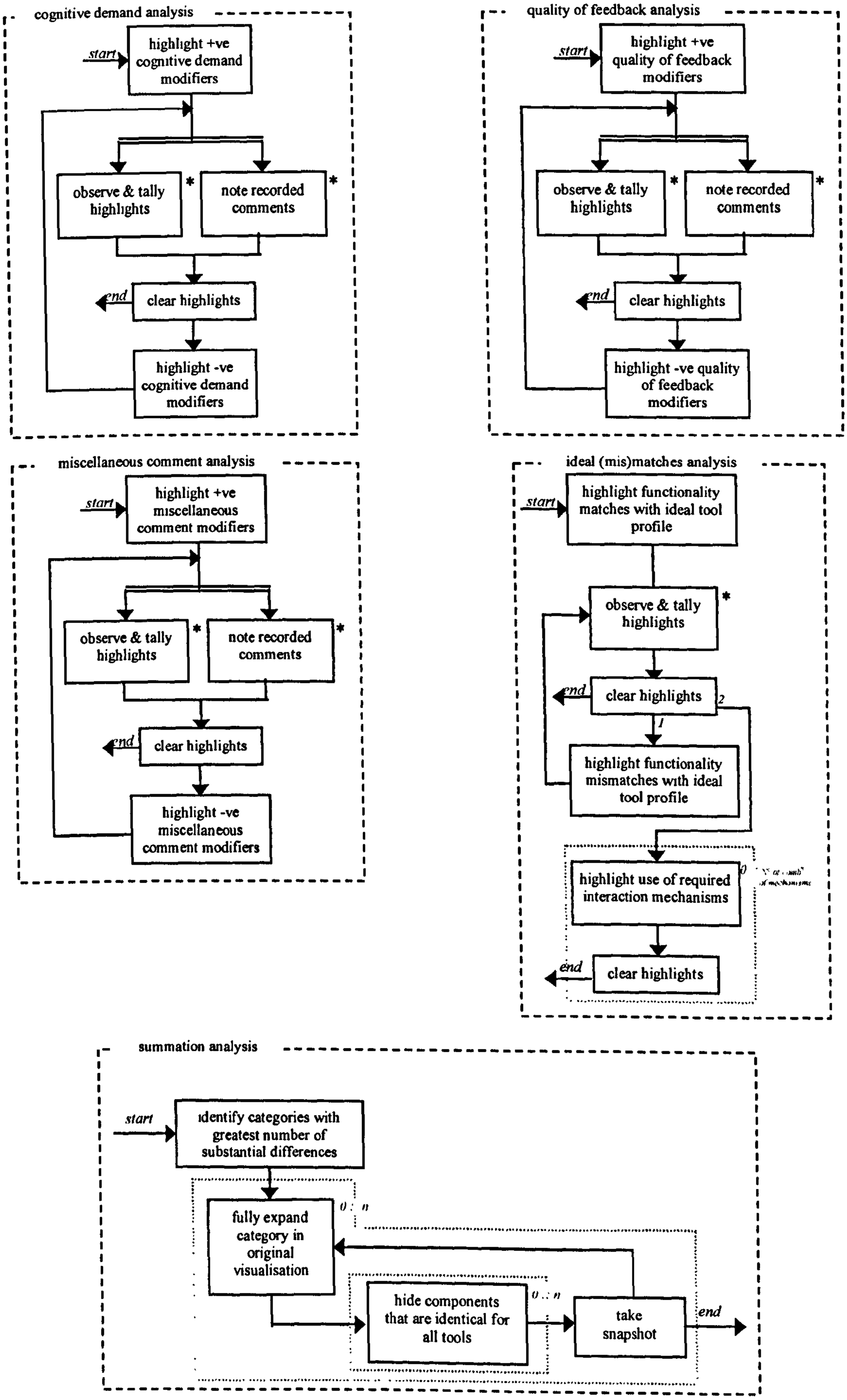


Figure 7.8 contd. - *combined strategy map for project-specific UIDT data comparison using SUIT*

7.7.3 Combining the Strategy Map with the SUIT Route Map

Both the individual strategy maps and the collective map shown in Figure 7.8 comprise an initial contribution to an understanding of the manner in which evaluators tackle the comparison and analysis of UIDT data within a visualisation environment similar to SUIT. This information can therefore be used to further extend, and make more complete, the SUIT route map introduced in chapter 4. The modelled strategies are only applicable to project-specific UIDT selection, and therefore immediate attempts to combine the strategy map in Figure 7.8 and the SUIT route map are restricted to the project-specific version of the latter. This is shown in Figure 7.9 which outlines the project-specific path through the SUIT route map and the top-level strategy breakdown for project-specific UIDT data comparison and analysis (for the further levels of detail corresponding to the latter see Figure 7.8).

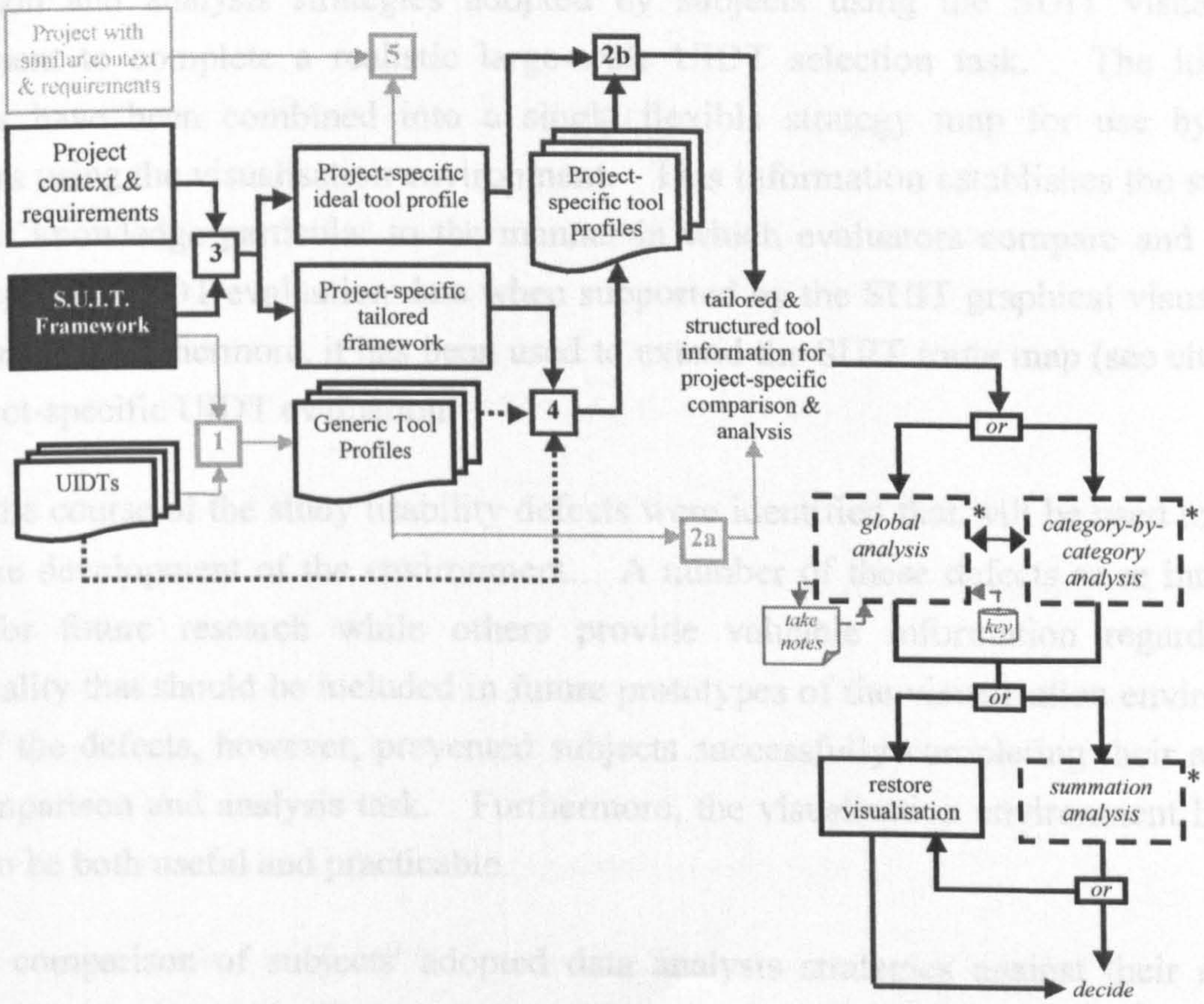


Figure 7.9 - *project-specific SUIT route map incorporating data comparison & analysis strategy guidance*

The diagram illustrates the potential for expansion of the SUIT route map to incorporate data comparison and analysis strategies. Further investigation will be required to determine the strategies adopted when performing generic UIDT evaluations and when selecting a UIDT for a specific project based on similarities with preceding projects. This was, however, beyond the scope of this thesis.

7.7.4 A Possible Correlation Between Strategy and Results

Of the subjects who completed their data comparison and analysis task, all bar one recommended the same UIDT. Interestingly, those who agreed on their recommendation adopted an often similar systematic structured strategy (see subjects 1, 2, 3, and 6) and the one subject who recommended the other UIDT was less obviously systematic in her approach (see subject 4)[78]. Although not taking into account the quality of their recommendations, this suggests that there may be a correlation between the strategy used and the result of a comparison and analysis. As an interesting issue for future investigation, this is discussed further in chapter 8.

7.8 Summary

This chapter presented the results of a qualitative study designed to investigate data comparison and analysis strategies adopted by subjects using the SUIT visualisation environment to complete a realistic large-scale UIDT selection task. The identified strategies have been combined into a single flexible strategy map for use by future evaluators using the visualisation environment. This information establishes the start of a corpus of knowledge particular to the manner in which evaluators compare and analyse project-specific UIDT evaluation data when supported by the SUIT graphical visualisation environment. Furthermore, it has been used to extend the SUIT route map (see chapter 4) for project-specific UIDT evaluation.

During the course of the study usability defects were identified that will be used to inform the future development of the environment. A number of these defects raise interesting issues for future research while others provide valuable information regarding the functionality that should be included in future prototypes of the visualisation environment. None of the defects, however, prevented subjects successfully completing their allocated data comparison and analysis task. Furthermore, the visualisation environment has been shown to be both useful and practicable.

Finally, comparison of subjects' adopted data analysis strategies against their resultant UIDT recommendations indicate potential correlation between strategy and results.

[78] Subject 5 did not complete his comparison and analysis task and did not, therefore, make a UIDT recommendation.

CHAPTER 8 : THESIS CONTRIBUTIONS

8.1 Introduction

This chapter reviews the main contributions of this thesis to the evaluation and selection of user interface development tools (UIDTs). The contributions are organised into three categories. The first collects contributions related to the main aim of this thesis. These represent the result of research into the development of a new context-sensitive facility for UIDT selection. The second category of contributions relate to the results of observations and investigations that were conducted in the process of developing a new UIDT selection facility. Finally, the third category discusses further work that has been identified as a result of completing the research presented in this thesis.

8.2 A New Facility For UIDT Evaluation

Having identified the complex decision space in which UIDTs are selected - and in particular the influence of project-specific requirements and context on this selection process - this thesis aimed to develop and support a project-specific context-sensitive methodology for UIDT selection. In achieving this aim this thesis:

- provides a context-sensitive UIDT evaluation framework and methodology,
- demonstrates that the context-sensitive evaluation framework and methodology is viable,
- presents a novel visualisation environment for SUIT-specific UIDT data,
- demonstrates that the visualisation environment is both useful and practicable,
- suggests a model of UIDT data comparison strategies.

Provides a Context-Sensitive UIDT Evaluation Framework and Methodology

The SUIT framework and methodology are a new project-specific context-sensitive perspective on UIDT evaluation and selection. While maintaining a facility for generic UIDT evaluation, SUIT introduces support for UIDT selection that is contextualised on the basis of project-specific fitness for purpose. Based on a framework that combines an extensible and tailorable reference model of all functionality and support features that might be found in a UIDT with the dimensions and associated scales against which a UIDT

is measured, SUIT defines a comprehensive methodical approach to UIDT evaluation that guides: (1) generic UIDT selection; (2) selection of a UIDT for a specific project that has no precedent for which existing project-specific SUIT evaluation data is available; and (3) selection of a project-specific UIDT on the basis of contextual comparison with previous projects for which project-specific SUIT evaluation data is available.

SUIT recasts previously accepted dimensions of measurement of UIDTs to reduce evaluator bias, increase the practicability of UIDT selection, and increase consideration of context. Most significantly, SUIT focuses on the learnability aspects of usability; SUIT defines *prima facie* learnability as an observation of the match between: (1) the interaction mechanisms used by the UIDT; and (2) the interaction mechanisms most suitable for the given project (and in particular the project team). SUIT introduces the concept of an ideal tool profile to embody or model the notion of project-specific UIDT suitability.

SUIT offers the basis for addressing the important issue of trustworthiness of UIDT evaluation data. Firstly, by virtue of being a repeatable documented methodology, it is possible to establish review procedures for SUIT evaluations. Secondly, the documents that can be generated using SUIT - completed evaluation frameworks, recorded visualisations, and structured and justified recommendations - have the potential to form the core components of evaluation audit trails. These issues are considered in section 8.4.

SUIT introduces the possibility of project-specific reuse of UIDT selection results as part of an overall strategy to reduce the time and effort costs incurred when performing UIDT evaluation. Collaborative evaluation is also potentially possible - see section 8.4.

Demonstrates that the Context-Sensitive Evaluation Framework and Methodology is Viable

SUIT has been shown to be effective in terms of directing evaluators' attention to aspects of context of use and to be viable as a paper-based facility for UIDT evaluation and selection (see chapter 5).

Presents a Novel Visualisation Environment for SUIT-Specific UIDT Data

The SUIT framework structure, and strategy embodied in the SUIT methodology, informed the design and development of a novel graphical visualisation environment for SUIT-specific UIDT data. Adopting a table-based spatial layout, the environment follows accepted principles of table manipulation wherever possible - while facilitating the principle of focus+context, it places restrictions on column relocation to embed SUIT-specific semantics into the table manipulation.

The visualisation presents abstractions over functionality using greyscale. Details of interaction mechanisms and interaction assistance are rendered using a novel graphical representation that exploits colour-coding, positional-coding, and orientation to display swatches corresponding to each of the interaction mechanisms and interaction assistance types recorded in the SUIT framework. The remaining measurement information is provided according to the principle of details-on-demand.

A combination of the graphical representation and a suite of filters enables evaluators to compare and analyse their evaluation data in a way not possible in any previous UIDT evaluation facility. In particular, given the presence of an ideal tool profile, it is possible to use visual pattern-matching techniques for data comparison.

Demonstrates that the Visualisation Environment is Both Useful and Practicable

The SUIT data visualisation environment has been shown to be both useful and practicable in terms of assisting evaluators to systematically compare and analyse complex UIDT evaluation data (see chapter 7).

Suggests a Model of UIDT Data Comparison Strategies

The qualitative evaluation of the SUIT data visualisation environment identified a set of strategies for data comparison, thereby establishing an initial corpus of knowledge regarding comparison and analysis for graphically-represented UIDT data. These strategies have been combined into a comprehensive model of, or guide to, SUIT-specific UIDT data comparison and analysis. When combined with the SUIT methodology, this completes an overall strategy for performing contextualised UIDT evaluation.

8.3 Associated Studies and Observations

To inform the development of the facility for context-sensitive UIDT evaluation a number of studies were undertaken, including an industrial survey and evaluative observations of the various stages of SUIT. The results of these studies contribute to a better understanding of the process of UIDT selection. In particular this associated investigation:

- provides an up-to-date review of UIDT evaluation facilities (see chapter 2);
- suggests a new categorisation of UIDT types (see chapter 2);
- provides survey data for user interface development strategies together with methods of selection and subsequent use of UIDTs (see chapter 3);
- provides an empirical comparison of two UIDT evaluation facilities (see chapter 5);

- suggests a new modelling notation for UIDT comparison and analysis strategies (see chapter 7).

Provides an Up-to-date Review of UIDT Evaluation Facilities

A comprehensive review of current UIDT evaluation facilities (methods, techniques, tools, etc.) identifies mechanisms available for UIDT evaluation prior to the development of SUIT, highlighting their strengths and weaknesses. The available facilities are categorised as: (1) checklist-based; (2) quality attribute-based; and (3) wish-list-based. The relationships between the facilities are highlighted where applicable. On the basis of their weaknesses, it is argued that available evaluation facilities do not adequately support current UIDT evaluation needs. In particular it is argued that these facilities: (1) place too great an emphasis on the skills, experience, and preference of the evaluator; (2) lack extensibility and tailorability and therefore flexibility; (3) do not tackle the issue of trustworthiness of either the data used during the evaluation or the data produced as a result of the evaluation; and (4) do not consider reuse of UIDT selection data and results. Evaluation facilities for CASE tools are also considered for comparative purposes, but in general these too are subject to similar criticisms. On the basis of these observations, the need is established for a new UIDT evaluation methodology that is sensitive to the contextualised requirements placed on UIDTs.

Suggests a New Categorisation of UIDT Types

Following a review of existing high-level UIDT classifications, an updated categorisation is suggested. On the basis of the UIDT sub-type descriptions defined by Valaer and Babb, this is further extended to include a series of connectivity relationships between the various tool sub-types. It is proposed not only that such a categorisation can be used to classify a UIDT, but also that the categorisation can determine the suitability of an evaluation facility for any given UIDT type.

Provides Survey Data for User Interface Development Strategies Together With the Selection and Subsequent Use of UIDTs

A detailed industrial study identifies and documents the user interface development strategies - according to application domain - adopted within industrial software development teams, and provides a detailed account of application characteristics. It also identifies the strategies used to select UIDTs within industrial software development teams and the types of UIDT used. The results indicate that there is not a domain-orientation to the majority of the features surveyed. On the basis of the identified data, relationships are observed among application characteristics and the use of UIDTs, as well as among software engineering methods and the use of UIDTs. These observations not only update

the study previously conducted by Myers and Rosson, but also contribute to a knowledge base and understanding of how interface development is performed in industrial practice, and demonstrate how UIDTs are selected to assist in that development.

Provides an Empirical Comparison of Two UIDT Evaluation Facilities

The detailed empirical study of the effectiveness and viability of SUIT - using the Hix *et al* checklists as a control - provides a comparison of the paper-based version of both UIDT selection facilities. This comparison identifies the difference in focus of evaluator attention during UIDT evaluation and subsequent UIDT recommendation using both methods.

Suggests a New Modelling Notation for UIDT Comparison and Analysis Strategies

A novel graphical notation was created to model identified data comparison and analysis strategies. Focusing on sequencing, flexibility, and patterns of activity, the notation adopts a graphical representation that allows data comparison and analysis strategies to be modelled.

8.4 Further Work

As a result of the research presented in this thesis, a number of suggestions for further work have been identified. These loosely correspond to: (1) scope for extensions to/generalisation of SUIT; (2) further evaluation or assessment of SUIT; and (3) associated research avenues highlighted as a result of developing SUIT. Each of these suggestions contribute to the research agenda in this field, and are therefore discussed below.

8.4.1 Extensions To and Generalisation Of SUIT

This research has identified a number of possible extensions to SUIT. Additional extensions were highlighted during conversations about SUIT with software engineers, many of whom were familiar with CASE tool use, development, and evaluation. The following suggested extensions therefore not only relate directly to the current focus of SUIT, but also extend the applicability of SUIT beyond the boundaries of UIDT selection.

The most obvious indicators of future extensions to SUIT are the identified usability defects discussed in chapter 7. In order that the SUIT visualisation environment may comprehensively support SUIT-specific data comparison and analysis, these results - and in particular the requests for additional functionality - need to be addressed during future development of the environment.

Observation of subjects using the SUIT visualisation environment illustrated the need to introduce higher level task and information management facilities. Although the precise nature and means to achieve such support would need careful consideration, inspiration may be taken, in part, from the likes of the 'to do' list facility in ARGO (Robbins et al., 1997, Robbins and Redmiles, 1998). ARGO - a software architecture design environment - has successfully adopted a 'to do' list metaphor to embody design critiques and personal reminders posted by designers during the evolution of architecture design. The success of the 'to do' list metaphor is attributed to the fact that designers of complex systems are accustomed to having many small tasks pending and so find it familiar and useful. Additionally, the 'to do' list reduces designers' reliance on short-term memory and provides a convenient mechanism by which they can organise and browse reminders (Robbins and Redmiles, 1998). Although the benefit of introducing critics to data comparison and analysis is as yet unexamined, the inclusion of a 'to do' list on which evaluators could post reminders of analysis activities has already been highlighted as a beneficial addition to the SUIT visualisation environment.

Additional extensions to the existing functionality were highlighted implicitly in chapter 6 during discussion of necessary restrictions placed on the implementation of the initial prototype. These facilities (including increased capabilities for query generation) and other more sophisticated features (for example, true fisheye lenses, parameterised filters, 'bubble-help', and abstractions over detail other than functionality) could be added to the SUIT visualisation environment. In particular, chapter 6 noted that automatic generation of the ideal tool profile was restricted to consideration of suitable interaction mechanisms only on the basis of team members' preferences. To more completely reflect the methodology, this aspect of the visualisation environment should be extended to allow detailed control and tailoring of the interaction mechanism/functionality correlation in the ideal tool profile.

Since research attention was focussed on the visualisation (in preference to the storage) of SUIT data, the implementation of assistance for tailoring electronic SUIT frameworks on the basis of individual UIDT sub-types was temporarily postponed. As an important issue to consider for future work, the specific characteristics of UIDT sub-types will need to be carefully defined, and the mechanism by which to assist tailoring will also require careful consideration. One potential solution is to provide a series of *framework templates* that are already tailored according to UIDT sub-type, thus allowing an evaluator to select the appropriate template to initiate an evaluation process. Alternatively, it may be more appropriate to build a *tailoring wizard* that guides an evaluator through the process of tailoring the SUIT reference model according to the characteristics of a selected UIDT sub-type. The provision of a wizard need not stop there - the idea could be further extended to

design a SUIT wizard that guides an evaluator through all steps required to complete a SUIT-based UIDT selection.

At present, only the component lists within the electronic frameworks can be tailored and/or extended. Consider, for example, the listed interaction mechanisms. Although this list can be extended using the paper-based framework, the list of mechanisms is at present fixed in the electronic framework. To fully observe the principle of extensibility in the software support for SUIT it is important to ensure that the dimensions of measurement are also flexible.

There is scope for further investigation of the software development processes highlighted by the industrial survey discussed in chapter 3. The identified processes highlighted industrially accepted user interface design and development activities such that they could justifiably be included in the SUIT framework. However, further investigation may help to model these processes with greater sophistication and/or detail. In particular, it would be interesting to investigate the specific points during the development life cycle at which different team members - according to their developmental rôle - are involved. Thereafter, it would be useful to determine which UIDT components are relevant to each of the activities and specific rôles. In relation, it would also be interesting to investigate the extent to which organisations alter their development procedures to accommodate UIDT use in situations where no UIDT can be found to support their existing practice. The results of such investigations could be used to refine and extend SUIT to guide the selection of UIDTs on the basis of contextualised rôles - that is, with respect to the project-specific context-sensitive rôle(s) or activities the UIDT will be required to support.

During discussions with software engineers, scope for generalising SUIT was explored. An initial suggestion was that a version of the SUIT framework could be developed for CASE tools. In principle the extensibility of the SUIT framework allows the tailoring necessary to accommodate CASE tool evaluation. However, since the extent of such tailoring may make the selection process cost prohibitive, a CASE tool version of the framework could be developed and provided with SUIT.

There is, however, potential to generalise SUIT far beyond CASE tool evaluation. While maintaining the underlying methodological principles of SUIT, the structure (not contents) of the framework and visualisation environment could be generalised such that they could accommodate the evaluation of any software[79]. In theory, the structure of the SUIT framework could be established such that a software engineer can specify the component hierarchy and dimensions of measurement relevant to the software being evaluated.

[79] Or ultimately any 'thing'.

Similarly, the implementation of the visualisation environment could be generalised to enable it to process any component hierarchy and measurement dimensions specified within the generalised SUIT framework. However, there are a number of practical issues that would require investigation. Not least of these is the question of whether the approach would successfully scale up in terms of the number of components and/or the number of software systems being evaluated during any single selection procedure. The investigation of issues surrounding the evolution of SUIT from a UIDT-specific evaluation facility into a generic facility for the evaluation of any type of software would therefore be an interesting avenue of future research.

The evaluation mechanisms outlined in chapter 2, together with the results of the industrial survey outlined in chapter 3, suggest that a UIDT evaluation and selection task is typically performed by one person. On this basis, SUIT has been developed for single person use. That said, however, the core constituent components of a SUIT evaluation (the electronic frameworks, electronic ideal tool profiles, and visualisations - in particular the snapshot facility) comprise a basis from which it would be possible to extend the SUIT evaluation methodology into the domain of collaborative activity. It would, therefore, be interesting to investigate the potential demand for support for collaborative UIDT evaluation and selection and thereafter, if required, to make the necessary technical extensions to SUIT.

The following suggested extension to SUIT is perhaps the most challenging - it concerns the development of a mechanism by which SUIT data can be shared and reused. The basic idea is to develop an open (freely available) source of SUIT data to maximise the benefit of tool evaluation using SUIT. Profiles could be generated for industrially available UIDTs and made available via this 'open-source'. The availability of such profiles would reduce the time overheads in UIDT evaluation, and thus perhaps make selections more cost effective. As was suggested in chapter 2, there is a complex issue of trust in the source of this kind of data and so mechanisms by which to increase levels of trust in the profiles have to be investigated. An initial suggestion is to allow a registered community of software developers to update and/or alter and/or version them, thereby establishing communally-agreed 'correctness' of profiles. On the basis of the accepted model of academic research publication review, it is anticipated that such peer-reviewed tool profiles would afford acceptable levels of trust from the software evaluators. Trust could be further enhanced given that, using SUIT, evaluators could compare versions of profiles for the same UIDT to determine the general consensus of opinion. The issue of assessing this trust is discussed further in section 8.4.3.

Allowing evaluators to update and version tool profiles raises a further complex problem - that of intellectual ownership of the data. There are a number of potential models of

ownership upon which a strategy for the peer-reviewed tool profiles could be based - for example the model according to which research is shared or the concept of freeware software. It would be very interesting to investigate or develop a model that best supports 'open-source' peer-reviewed UIDT profiles, given their intended use.

It may be possible to extend the SUIT 'open-source' concept one step further to include the results of actual UIDT selections. The third route through the SUIT methodology promotes the idea that UIDTs can be chosen on the basis of project context match between new projects and old projects, for which there exists SUIT-based project-specific selection documentation. Theoretically, the boundaries of this information sharing need not correspond to the boundaries of an organisation. Instead, the 'open-source' could provide the means by which evaluators could register their electronic ideal tool profiles, visualisations, and documented UIDT recommendations. Future evaluations could then benefit from comparison of project requirements and context with previously registered selection documentation. At present, a combination of paper-based forms, ideal tool profiles, and visualisations constructed by evaluators' captures some of the rationale supporting UIDT selection decisions. It would be necessary to further investigate exactly what additional decision rationale information is required, and the manner in which it should be recorded, in order to fully support effective UIDT evaluation and selection decision reuse. Given the availability of appropriate information concerning decision rationale, there is scope to adopt and incorporate the principles of recommender or path-model based systems (Chalmers et al., 1998, Chalmers, 1999). If the 'open-source' was to record the activities of evaluators when reusing evaluation data and results - for example, their search path and resulting selection - it would be possible to either: (1) recommend UIDTs to future evaluators on the basis of a match between their recorded activity and that of previous evaluators; or (2) to allow future evaluators to selectively follow the path or paths of previous evaluators in whom they *choose* to place their trust. Perhaps to an even greater extent than peer-reviewed tool profiles, this concept would have to tackle the issues of trust and intellectual ownership, and to consider company confidentiality.

8.4.2 Further Evaluation or Assessment of SUIT

The results of the industrial study discussed in chapter 3 (a) exhibited potential bias that could ultimately limit the applicability of SUIT and (b) informed some assumptions upon which SUIT is founded. Having developed SUIT, it would be interesting to assess the effect of the potential bias and to determine whether the assumptions are universally justifiable. The most salient of these issues (identified within chapter 3) are discussed below.

The results of the study did not illustrate domain-oriented classification of application characteristics, software development procedures, or the use of UIDTs. Neither did they highlight domain-orientation with respect to the relationships between these factors. SUIT has therefore been developed on the assumption that its applicability is not restricted to any given domain. If it was feasible to identify a sample population distributed across different domains and to evaluate the use of SUIT within each domain, it would be possible to assess whether or not SUIT is - on the grounds of comparative effectiveness and viability - in fact restricted to certain domains and, if so, which ones.

The design of SUIT was informed by characteristics of applications and their development that were observed as a result of the industrial survey (see chapter 3). Since the majority of the responses to the survey were given for middle-sized applications, it cannot currently be claimed that SUIT is applicable outwith this scope of application size. Although it is not anticipated that the size of an application will adversely affect the qualification of SUIT - other than perhaps the time available for UIDT selection - this should be further investigated to either place restrictions on, or remove application-size oriented boundaries from, the applicability of SUIT.

When the SUIT visualisation environment was introduced, it was noted that an accompanying scenario-based user manual had also been developed. The evaluative studies of the use of the visualisation environment focussed on the software and thus the user manual was not evaluated. To ensure that SUIT is adequately supported in terms of assistance, the user manual also needs to be evaluated. Given that the user manual is scenario-based, there are a number of potential perspectives from which to approach its evaluation. One such approach might be to assess the degree to which, and the effectiveness with which, subjects explore the environment on the basis of the scenarios shown in the user manual.

To further increase the level of assistance provided in the environment - in particular to assist understandability of the various categories/dimensions - the introduction of 'bubble-help' was suggested (see section 8.4.1). This too would require evaluation to determine whether it is the most appropriate means of conveying this information, and to clarify the component and dimension definitions.

A model of data comparison and analysis strategies (see chapter 7) was used to extend the project-specific path through the SUIT methodological route map to include optional guidance for the comparative analysis of the data once collected. This addition to the SUIT route map has not yet been evaluated. The comparison map, based on observation of comparison activities in practice, is only intended as an optional guide for evaluators. It would be interesting to base its evaluation on an observation of: (1) whether evaluators use

the suggested comparison strategies or revert to their own; and (2) which of the optional routes through the comparison strategy are most often and most successfully employed. It would also be interesting to provide a series of evaluators with a visualisation of several tools including an ideal tool profile, the map of comparison strategies, and a series of questions of varying complexity and ask them to follow the map to answer the questions. Hopefully the results of such evaluation would lead to a more complete appreciation of the value of the map as it stands, and to an understanding of how it could be extended or improved. Once evaluated, and if necessary amended, the data comparison and analysis strategy map could be incorporated into the interaction design of the SUIT data visualisation environment so that evaluators could, if desired, be guided by the tool itself. The best means by which to achieve such automated guidance would require careful investigation and, once implemented, further evaluation.

At present, a model of data comparison and analysis strategies is only available for project-specific UIDT selection; the strategies by which evaluators compare data for generic UIDT evaluations, and their strategies for reuse of UIDT selection results have not yet been observed. To model these strategies, so that they can be included in the corresponding paths through the SUIT route map, will require further observational studies similar to the one discussed in chapter 7.

Contrary to research findings which suggest that when tables are being used to present data for comparison tasks the salient issue should be represented by the table rows, a number of the subjects requested that SUIT columns be used to represent the tools and SUIT rows be used to represent components (see chapter 7). It is unclear whether this is a direct result of their observed problems with the orientation of the component labelling or is a genuine preference. To ensure that the spatial layout of the visualisation environment is not considered a hurdle to its effective use for the majority of evaluators, alternatives to the adopted label orientation should first be evaluated to determine its influence on the perception of the table; if necessary, the alternative table orientation should then be evaluated.

In chapter 7 it was stated that there is evidence of a correlation between (i) data comparison and analysis strategies and (ii) the result of a UIDT selection. Taking into consideration the quality of the associated tool recommendations, it would be interesting to further investigate - and empirically substantiate - such a correlation to better support or guide future evaluators in the use of SUIT.

SUIT has been shown to be effective in terms of directing evaluators' attention to aspects of context of use, to be viable, and (with the inclusion of the SUIT data visualisation environment), to be practicable. What remains to be evaluated is the effectiveness of

SUIT in terms of assisting the selection of the most appropriate UIDT for any given project. This is a very complex concept to measure and it will require considerable thought to determine the means by which to assess effectiveness in this respect. Perhaps one of the best means by which the effectiveness of SUIT may be measured is via the use of a number of truly longitudinal industrial studies. To do this, the lifetime of a series of software development projects would need to be monitored - from the initial stages of project team allocation through to the accepted installation and maintenance of the developed software - for which SUIT is used to select the most appropriate UIDT. This kind of study would allow problems with the selected UIDT to be identified in real-time and thus related to the selection process if appropriate. It would illustrate the extent to which the selected UIDT actually met the needs of the project in terms of functionality and context of use. Furthermore, over an extended period of time, the reusability of the SUIT data could also be evaluated. Such studies would, however, be costly to conduct given that to return meaningful data they would potentially have to run for several months, even years.

Ultimately, it would be interesting to devise a structured means by which to evaluate the effectiveness of any evaluation methodology. Having devised a mechanism for this kind of evaluation, it could be applied to the various tool type-based versions of SUIT as suggested in section 8.4.1. In particular, it would be interesting to use SUIT to evaluate itself. This would necessitate characterisation of UIDT selection facilities and the inclusion of this characterisation within the generic SUIT framework suggested in section 8.4.1.

8.4.3 Associated Research Avenues

During the course of this research a number of possible avenues for associated research were identified but were considered to be beyond the primary focus of this thesis. These issues are highlighted and discussed briefly below.

Chapter 1 identified the complex process of assessing trade-offs between various evaluation criteria within the decision matrix. At present there is little knowledge of the mechanism(s) by which UIDT-specific trade-offs are performed. The qualitative evaluation of the SUIT visualisation environment (see chapter 7) highlighted some observations in this respect - for example, the general call for prioritisation of functionality in order to assist the balancing of decision criteria. Not only would such investigation prove very interesting, but it would also inform the further development or extension of systems designed to assist in flexible decision making, not least SUIT.

SUIT incorporates a two scale ranking of evaluation components - that is, components are either included or excluded from the active evaluation set. The qualitative evaluation of the SUIT visualisation environment identified a general demand for component prioritisation to assist evaluators in the decision making process. Although the visualisation environment allows the left-to-right spatial prioritisation of components, one might investigate alternative mechanisms to express the priority of components. As was commented in chapter 2, numerical prioritisation of components has proven to be costly and often infeasible. This therefore suggests that the following needs to be investigated: (a) the point during the evaluative process at which component prioritisation need/should be considered; (b) depending on the stage at which it is introduced, the mechanism - for example, language - by which it is expressed; and (3) the optimal number of priority levels. The avoidance of weights and ratings in the SUIT framework is considered a strength of its design. In particular, their absence makes the proposed reuse of SUIT evaluation documentation more feasible (see section 8.4.1). The investigation of the means by which to incorporate component prioritisation within the SUIT methodology should therefore also consider this dimension of SUIT and how best to prevent hindrance of evaluation result reuse.

When discussing the results of the industrial study in chapter 3, it was noted that bias may have been introduced into the data because the companies surveyed were all situated in or near Glasgow. It was, however, commented that it may be difficult to avoid geographically-oriented bias on the grounds that not enough is yet known about the correlation (if any) between geographical location and software development. Focussing primarily on SUIT, this would be a very interesting question to address. In particular, it would be interesting to apply the study questionnaire across a geographically selected sample population to try to identify geographically characteristic responses to assess whether the results - upon which SUIT was based - would be significantly different. Similarly it would be interesting to evaluate the use of SUIT across a geographically distributed population to assess the extent to which the geographical bias has restricted the applicability of SUIT.

The primary aim of SUIT is to assist an evaluator in the identification of issues to be considered during a UIDT selection - in particular, to focus on the tool data to be collected and then compared to the ideal tool profile. SUIT makes no suggestion as to the most appropriate techniques for collecting that data. It would be interesting, however, to investigate such techniques - which include feature inspection, architectural inspection, and benchmarking - in order to identify which is most appropriate given the characteristics of SUIT.

During the assessment of the effectiveness of SUIT in terms of focussing evaluators' attention on issues of context of use (see chapter 5), it was discovered that, even unsupported, there were some dimensions against which evaluators seemed to intuitively measure UIDTs - for example, learnability and interaction assistance provision. It would therefore be interesting to investigate the criteria and dimensions that would feature in unsupported evaluation of UIDTs performed by subjects who are unfamiliar with UIDT evaluation and/or have no experience of using any of the available UIDT (or similar) evaluation facilities. This would require careful profiling of the subjects to select a population of subjects who either had no experience of formal UIDT evaluation and/or had no experience of any of the available UIDT evaluation facilities.

The statistical analysis of the results of the evaluation of the paper-based version of SUIT (see chapter 5) identified a significant negative correlation between mental demand and performance for the subjects in the Hix group - for these subjects high mental demand correlated to low confidence in their performance. In contrast, albeit not statistically significant, the equivalent correlation for SUIT subjects was positive - increased mental demand correlated with high levels of confidence in performance. Since measures of correlation are not indicative of causal relationships between the contributing factors, no conclusion can be drawn from these findings. The relationship between mental demand and confidence in performance is interesting and important, given that, if it could be shown that increased mental demand was the *cause* of increased confidence in performance for SUIT, it would endorse SUIT.

The issue of confidence in performance highlights perhaps the most significant avenue for further associated research, that is, trustworthiness of UIDT evaluation data. As was discussed in chapter 2, there are two contributory aspects of the trustworthiness of UIDT data: (1) trust in the abilities or judgement of the person(s) who makes the final UIDT recommendation; and (2) trust in the correctness of the fundamental UIDT data. Consider first the issue of trust in the judgement of the evaluator making a UIDT recommendation. SUIT promotes the idea of archiving UIDT evaluation and recommendation data for reuse by future evaluators. In section 8.4.1, it was suggested that this idea was extended to create an 'open-source' of SUIT-based UIDT recommendations. Were this to be available, it raises the important question of whether or not evaluators would have sufficient trust in the quality and judgement of another person's evaluation to use the catalogued results. This question is likely to be even more significant when considering cross-organisation trust in archived data where an evaluator has no knowledge of the person(s) who completed the catalogued evaluation. There are also likely to be issues of confidentiality that may impede the completeness of catalogued evaluation records. It was suggested in chapter 2 that SUIT has the potential to assist in establishing trust of this nature by way of

the fact that: (1) it provides documented repeatable methods by which the data could be validated; and (2) the tailored framework, ideal tool profile, and visualisation records could be considered to comprise an audit trail by which the evaluation process and recommendation could potentially be traced. Furthermore, if the peer-review procedures suggested in section 8.4.1 were to be implemented - possibly by trained and certified evaluators - this should potentially increase the perceived trustworthiness both of UIDT data and ultimately UIDT recommendations. This trust could potentially be further increased if the 'open-source' was administered by an accredited third-party who was responsible for managing and versioning UIDT data profiles, and ensuring submitted UIDT recommendation document met with an agreed standard.

Given that the concept of a contributory reservoir of UIDT evaluation information is currently only hypothetical, the issue of the associated trustworthiness of the data is, as yet, untested. Although there are a number of possible means by which the trustworthiness of SUIT data could (and should) be assessed, the immediate approach (prior to establishing the 'open-source') might be to compare the trustworthiness of SUIT data to that of data generated using the Hix *et al* checklists. The researcher could conduct and document - that is, provide the evaluation material and a short recommendation report - an evaluation of two UIDTs using both methods. Subjects could then be asked to indicate and justify which recommendation they considered to be more trustworthy or reliable - in other words, which represents the better quality decision (see chapter 1). The issue of how to quantify subjective measures of trust is obviously a major component of this further work.

During the pilot evaluation of the SUIT visualisation environment, the subject stated that he would be likely to question the reliability of pre-recorded UIDT profiles and would therefore only afford as much trust in the final outcome of an evaluation as he had in the source data (see chapter 7). It is obviously very important to evaluate this issue of trust with respect to SUIT and in particular to investigate the measures that could be taken to increase evaluators' confidence in such data were it to be collected and made available.

8.5 Chapter Summary

Dividing them into three categories, this chapter reviewed the main contributions of this dissertation. The first category concerned contributions directly related to the thesis of this research: that is, the development of an explicit methodology and associated useful and practicable tools to support UIDT selection according to suitability in a given context (see chapter 1). As a novel methodology developed to guide systematic selection of UIDTs according to project-specific context-sensitivity, SUIT successfully focuses evaluator attention on issues of context of use when selecting a UIDT for a given project. By

recasting conventional approaches to UIDT evaluation, SUIT demonstrates the potential to address the issue of the trustworthiness of UIDT evaluation and selection data. SUIT is supported by a novel visualisation environment which has been shown to be a useful and practicable aid to SUIT-specific data comparison and analysis. Observation of the use of this visualisation environment has established an initial corpus of knowledge regarding the manner in which UIDT data is compared and analysed for the purpose of project-specific UIDT selection.

The second category considered the contributions related to the main aim of this dissertation: that is, to investigate the means by which to guide and support consideration of context of use during UIDT evaluation such that tools are selected according to their suitability within a given context. The result of the principal investigation necessary to inform the development of SUIT, these contributions include: (1) an up-to-date understanding of UIDT selection and subsequent use in industrial practice, including an understanding of industrial user interface development strategies and the significance of UIDTs therein; (2) a comprehensive critical review of available UIDT evaluation facilities; (3) an empirical comparison between SUIT and the Hix *et al* checklists; (4) an updated UIDT categorisation; and (5) a new notation to model the complexities of UIDT data comparison and analysis strategies.

The final category focussed on how the contributions (and associated research) identified in the first two categories can be exploited, and the associated tool support improved or extended. In essence, the future work contributions comprise: (1) extensions to and generalisation of the SUIT selection facility; and (2) investigation of issues that are either directly related to such extensions or that are suggested as a result of concepts and questions raised during the development of SUIT.

One of the most significant suggestions is perhaps the future extension of SUIT to more easily accommodate different UIDT sub-types, CASE tools, and ultimately its generalisation as a structured, principled software evaluation facility. Initial, informal discussion with software engineers indicates the demand for such facilities, suggests that SUIT is well placed to meet those needs given the appropriate extensions.

One of the most challenging of the identified suggestions for further research is the investigation of the means by which to measure the effectiveness of SUIT and, more generally, software evaluation facilities. It has been demonstrated that SUIT is effective in terms of directing evaluator attention towards issues of context of use and increases the quality of UIDT selection decisions, but the question that remains to be answered is whether or not SUIT evaluations actually result in the selection of the most appropriate

UIDT. Although the initial results outlined in this thesis suggest this will be the case, this needs to be formally assessed via longitudinal case studies.

Finally, this chapter outlined an idea to 'open-source' SUIT - to establish a repository of updateable UIDT profiles, and to establish a registration and reference mechanism for completed UIDT evaluation documentation so that previous evaluative work is more widely accessible and reusable. Although the most probable medium by which this can be achieved is the world wide web, there are many issues that will require investigation, such as: the security measures that software engineers would expect; whether access should be restricted to registered contributors only; the means by which review procedures for the evaluation documentation can or should be established; the ownership of the data; and the means by which evaluators can contribute to, and peer-review, the collaboratively-generated UIDT profiles. Nevertheless, the most significant issue concerning the establishment of an 'open-source' for SUIT is the matter of trust. Ultimately, the goal of this future research is to extend SUIT as a UIDT evaluation facility that is adopted by industrial software developers, and in which software engineers place their trust.

REFERENCES

Ahlberg, C. and Shneiderman, B. (1994), Visual Information Seeking: Tight Coupling of Dynamic Query Filters with Starfield Displays, In *Proceedings of Human Factors in Computing Systems (CHI'94)*, Boston, Massachusetts, April 24-28, pp. 313 - 317

Ahlberg, C. and Truve, S. (1995), Exploring Terra Incognita in the Design Space of Query Devices, In *Proceedings of Engineering for Human Computer Interaction EHCI'95*, North Holland

Albers, M. J. (1996), Decision Making: A Missing Facet of Effective Documentation, In *Proceedings of The 14th annual international conference on Marshalling new technological forces: building a corporate, academic, and user-oriented triangle (SIGDOC'96)*, ACM Press, pp. 57 - 65

Bass, L., Abowd, G. and Kazman, R. (1994), Issues in the Evaluation of User Interface Tools, In (Eds, Taylor, R. and Coutaz, J.) *Proceedings of Workshop on Software Engineering & Computer-Human Interaction*, May, Springer-Verlag Lecture Notes in Computer Science, pp. 17 - 27

Bertin, J. (1981), *Graphics and Graphic Information Processing*, Walter de Gruyter Press, Berlin

Bodart, F., Hennebert, A. M., Leheureux, J. M., Provot, I., Sacre, B. and Vanderdonckt, J. (1995), Towards a Systematic Building of Software Architectures: the TRIDENT Methodological Guide, In *Proceedings of Interactive Systems: Design, Specification and Verification*, Berlin, Springer-Verlag, pp. 77-94

Boehm, B. (1988), A Spiral Model of Software Development and Enhancement, *IEEE Software*, **21**(5), pp. 61 - 72

Boker, G. A. (1995), User Interface Design Assistance for Large Scale Software Development, *Automated Software Engineering*, **2**(3), pp. 203-217

Booch, G. (1994), *Object-Oriented Analysis & Design with Applications*, The Benjamin/Cummings Publishing Company Inc., Santa Clara

Brown, A. W., Earl, A. N. and McDermid, J. A. (1992), *Software Engineering Environments : Automated Support for Software Engineering*, McGraw-Hill Book Company, London

Brown, A., Carney, D., Morris, E., Smith, D. and Zarella, P. (1994), Principles of CASE Tool Integration, In *Proceedings of OUP'94*

Card, S. K., Mackinlay, J. D. and Robertson, G. G. (1990), The Design Space of Input Devices, In *Proceedings of CHI'90*, ACM Press, pp. 117 - 124

Chalmers, M. (1999), Comparing Information Access Approaches, In *Journal of the American Society for Information Systems (JASIS) 50th Anniversary Issue,* **50,** 12, pp. 1108 - 1118

Chalmers, M., Rodden, K. and Brodbeck, D. (1998), The Order of Things: Activity-Centred Information Acccess, In *Proceedings of WWW7*, Brisbane, pp. 359 - 367

Chambers, J., Cleveland, W., Kleiner, B. and Tukey, P. (1983), *Graphical Methods for Data Analysis*, Duxbury Press, Boston, MA

Chikofsky, E. J., Martin, D. E. and Chang, H. (1992), Assessing the State of Tool Assessment, *IEEE Software,* May, pp. 18 - 21

Clarke, S. (1997), *Encouraging the effective use of contextual information in design,* Ph.D. Thesis, Department of Computing Science, University of Glasgow, Glasgow

Cockton, G., Clarke, S. and Gray, P. (1996), Literate Development: Weaving Human Context Into Design Specifications, In *Critical Issues in User Interface Systems Engineering*, (Eds, Benyon, D. and Palanque, P.), Springer Verlag, pp. 227 - 248

Dawkes, H., Tweedie, L. and Spence, R. (1996), VICKI - The VIsualisation Construction Kit, In (Eds, Catarci, T., Costabile, M., Levialdi, S. and Santucci, G.) *Proceedings of the Workshop on Advanced Visual Interfaces (AVI)*, Gubbio Italy, May 27 - 29, ACM Press, pp. 257 - 259

Dix, A., Finlay, J., Abowd, G. and Beale, R. (1993), *Human-Computer Interaction*, Prentice Hall International (UK) Ltd, Cambridge

Eckstein, R., Loy, M. and Wood, D. (1998), *Java Swing*, O'Reilly, Sebastopol

Faulkner, X. (2000), *Usability Engineering*, MacMillan Press Ltd, London

Fishkin, K. and Stone, M. (1995), Enhanced Dynamic Queries vs Movable Filters, In (Eds, Katz, I. R., Mack, R. and Marks, L.) *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, Denver CO, May 7 - 11, pp. 415 - 420

Foley, J., Wallace, V. L. and Chan, P. (1984), The Human Factors of Computer Graphics Interaction Techniques, *IEEE Computer Graphics and Applications,* pp. 13 - 48

Gilmore, D. (1991), Visbility: A Dimensional Analysis, In (Eds, Monk, A., Diaper, D. and Harrison, M.) *Proceedings of People and Computers VI - HCI'91*, pp. 317 - 329

Gray, P., England, D. and McGowan, S. (1994), XUAN: Enhancing UAN to Capture Temporal Relationships Among Actions, In (Eds, Cockton, G., Draper, S. W. and Weir, G. R. S.) *Proceedings of People and Computers IX : HCI'94*, Glasgow, Scotland, Cambridge University Press, pp. 301 - 312

Griffiths, T., McKirdy, J., Paton, N., Kennedy, J., Cooper, R., Barclay, P., Goble, C., Gray, P., Smyth, M., West, A. and Dinn, A. (1998), An Open Model-Based Interface Development System: The Teallach Approach, In (Eds, Markopoulos, P. and Johnson, P.) *Proceedings of 5th Eurographics Workshop on Design, Specification & Verification of Interactive Systems - DSV-IS'98*, Abingdon, UK, June 3-5, pp. 32 - 49

Harris, R. (1998a), *Introduction to Decision Making,* Technical Report, Vanguard University of Southern California, July 2, pp. 15

Harris, R. (1998b), *Problem Solving Techniques,* Technical Report, Vanguard University of Southern California, July 2, pp. 17

Harston, H. R. and Gray, P. D. (1992), Temporal Aspects of Tasks in the User Action Notation, *Human-Computer Interaction,* 7(1), pp. 1 - 45

Hart, S. and Staveland, L. (1988), Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research, In *Human Mental Workload*, (Eds, Hancock, P. and Meshkati, N.), North Holland B.V., Amsterdam, pp. 139 - 183

Hart, S. G. and Wickens, C. (1990), Workload assessment and prediction, In *MANPRINT: an approach to systems integration*, (Ed, Booher, H. R.), Van Nostrand Reinhold, New York, pp. 257 - 296

Hix, D. (1986), Assessment of An Interactive Environment for Developing Human-Computer Interfaces, In *Proceedings of the Human-Factors Society - 30th Annual Meeting*, pp. 1349 - 1353

Hix, D. (1991), *An Evaluation Procedure for User Interface Development Tools Version 2.0,* Technical Report Virginia Polytechnic Institute & State University, pp. 58

Hix, D. and Hartson, H. R. (1993), *Developing User Interfaces : Ensuring Usability Through Product & Process*, John Wiley & Sons Inc., USA

Hix, D. and Ryan, T. (1992), Evaluating User Interface Development Tools, In *Proceedings of the Human Factors Society - 36th Annual Meeting*, pp. 374 - 378

Hix, D. and Schulman, R. S. (1991), Human-Computer Interface Development Tools: A Methodology for Their Evaluation, *Communications of the ACM*, **34**(3), pp. 74 - 87

Hix, D., Tan, K. C. and Schulman, R. S. (1989), Development and Testing of An Evaluation Procedure for User Interface Management Systems (UIMS), In *Proceedings of the Human Factors Society 33rd Annual Meeting*, pp. 264 - 267

Holtzblatt, K. and Jones, S. (1993), Contextual Inquiry: a participatory technique for system design, In *Participatory Design: principles and practices*, (Eds, Schuler, D. and Namioka, A.), Lawrence Erlbaum, Hillsdale, NJ, pp. 177-210

Humphrey, W. S. (1989), *Managing the Software Process*, Addison-Wesley, Reading MA

Jacobs, S. and Holten, R. (1995), Goal Driven Business Modelling - Supporting Decision Making within Information Systems Development, In *Proceedings of Conference on Organizational computing systems (COOS)*, Milpitas CA USA, ACM Press, pp. 96 - 105

Joachim, M., Shinar, D. and Leiser, D. (1997), Multiple Factors that Determine Performance with Tables and Graphs, *Human Factors*, **39**(2), pp. 268-286

Johnson, P., Johnson, H. and Wilson, S. (1995), Rapid Prototyping of User Interfaces Driven By Task Models, In *Scenario-Based Design*, (Ed, Carroll, J.), John Wiley & Son, London, pp. 209-246

Jorgensen, M., Bygdas, J. J. and Lunde, T. (1995), *Efficiency Evaluation of CASE Tools - Methods and Results*, Technical Report TF R 38/95, Telnor, 31st August 1995, pp. 35

Kelly, K. (1966), Twenty-Two Colors of Maximum Contrast, *Colour Engineering*, 3, pp. 26-27

Kemerer, C. F. (1992), How the Learning Curve Affects CASE Tool Adoption, *IEEE Software*, May, pp. 23 - 28

Kitchenham, B. A., Linkman, S. G. and Law, D. T. (1994), Critical Review of Quantitative Assessment, In *Software Engineering Journal*, **9**, 2, pp. 43 - 53

Kyng, M. (1994), Scandinavian Design: Users in Product Development, In *Proceedings of Human Factors in Computing Science CHI'94*, Boston Massachusetts, 24 - 28 April 1994, pp. 3 - 9

Kyng, M. (1995), Making Representations Work, *Communications of the ACM*, **38**(9), pp. 46-55

Lindgaard, G. (1994), *Usability Testing & System Evaluation: A Guide for Designing Useful Computer Systems*, Chapman & Hall, London

Lindley, D. and Scott, W. (1984), *The New Cambridge Elementary Statistical Tables*, Cambridge University Press

Loch, K. D. and Conger, S. (1996), Evaluating Ethical Decision Making and Computer Use, In *Communications of the ACM,* **39,** 7, pp. 74 - 83

Lumsden, J. and Gray, P. (2000), SUIT - Context Sensitive Evaluation of User Interface Development Tools, In (Eds, Palanque, P. and Paterno, F.) *Proceedings of Design, Specification and Verification of Interactive Systems*, Limerick, Ireland, June 5 - 6, pp. 91 - 108

McKirdy, J. (1998), *An Empirical Study of the Relationships Between User Interface Development Tools & User Interface Development,* Technical Report TR-1998-06, University of Glasgow, March 1998, pp. 57

Microsoft (1997), Microsoft® Excel97

Mosley, V. (1992), How To Assess Tools Efficiently and Quantitatively, *IEEE Software,* May, pp. 29 - 32

Myers, B. A. (1990a), All the Widgets (2 hour videotape), In *Proceedings of Technical Video Program of the SIGCHI'90*, Seattle, W.A., April 1 - 4, ACM Press

Myers, B. A. (1990b), Some of the Widgets (17 minute videotape), In *Proceedings of Technical Video Program of Interact'90*, Cambridge, England, August 27 - 31

Myers, B. A. (1994), *User Interface Software Tools,* Technical Report CMU-CS-94-182, School of Computer Science, Carnegie Mellon University, August 1994, pp. 43

Myers, B. A. and Rosson, M. B. (1992), *Survey on User Interface Programming,* Technical Report CMU-CS-92-113, School of Computer Science, Carnegie Mellon University, February 1992, pp. 14

Mynatt, B. (1990), *Software Engineering with Student Project Guidance*, Prentice-Hall Inc, Englewood Cliffs, NJ

Newman, W. M. and Lamming, M. G. (1995), *Interactive System Design*, Addison-Wesley Publishing Company, Wokingham

O'Neill, E. J. (1998), *User-developer co-operation in software development: building common ground and usable systems,* PhD Thesis, Department of Computing Science, Queen Mary and Westfield College University of London, London

Oskarsson, O. (1982), Mechanisms of modifiability in large software systems, In *Linking Studies in Science and Technology Dissertations,* 77

Paulk, M. C., Curtis, B., Chrissis, M. B. and Weber, C. V. (1993), The Capability Maturity Model for Software, In *IEEE Software,* **10,** 4, pp. 18-27

Poston, R. M. and Sexton, M. P. (1992), Evaluating and Selecting Testing Tools, *IEEE Software,* May, pp. 33 - 42

Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S. and Carey, T. (1994), *Human-Computer Interaction*, Addison-Wesley Publishing Company, Wokingham

Rao, R. and Card, S. (1994), The Table Lens: Merging Graphical and Symbolic Representations in an Interactive Focus+Context Visualization for Tabular Information, In *Proceedings of Human Factors in Computing Systems (CHI'94)*, Boston, Massachusetts, April 24 - 28, pp. 318 - 322

Rasmussen, J. (1992), The Ecology of Work and Interface Design, In (Eds, Monk, A., Diaper, D. and Harrison, M. D.) *Proceedings of People and Computers VII - Proceedings of HCI'92*, York, September, Cambridge University Press, pp. 3 -20

Robbins, J., Hilbert, D. and Redmiles, D. (1997), Argo: A Design Environment for Evolving Software Architectures, In *Proceedings of 19th International Conference on Software Engineering (ICSE'97)*, Boston, MA, May, Springer, pp. 600 - 601

Robbins, J. and Redmiles, D. (1998), Software Architecture Critics in the Argo Design Environment, In *The Best of Intelligent User Interfaces (IUI'98),* **10**(5), pp. 64

Rumble, C. D. (1991), The Human Element Approach to Decision Making...Let's Try Reorganisation, In *Proceedings of 19th ACM SIGUCCS conference on User services,* ACM, pp. 345 - 350

Sauter, V. L. (1999), Intuitive Decision-Making, In *Communications of the ACM,* **42,** 6, pp. 109 - 115

Schlungbaum, E. and Elwert, T. (1996), Automatic User Interface Generation from Declarative Models, In (Ed, Vanderdonckt, J.) *Proceedings of Computer Aided Design of User Interfaces (CADUI'96)*, Namur, 5-7 June 1996, Presses Universitaires de Namur, pp. 3-18

Scottish-Software-Federation (1995), *Scottish Software Federation: Directory of Member Companies*, Scottish Software Federation, Livingston

Shipman, F., Marshall, C. and Moran, T. (1995), Finding and Using Implicit Structure in Human-Organized Spatial Layouts of Information, In (Eds, Katz, I. R., Mack, R. and Marks, L.) *Proceedings of Human Factors in Computing Systems CHI'95*, Denver, Colorado, 7 - 11 May, ACM, pp. 346 - 353

Shneiderman, B. (1998), *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley, Reading, MA

Simon, H. (1986), *Research Briefings 1986: Report of the Research Briefing Panel on Decision Making and Problem Solving,* Technical Report National Academy of Sciences, pp. 21

Sommerville, I. (2000), *Software Engineering*, Pearson Education, USA

Spenke, M., Beilken, C. and Berlage, T. (1996), FOCUS: The Interactive Table for Product Comparison and Selection, In *Proceedings of ACM Symposium on User Interface Software Technology (UIST)*, Seattle Washington, November 6 - 8, ACM Press, pp. 41- 50

Sundaram, S. and Ramamurthy, K. (1996), A Measurement Methodology for Evaluating User Interface Management Systems, *Journal of Computer Information Systems,* **37**(2), pp. 54 - 61

Symantec (1999), VisualCafe, Professional Version 3.0a, Symantec Corporation

Szekely, P., Luo, P. and Neches, R. (1992), Facilitating the Exploration of Interface Design Alternatives: the HUMANOID Model of Interface Design, In *Proceedings of CHI'1992*, pp. 507-515

Szekely, P., Sukaviriya, P., Castells, P., Muhtkumarasamy, J. and Salcher, E. (1996), Declarative Interface Models for User Interface Construction Tools: The MASTERMIND Approach, In (Eds, Wierse, A., Grinstein, G. G. and Lang, U.) *Proceedings of Engineering for Human-Computer Interaction - 2nd Workshop on Database Issues for Data Visualisation*, Spring-Verlag

Tufte, E. (1983), *The Visual Display of Quantitative Information*, Graphics Press, Cheshire Connetticut

Tweedie, F. (2000), Personal Communication, September 2000

Valaer, L. A. and Babb, R. G. (1997), Choosing a User Interface Development Tool, *IEEE Software,* July/August, pp. 29 - 39

Vessey, I., Jarvenpaa, S. L. and Tractinsky, N. (1992), Evaluation of Vendor Products: CASE Tools as Methodology Companions, In *Communications of the ACM,* **35,** 4, pp. 90 - 105

Vessey, I. and Sravanapudi, A. P. (1995), CASE Tools as Collaborative Support Technologies, In *Communications of the ACM,* **38,** 1, pp. 83 - 95

Weaver, P. L. (1993), *Practical SSADM Version 4 : A Complete Tutorial Guide*, Pitman Publishing, UK

Woods, A., Fletcher, P. and Hughes, A. (1986), *Statistics in Language Studies*, Cambridge University Press

WEB SITES

Chrisholm, W., Vanderheiden, G. and Jacobs, I. (1999), *Web Content Accessibility Guidelines 1.0*, WC3, 2000, http://www.w3.org/TR/1999/WAI-WEBCONTENT-19990505

Kasday, L. (1999), *WAVE*, Pennsylvania's Initiative on Assistance Technology (PIAT), 2000, http://www.temple.edu/inst_disabilities/piat/wave/index.html

Mosley, V. (1995), *Improving Your Process for the Evaluation and Selection of Tools and Environments*, Westington Electronic Systems, 1998, http://www.stsc.hill.af.mil/CrossTalk/1995/sep/improvin.html

Myers, B. (2000), *User Interface Software Tools*, 2000, http://www.cs.cmu.edu/afs/cs.cmu.edu/user/bam/www/toolnames.html

Myers, B. A. (1996), *UIMSs, Toolkits, Interface Builders*, Human Computer Interaction Institute, Carnegie Mellon University, 1996, http://www.cs.cmu.edu/afs/cs/user/bam/www/toolnames.html

ADDITIONAL READING

Aalto, J.-M. and Jaaksi, A. (1994), Object-Oriented Development of Interactive Systems with OMT++, In *TOOLS 14 - Technology of Object-Oriented Languages and Systems*, (Eds, Ege, R., Singh, M. and Meyer, B.), Prentice Hall

Arnold, K. and Gosling, J. (1996), *The Java Programming Language*, Addison Wesley Publishing Company, Reading, MA

Barclay, P. J., Griffiths, T., McKirdy, J., Paton, N. W., Cooper, R. and Kennedy, J. (1999), The Teallach Tool: Using Models for Flexible User Interface Design, In *Proceedings of 3rd International Conference on Computer Aided Design - CADUI'99*, Louvain-la-Neuve, Belgium, 21 - 23 October 1999

Bevan, N. and MacLeod, M. (1994), Usability Measurement in Context, *Behaviour & Information Technology,* 13(1 & 2), pp. 132 - 145

BIUSEM Consortium Partners and U.C.L. (1995), *Benefits of Integrating Usability and Software Engineering Methods,* Technical Report ESSI Project 10290

Blomberg, J. L. and Henderson, A. (1990), Reflections on Participatory Design: Lessons from the Trillium Experience, In *Proceedings of Human Factors in Computing Science CHI'90*, ACM Press

Breenbaum, J. and Kyng, M. (1991), *Design at Work : Co-operative Design of Computer Systems*, Lawrence Erlbaum Associates, Hillsdale New Jersey

Buxton, W. (1986), There's More to Interaction Than Meets the Eye: Some Issues in Manual Input, In *User Centred System Design*, (Eds, Norman, D. A. and Draper, S. W.), Lawrence Erlbaum Assoc., London, pp. 319 - 337

Castells, P., Szekely, P. and Salcher, E. (1995), *Declarative Models of Presentation*, 1995,

Chin, J., Diehl, V. and Norman, K. (1988), Development of a tool measuring User Satisfaction of the Human Computer Interface, In *Proceedings of Human Factors in Computing Science CHI'88*, pp. 213-218

Clarke, D. (1996), Noddy's Guide to...Interface Development Environments, *Human Computer Interaction Interfaces*, Spring 1996, pp. 6 - 9

Clement, A. and Besselaar, P. V. D. (1993), A Retrospective Look at PD Projects, *Communications of the ACM*, **36**(4), pp. 29 - 37

Curtis, B., Krasner, H. and Iscoe, N. (1988), A Field Study of the Software Design Process for Large Systems, *Communications of the ACM*, **31**(11), pp. 1268 - 1287

Draper, S. W., Waite, K. W. and Gray, P. D. (1990), Alternative Bases for Comprehensibility and Competition for Expression in an Icon Generation Tool, In (Ed, Diaper, D.) *Proceedings of Human-Computer Interaction - INTERACT'90*, Elsevier Science Publishers B.V. North Holland, pp. 473 - 477

Freeman-Benson, B. N. and Borning, A. (1992), Constraint Imperative Programming Languages for Building Interactive Systems, *Languages for Developing User Interfaces*, (Ed, Myers, B.), Jones & Bartlett, pp. 161 - 181

Gosling, J. and Yellin, F. (1996), *The Java Application Programming Interaction: Window Toolkit and Applications*, Addison Wesley Publishing Company, Reading, MA

Gray, P., Cooper, R., Kennedy, J., McKirdy, J., Barclay, P. and Griffiths, T. (1998), A Lightweight Presentation Model for Database User Interfaces, In *Proceedings of ERCIM'98*, Stockholm, October 1998

Gray, P. D. and Draper, S. W. (1996), A Unified Concept of Style and its Place in User Interface Design, In *Proceedings of People and Computers XI - HCI'96*, London, August 1996, Springer-Verlag, pp. 49 - 62

Gray, P. D., Waite, C. A. and Kilgour, A. C. (1990a), Do-It-Yourself Iconic Displays, In (Ed, Diaper, D.) *Proceedings of Human-Computer Interaction - INTERACT'90*, Elsevier Science Publishers B.V. North Holland, pp. 645 - 650

Gray, P. D., Wood, C. A. and Kilgour, A. C. (1990b), Localisation of Application Knowledge in Incremental Development of User Interfaces, In (Ed, al, D. D. e.) *Proceedings of Human-Computer Interaction - INTERACT'90*, Elsevier Science Publishers B.V. North Holland, pp. 645 - 650

Green, T. R. G. and Petre, M. (1996), Usability Analysis of Visual Programming Environments : a 'cognitive dimensions' framework, *Journal of Visual Languages and Computing*

Griffiths, T., J.Barclay, P., McKirdy, J., Paton, N. W., Gray, P. D., Kennedy, J., Cooper, R., Goble, C. A., West, A. and Smyth, M. (1999), Teallach: A Model-Based User Interface Development Environment for Object Databases, In (Eds, Paton, N. W. and Griffiths, T.)

Proceedings of User Interfaces to Database Intensive Systems UIDIS'99, Edinburgh, 5- 6th September, IEEE Press, pp. 86 - 96

Griffiths, T., McKirdy, J., Forrester, G., Paton, N., Kennedy, J., Barclay, P., Cooper, R., Goble, C. and Gray, P. (1998), Exploiting Model-Based Techniques for User Interfaces to Databases, In (Eds, Ioannidis, Y. and Klas, W.) *Proceedings of Visual Database Systems - VDB4*, L'Aquilla, Italy, 27-29 May 1998, Chapman and Hall, pp. 21 - 46

Hammond, R. and McCullagh, P. S. (1991), *Quantitative Techniques in Geography*, Clarendon Press, London

Hemmje, M. (1994), Lyberworld - A 3D Graphical User Interface for Fulltext Retrieval, In *Proceedings of SIGIR'94*, Dublin, July 3 - 6

Hix, D. (1990), Generations of User-Interface Management Systems, *IEEE Software,* September, pp. 77 - 87

Huff, D. (1991), *How to Lie With Statistics*, Penguin Books Ltd, London

Hutchins, E. L., Hollan, J. D. and Norman, D. A. (1986), Direct Manipulation Interfaces, In *User Centred System Design*, (Eds, Norman, D. A. and Draper, S. W.), Lawrence Erlbaum Assoc, London, pp. 87 - 123

Karat, J. (1992), Software Evaluation Methodologies, In *Handbook of Human-Computer Interaction*, (Ed, Helander, M.), Elsevier Science Publishers North Holland, Amsterdam, pp. 891 - 903

Kazman, R., Bass, L., Abowd, G. and Webb, M. (1994), SAAM : A Method for Analysing the Properties of Software Architectures, In *Proceedings of 16th International Conference on Software Engineering*, Sorrento, Italy, May, pp. 81 - 90

Kumar, H. P., Plaisant, C. and Shneiderman, B. (1995), Browsing Hierarchical Data with Multi-Level Dynamic Queries, *International Journal of Human Computer Studies,* **46**(1), pp. 103 - 124

Laurel, B. K. (1986), Interface as Mimesis, In *User Centred System Design*, (Eds, Norman, D. A. and Draper, S. W.), Lawrence Erlbaum Assoc, London, pp. 67 - 85

Leventhal, L., Teasley, B., Instone, K. and Stone, D. (1995), Searching Without a Keyboard in a Multimedia Environment, In (Eds, Nordby, K., Helmensen, P. H., Gilmore, D. J. and Annesen, S. A.) *Proceedings of Human-Computer Interaction - INTERACT'95*, 1995, Chapman & Hall, pp. 241 - 246

Luo, P., Szekely, P. and Neches, R. (1993), Management of Interface Design in Humanoid, In *Proceedings of InterCHI'93*, Amsterdam, April 24 - 29

MacEachren, A.M. (1995), *How Maps Work*, The Guilford Press, New York

McKirdy, J. (1999a), Choosing the UI Tool Which Best Suits Your Needs, In (Eds, Brewster, S., Cawsey, A. and Cockton, G.) *Proceedings of Human Computer Interaction INTERACT'99*, Edinburgh, 30th August - 3rd September, IFIP, pp. 49 - 50

McKirdy, J. (1999b), *S.U.I.T. - A Framework & Methodology for the Selection of User Interface Development Tools Based on Fitness Criteria,* Technical Report TR-1999-34, University of Glasgow, May 1999, pp. 79

Miller, S. (1996), *Experimental Design and Statistics*, Routledge, London

Mirel, B. (1999), Usability Test Results for Information Visualisations: Determinants of Usefulness for Complex Business Problems, In (Eds, Sasse, M. A. and Johnson, C.) *Proceedings of Human-Computer Interaction INTERACT'99*, Edinburgh, 30th August - 3rd September 1999, IFIP, pp. 70 - 78

Monk, A., Wright, P., Haber, J. and Davenport, L. (1993), *Improving Your Human-Computer Interface: A Practical Technique*, Prentice Hall, London

Myers, B. A. (1989), User-Interface Tools: Introduction and Survey, *IEEE Software,* January, pp. 15 - 23

Myers, B. A. (1993), *Why are Human-Computer Interfaces Difficult to Design and Implement?*, Technical Report CMU-CS-93-183, Computer Science Department, Carnegie Mellon University, July 1993, pp. 15

Myers, B. A. (1996), User Interface Software Technology, *ACM Computing Surveys,* **28**(1), pp. 189 - 191

Neave, H. R. (1989), *Statistics Tables*, Unwin Hyman, London

Norman, D. A. (1986), Cognitive Engineering, In *User Centred System Design*, (Eds, Norman, D. A. and Draper, S. W.), Lawrence Erlbaum Assoc, London, pp. 31 - 61

Olsen, D. R. (1992), *User Interface Management Systems: Models and Algorithms*, Morgan Kaufmann Publishers, San Mateo

Perlman, G. (1988), Software Tools for User Interface Development, In *Handbook of Human-Computer Interaction*, (Ed, Helander, M.), Elsevier Science Publishers B.V. North Holland, Amsterdam, pp. 819 - 833

Roberts, D., Berry, D., Isensee, S. and Mullaly, J. (1998), *Designing for the user with OVID: Bridging User Interface Design and Software Engineering*, Macmillan Technical Publishing, Indianapolis, IN

Shaw, M. (1996), Some Patterns for Software Architectures, In *Pattern Languages of Program Design*, (Eds, Vlissides, J., Copliem, J. and Kerth, N.), Addison-Wesley, USA, pp. 255 - 269

Shneiderman, B. (1992), *Designing the User Interface : Strategies for Effective Human-Computer Interaction*, Addison-Wesley Publishing Company, USA

Shneiderman, B. (1994), Dynamic Queries for Visual Information Seeking, *IEEE Software*, November, pp. 70 - 77

Stone, M., Fishkin, K. and Bier, E. (1994), The Movable Filter as a User Interface Tool, In *Proceedings of Human Factors in Computing Systems (CHI'94)*, Boston MA, April 24 - 28, pp. 306 - 312

Thimbleby, H. (1985), Failure in the Technical User-Interface Design Process, *Computers and Graphics*, **9**(3), pp. 187 - 193

Tweedie, L., Spence, R., Williams, D. and Bhoghal, R. (1994), The Attribute Explorer, In *Proceedings of Human Factors in Computing Systems CHI'94*, Boston, ACM Press

Wasserman, A. I., Pircher, P. A., Shewmake, D. T. and Kersten, M. L. (1986), Developing Interactive Information Systems with the User Interface Engineering Methodology, *IEEE Transactions on Software Engineering*, **12**(2), pp. 326 - 345

Winograd, T. and Flores, F. (1987), *Understanding Computers and Cognition: A new foundation for design*, Ablex Publishing, Norwood, NJ

WEB SITES

Myers, B. (1999), *User Interface Software Tools*, Brad Myers, Human Computer Interaction Institute, Carnegie Mellon University, Pittsburgh, 1999, http://www.cs.cmu.edu/afs/cs.cmu.edu/user/bam/www/toolnames.html

Senay, H. and Ignatius, E. (1998), *Rules and Principles of Scientific Data Visualisation*, 1998, http://www.education.siggraph.org/materials/hypervis/percept/visrules.html

APPENDIX A: QUESTIONNAIRE USED IN INDUSTRIAL SURVEY

Part 1 - Contact Details

§1.1 Company details

Company Name:

Company Address:

Central Tel. N°:

Central Fax. N°:

Central E-mail address:

Web URL:

§1.2 Personal Contact details

Name:

Tel. N°:

Fax. N°:

E-mail address:

Job title:

Short description of rôle in company:
(e.g. responsibilities etc)

Qualifications/ Experience:
(in particular Human Factors/HCI experience /training/background)

Part 2 - Company & Project Profiles

§ 2A Company

The following questions aim to ascertain the scale and nature of both your organisation and of the work undertaken therein.

2.1 Size of organisation

Please tick the appropriate box for the number of employees within your organisation who are involved (in any way) with software development

employees	☐
10 < employees	☐
50 < employees	☐

2.2 Size of projects : How large (lines of code) are typical projects ?

Please select as many of the boxes as necessary

lines	☐
10,000 < lines	☐
100,000 < lines	☐

2.3 Approximately how many projects will run concurrently ? ☐

2.4 What is the average lifetime of a typical project ? ☐ months

§ 2B Projects

Please identify **one** category of application which is commonly developed within your organisation.
Given the nature of the questions which follow in the remainder of this section and the next, it would be easiest if you were to select one of the application domains with which you are involved (although this is not essential).

2.5 Generic name for the selected application category/domain.

Please identify a generic (catch all) name for your selected application domain.

2.6 Application sizes within this domain

Please indicate the approximate size of (typical) applications within this domain

	10,000 - 100,000	> 100,000	
☐	☐	☐	lines

2.7 Criticality of the applications

Would you class the typical applications within the identified domain as safety critical ?

Yes	No
☐	☐

2.8 Locus of control

Which of the following best describes the locus of operational control within typical applications for the identified domain? (Please tick the appropriate box)

Autonomous user served by system: user decides the task and pace (eg research systems, public information systems)	☐
Autonomous user with constraints: user constrained by law and policy (eg administrative systems, case handling systems)	☐
"Operator" serving the system: user paced and constrained by laws of physics (eg process control systems, manufacturing systems)	☐

2.9 Platform

Are typical applications within the identified domain single or cross platform ?

Single	Cross

Please give a brief description of the typical platform for which the applications are built:

2.10 Data types

For the identified domain, please indicate the predominant data types for the typical applications
Please tick as appropriate

Text:	
Multi-media: (combination of video, sound, graphics etc)	
Diagrammatic graphics: (eg for CAD tools)	
Other: (please specify)	

2.11 Intensity of data & interaction

a) Are applications within the identified domain data intensive ?

Yes	No	Not sure

b) Are applications within the identified domain interaction intensive ?

Yes	No	Not sure

c) What is the average response time required for applications in this domain ? _____ seconds

d) What is the average critical load for applications in this domain ? _____ people
[where critical load = number of operators using the system simultaneously]

2.12 Cognitive load

Please indicate the cognitive load for typical applications within the identified domain

High	Medium	Low	Don't know

[where cognitive load = amount of mental effort required of the user when interacting with the system - for example memory demands related to actions]
[e.g. high cognitive load would be situations where the user is required to remember a large number of menu commands; in contrast, low cognitive load would be interfaces where the user was not expected to remember any information related to operational commands (all such information being visible on the interface).]

Part 3 - Work Practice

Part 3 of this questionnaire is concerned with the work practice associated with software development in your organisation. Please answer all the following questions with respect to the application domain identified in Part 2.

§ 3A Qualifications of Team Members Responsible for HCI

[HCI = Human Computer Interaction]

3.1 Team Sizes

a) How many people constitute a typical entire project team within this domain ? ☐ people

b) How many people from the overall project team are involved with HCI ? ☐ people

3.2 With reference to a typical team for this domain, what qualifications do those responsible for **HCI** possess ? Please indicate the number of people with the following training.

No formal computing training	☐
Formal computing training but with no HCI training	☐
Graphic Design	☐
Computing science based HCI training	☐
Non-computing science based HCI training [e g. cognitive psychology]	☐
Other (please specify)	_____

§ 3B General Team/Work Allocation

For projects within this application domain, what is the general division of labour amongst the team members ? In particular, how is the division decided ? How is responsibility allocated ?

3.3 For a typical project within the identified domain, are all team members involved at every stage during development ?

Yes	No
☐	☐

3.4 If the answer to the above question is *no* please identify the structure of sub-teams and indicate the number of team members involved in each. In particular please indicate if there is a sub-team dedicated to user interface (UI) evaluation

_____ ☐ people

_____ ☐ people

_____ ☐ people

_____ ☐ people

_____ ☐ people

3.5 Do the people responsible for UI development also develop the underlying application functionality for typical applications in this domain ?

Yes	No
☐	☐

3.6 How would *you* rate the importance of UI construction compared to construction of the underlying functionality ? (Please circle the appropriate rank below)

Less important than underlying functionality			More important that underlying functionality	
5	4	3	2	1

§ 3C Design v. Implementation

3.7 Is the **design** of the UI separated from the **design** of the rest of the application for typical application development within the identified domain ? Yes ☐ No ☐

3.8 If the answer to 3.7 was *yes*, is there further separation of **Graphic Design** from the rest of the UI design ? Yes ☐ No ☐

3.9 Is the **implementation** of the UI separated from the **implementation** of the rest of the application from typical applications within the identified domain ? Yes ☐ No ☐

3.10 Is the person responsible for designing the UI also responsible for implementing his/her design for typical applications within the identified domain ? Yes ☐ No ☐

3.11 If the answer to 3.10 is *no* please tick one (or more) of the following boxes to indicate how the design is communicated from the designer to the implementor for applications within the identified domain.

Communicated orally ☐
Written specification ☐
Working prototype ☐
Other (please specify) _____

§ 3D Rôle and Nature of HCI

3.12 At which point(s) during the construction of a software application do you think HCI is appropriate ? (Please tick the relevant box)

After the underlying functionality has been specified ☐
After the underlying functionality has been implemented ☐
At all stages throughout the software development process ☐
Not at all ☐
Other (please specify) _____

3.13 What percentage of overall design and development time is devoted to the UI for typical applications within the identified domain ? ☐ %

3.14 Please tick the appropriate box(es) below for features common to user interfaces within this application domain.

a) Output Media:
Text ☐
2D images ☐
3D images ☐
Pre-built animation [e g. cartoons] ☐
Video ☐
Sound : simple [e.g beeps] ☐
high quality (non-speech) ☐
speech ☐
Other (please specify) _____

b) Input Device :
Speech ☐
Keyboard ☐
Mouse ☐
Joystick ☐
Touch Screen ☐
Other (please specify) _____

c) User style :
Expert ☐
Novice ☐
Both ☐

§ 3E Interface Specific Software In Use

When developing the user interface for a software application many people (or organisations) make use of various types software support. Such support ranges from graphics editors to full application development environments. The aim of the following questions is to identify which (if any) of the available software support (and in particular that used for interface development) is used by your company during the course of application development within this domain.

3.15 Please tick the appropriate box(es) to indicate which of the available software support (if any) is used. Please also indicate the percentage of the total application code which is generated through use of the specific software support, and answer the accompanying questions.

a) ☐ No software support used (everything written "from scratch") ☐100☐ % of total code

Please specify the implementation language used: _____

b) ☐ Pre-provided libraries of routines ☐ % of total code
[e.g. C++/Java classes libraries]

Please specify: _____

Why was this support chosen ? _____

c) ☐ Interface Builders ☐ % of total code
[software which supports the design & development of the UI but which does not support the implementation of the underlying application - e.g. Sun's Developers Guide]

Please specify: _____

Details of interface builders
Why was this support chosen ? _____

Were you given demos of the product prior to purchase ? Yes ☐ No ☐

Was systematic evaluation of the software undertaken prior to purchase ? (Please specify) _____

What were the estimated costs in terms of purchasing the software & training staff ? _____

What proportion of the available functionality is used ? ☐ % used
Comment : _____

Are there aspects of inefficiency or missing functionality within the software ? _____

Please identify aspects of the software which either impeded or greatly assisted your progress ? _____

Who is/are the principle user(s) of the software ? _____

Are you considering changing the support software ? _____

How often do you review the software support ? _____

d) ☐ Application software development environments ☐ % of total code
[software which supports the design/development of the UI in addition to that of the core application functionality - e.g Borland's Delphi, Visual C++ etc]

Please specify: _____

Details of application software development environments

Why was this support chosen ? _____

Were you given demos of the product prior to purchase ? Yes ☐ No ☐

Was systematic evaluation of the software undertaken prior to purchase ? (Please specify) _____

What were the estimated costs in terms of purchasing the software & training staff ? _____

What proportion of the available functionality is used ? ☐ % used

Comment : _____

Are there aspects of inefficiency or missing functionality within the software ? _____

Please identify aspects of the software which either impeded or greatly assisted your progress ? _____

Who is/are the principle user(s) of the software ? _____

Are you considering changing the support software ? _____

How often do you review the software support ? _____

e) ☐ Others ☐ % of total code

Please specify _____

3.16 Were you satisfied with the current tool evaluation methods ? (where applicable) Yes ☐ No ☐

3.17 Would you like to have access to better evaluation tools/methods for evaluation of software support ? Yes ☐ No ☐

§ 3F User Interface Design Process

This last selection of questions aims to identify the development process employed within your company for design and implementation of the **user interface** to applications within this domain. It is hoped that these questions will uncover what process is adopted in **reality** rather than that which is stipulated in academia, so please answer factually !

When developing a user interface, which of the following steps are undertaken ? Please note that these steps are **not** listed in order.

- **Please indicate whether the step is performed during your development process by ticking the first box.**
- **Please indicate the order in which the steps are performed by allocating a sequence number to each step.**

** *Alternatively, use the space below to draw a diagram of your design process.*

- Please feel free to annotate the steps in any way - especially if to indicate iteration or combination of steps.

	Performed [✗/✓]	Sequence N°		
3.18			· **rapid prototyping**	
			Who is responsible for this step ?	
			What are his/her qualifications ?	
			What role (if at all) does the software support play in this step ?	
			· **user based evaluation of the interface** : assessment of the interface is the responsibility of the user who is free to provide whatever feedback he/she deems appropriate - ie only loosely structured process.	
			Who is responsible for this step ?	
			What are his/her qualifications ?	
			What role (if at all) does the software support play in this step ?	
			· **formative user-based evaluation** : evaluation of the interface design *as it is being developed* - the results of the evaluation being fed back into the design at early stages.	
			Who is responsible for this step ?	
			What are his/her qualifications ?	
			What role (if at all) does the software support play in this step ?	
			· **systems analysis**	
			Who is responsible for this step ?	
			What are his/her qualifications ?	
			What role (if at all) does the software support play in this step ?	
			· **user interface *software* design** : similar to design of any other software - system-centered.	
			Who is responsible for this step ?	
			What are his/her qualifications ?	
			What role (if at all) does the software support play in this step ?	
			· **user-based testing of the interface software** : user given specific structured tasks to do which are monitored by the developer in order to assess specific aspects of the interface design.	
			Who is responsible for this step ?	
			What are his/her qualifications ?	
			What role (if at all) does the software support play in this step ?	
			· **user interface software implementation**	
			Who is responsible for this step ?	
			What are his/her qualifications ?	
			What role (if at all) does the software support play in this step ?	
			· **user interface *interaction* design** : design of the user actions, feedback, screen appearance, user tasks, functionality sequencing, content, information access, design of interface objects, screen layout and interaction styles - user centered.	
			Who is responsible for this step ?	
			What are his/her qualifications ?	
			What role (if at all) does the software support play in this step ?	
			· others - please state	

3.19 Please indicate the nature of the use of prototypes within the development process for applications in the identified domain. [Please note that the categories are **not** mutually exclusive]

Horizontal Prototypes (incorporates a large number of features with little functional depth)	
Vertical Prototypes (small number of functions are prototyped in detail allowing realistic usability evaluation)	
Global Prototypes (prototype of much of the system from which user can get good idea of final product)	
Local Prototypes (prototype of single specific details - for evaluation of alternatives for isolated interaction)	
Evolutionary Prototypes (prototype from iteration n evolves to become prototype for iteration n+1)	
Revolutionary Prototypes (prototype discarded between iteration n & n+1)	
Paper Based Prototypes (i.e. simple sketches of the proposed look of the interface)	

3.20 Please describe the method of completion used for steps in the development process ?

_____

3.21 Is use made of versioning mechanisms ? Yes ☐ No ☐

Please specify: _____

3.22 Between steps in the design process is there some form of "signing-off" procedure ? Yes ☐ No ☐

Please specify: _____

3.23 Why is this particular design process followed ? _____

_____

3.24 Were HCI specialists involved in the establishment of the design process ? Yes ☐ No ☐

Thank you VERY much for taking the time to fill out this questionnaire. Should you wish to make any additional comments please use the space provided on the next page

Comments

Question N°	Comment

APPENDIX B: SUIT FRAMEWORK & METHODOLOGY

SUIT

A Framework & Methodology For The *S*election Of *U*ser *I*nterface Development *T*ools Based On Fitness Criteria

1 INTRODUCTION

SUIT is a framework and associated methodology for the selection of user interface development tools.

Figure 1 shows SUIT in the context of its potential usage, highlighting the various components which input to, and are the output from, the evaluation and selection mechanism.

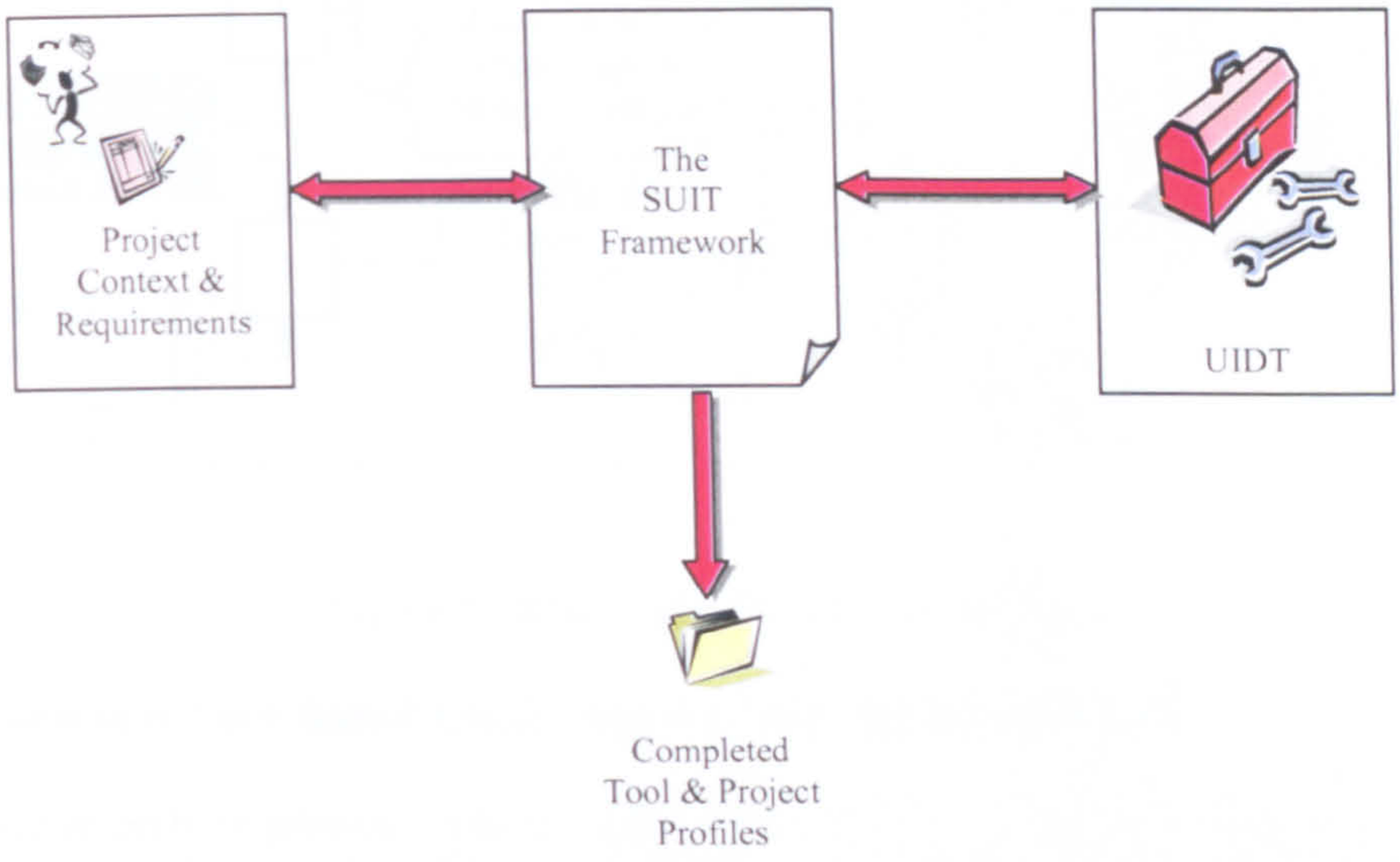


Figure 1 – *The SUIT Framework in context*

SUIT adopts a reference model-based approach to tool selection, and can be used in three different ways. These are:

- to select a UIDT based on a generic comparison of tools;
- to select the 'best-fit' UIDT for an unprecedented[1] project based on the specific context and requirements of that project;
- to identify an appropriate UIDT for a specific project based on comparisons with previous projects.

The applicability of each approach has three determining factors: the stage of design and/or development of the project; the precedence of the project; and the intended specificity of the outcome of the use of SUIT.

Each approach demands that the SUIT framework be manipulated in a different manner. This is reflected in the three different routes through the SUIT methodology which are shown in Figure 2.

Consider the SUIT framework as a reference model of *everything* which could be included in a UIDT[2]. Guided by the appropriate methodological thread, the SUIT framework is tailored[3] to provide a pilot for data-collection and thereafter a context for the interpretation of that data. The degree to which the framework *can* be tailored is determined by the amount of information which is available to the evaluator at the time of using SUIT. The degree to which the framework *is* tailored is dependent on the amount it can be tailored and the intended style of use of SUIT.

---

[1] In SUIT, an "unprecedented" project refers to a project which has no precedent within the context of a specific organisation. It does not refer to the general field of computing.

[2] The SUIT framework is extensible to allow inclusion of references to evolving technology and concepts within the framework, thus ensuring SUIT is up-to-date.

[3] Tailoring constitutes making appropriate selections of and discarding unnecessary features within the framework – see later sections for concrete tailoring information.

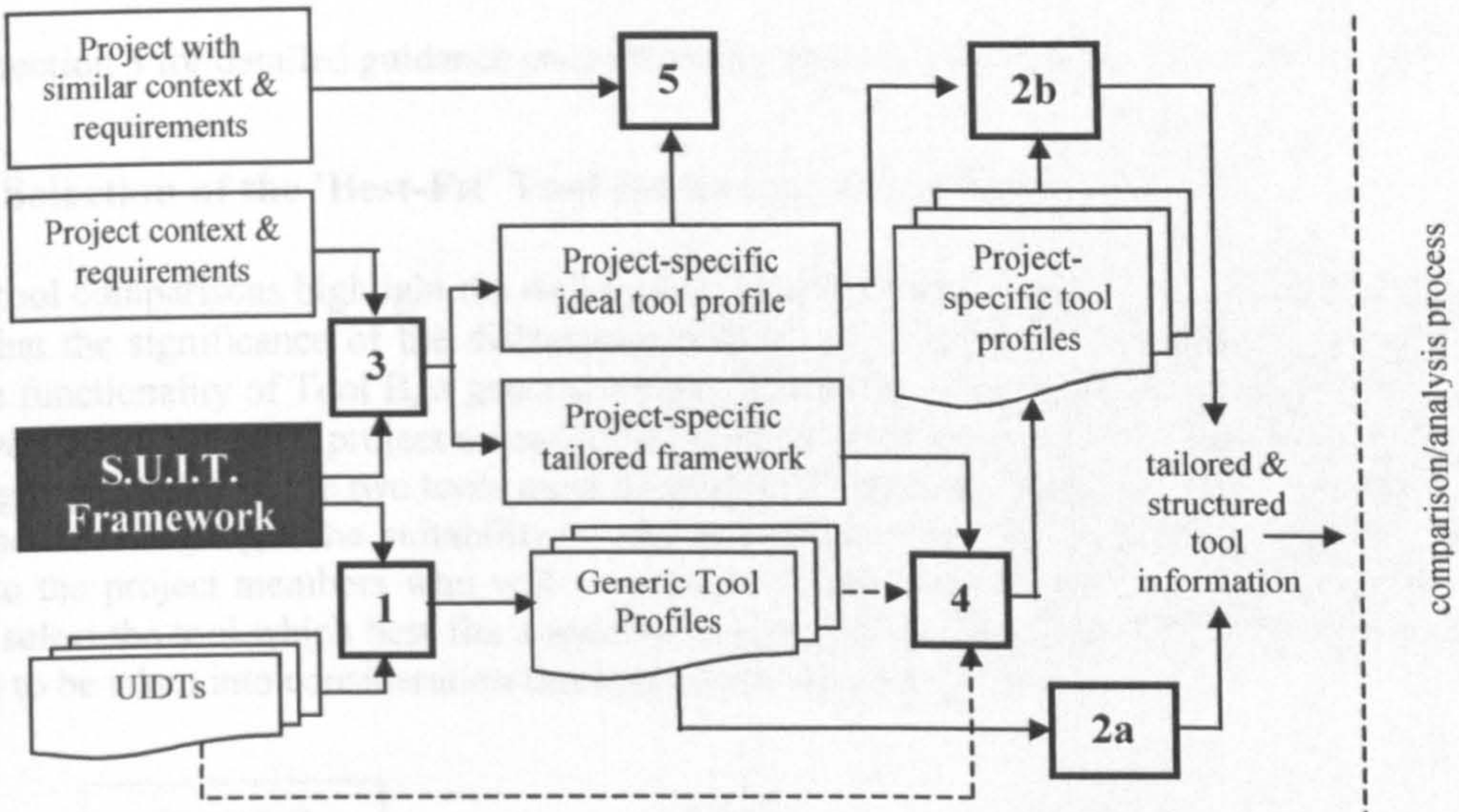


Figure 2 – *Route map of the SUIT methodology*

1.1 Selection of a Tool Based On a Generic Comparison of Tools

SUIT can be used to perform generic – that is, non project-specific – comparisons of tools. A comparison of this nature is done outwith the context of a project and so no knowledge of the project or the context in which the project is taking place is required.

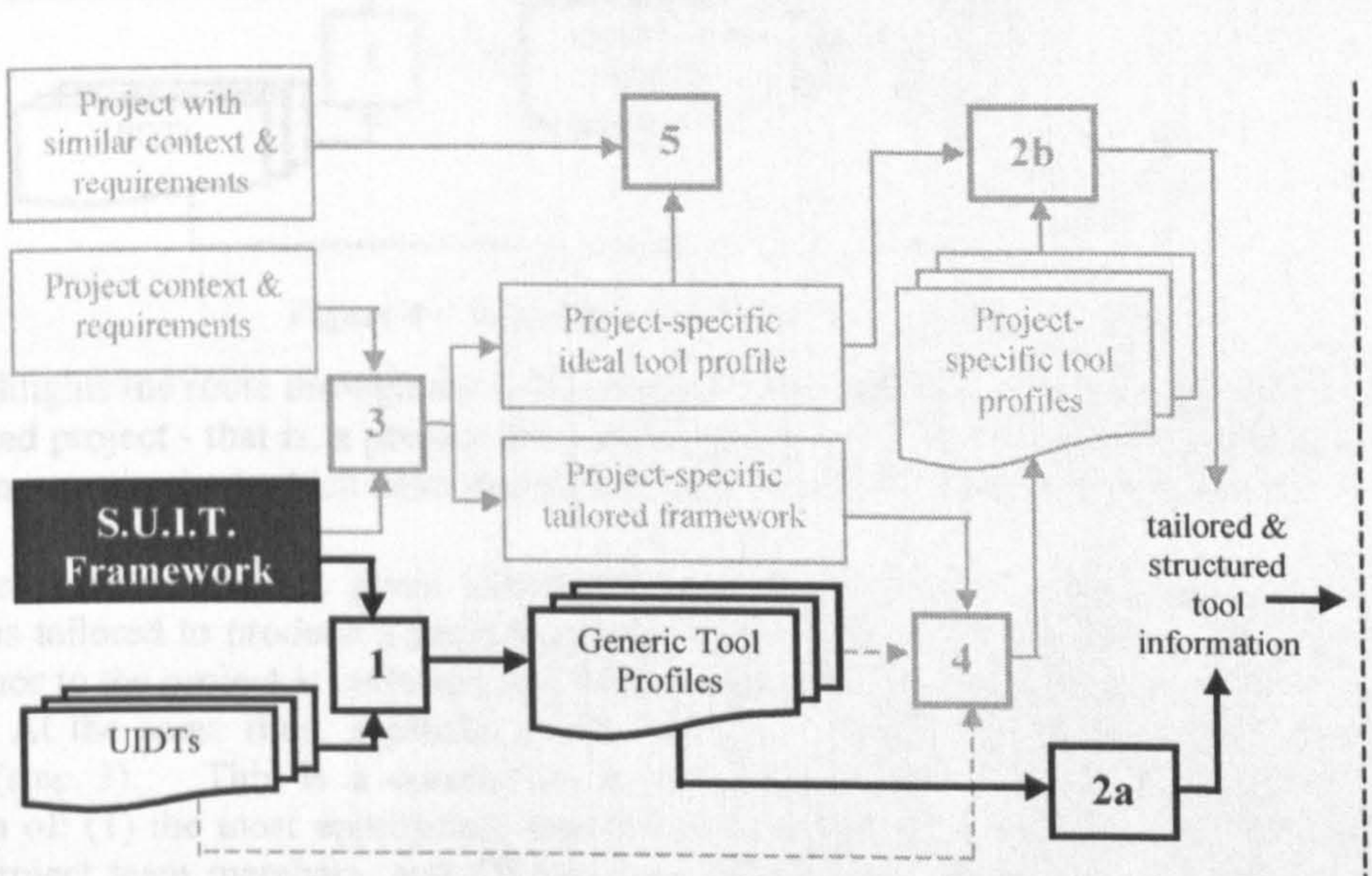


Figure 3 – *Performing a generic comparison of tools*

Figure 3 highlights the path which would be taken through the SUIT methodology in order to perform a generic comparison of tools.

Essentially, the SUIT framework is adopted as a complete reference model. In other words, every component in the framework is an active criteria for the comparison of the tools.

At point 1 in Figure 3, each tool is examined in turn to ascertain which of the components listed in the SUIT framework it meets and those it fails to meet. For each tool, the information recorded during this process effectively forms a generic profile of the individual tool. That is, a profile of the tool in relation to the entire SUIT framework (or reference model) which is not influenced by the specifics of a project.

The generic tool profiles can then be compared (step 2a) in order to evaluate them with respect to each other in the context of the SUIT framework. From this comparison, it is possible to derive a tool recommendation.

Refer to section 3 for detailed guidance on performing generic tool comparisons using SUIT.

1.2 Selection of the 'Best-Fit' Tool for an Unprecedented Project

Generic tool comparisons highlight the differences between various tools, but it is in the context of a specific project that the significance of the differences becomes apparent. For example, if Tool A provides three times the functionality of Tool B, a generic comparison of the tools is likely to rank Tool A higher than Tool B. However if, for a given project a specific set of functionality is required, the generic comparison does not *easily* highlight which of the two tools most accurately meets these requirements. Further, a project-specific comparison can highlight the suitability of the interaction style and assistance afforded by the tools with respect to the project members who will be using the tools, and the tasks to be performed. Therefore, in order to select the tool which best fits a specific project, the project's functional requirements and context of use have to be taken into consideration throughout the selection process.

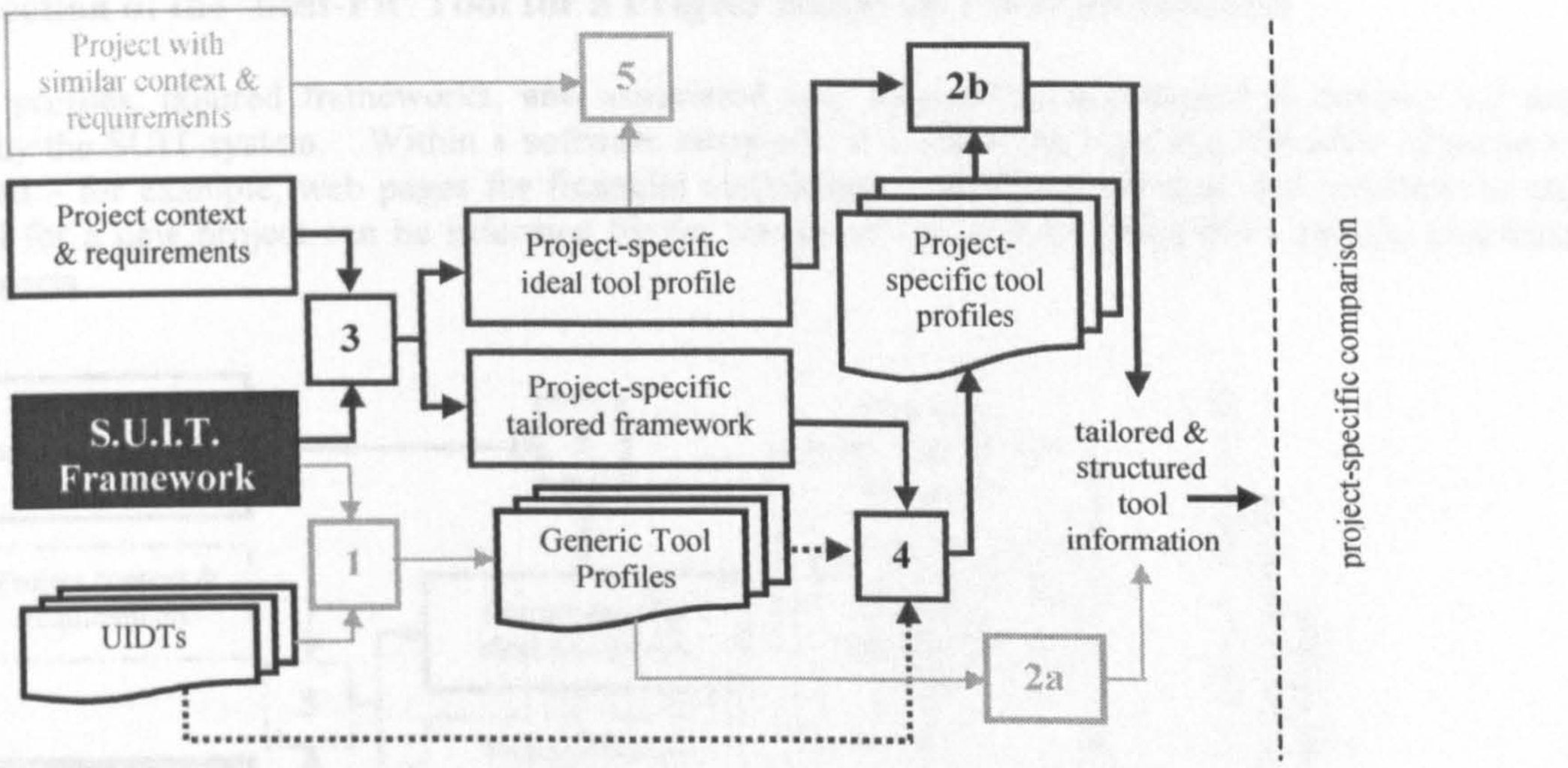


Figure 4 – *Selecting a tool for an unprecedented project*

Figure 4 highlights the route through the SUIT methodology which is adopted to select the best-fit tool for an unprecedented project - that is, a project for which there have been no similar preceding projects (within the context of an organisation) which have themselves used the SUIT system to select a tool.

For an unprecedented project, given information about the project context and requirements, the SUIT framework is tailored to produce a project-specific tailored framework which ensures that only information of significance to the project is collected and then compared when examining the various tools - see step 3 in Figure 4. At the same time, a profile of the *ideal* tool, given the project requirements and context, is developed (step 3). This is a correlation of the functionality which the tool must support against a combination of: (1) the most appropriate interaction mechanisms as indicated by the available information about the project team members; and (2) the most appropriate interaction mechanisms for the tasks to be performed using the tool. It is against this picture of the project-specific ideal tool that the *actual* tools are compared. The stage of design and/or development reached by the project will determine the accuracy or level of detail at which the evaluator can tailor the framework and therefore generate the ideal tool profile. It is recommended that SUIT be used for project-specific tool selection at the point during design and development when user interface requirements are relatively well defined – for example, when it is known which user interface features will (or are likely to) be included - and when the project team has been finalised. This will ensure that the tailoring can be very specific, accurate, and decisive.

However, SUIT can be used before this level of knowledge is available to generate an initial short list of potentially suitable tools based on whatever information *is* available. As more information becomes available, or as more detail is finalised, an iterative process of framework tailoring and tool short-listing can be performed.

A project-specific tailored SUIT framework, which is essentially a project-specific reference model, is used to pilot the collection of (only relevant) information about the various tools being considered for selection. In Figure 4, step 4 shows that the information can be sourced in one of two ways: it can either be extracted (or filtered) from existing generic tool profiles (see section 1.1); or it can be collected via direct examination of the actual tools. In either case, the result is a series of project-specific tool profiles - that is, profiles of the tools with respect to only those features which are of interest or relevance to the given project.

The project-specific tool profiles are then compared to the project-specific ideal tool profile - see step 2b. In other words, each *real* tool's functional provision with respect to the interaction mechanisms it uses is compared to the picture of the *ideal* set of functionality and corresponding interaction mechanisms in order to identify which tool is the closest to the ideal. During this comparative selection process, the interaction assistance, cognitive demands, quality of feedback and other miscellaneous observations about the tools are taken into consideration in order to make possible the recommendation of the 'best-fit' tool for the given project.

Refer to section 4 for detailed guidance on selecting a UIDT for an unprecedented project using SUIT.

1.3 Selection of the 'Best-Fit' Tool for a Project Based on Previous Projects

The project profiles, tailored frameworks, and associated tool information mentioned in section 1.2 are maintained by the SUIT system. Within a software company, it is often the case that 'families' of projects are developed – for example, web pages for financial institutions. If this is the case, the selection of the 'best-fit' tool for a new project can be informed by the results of the tool selection from closely matching previous projects.

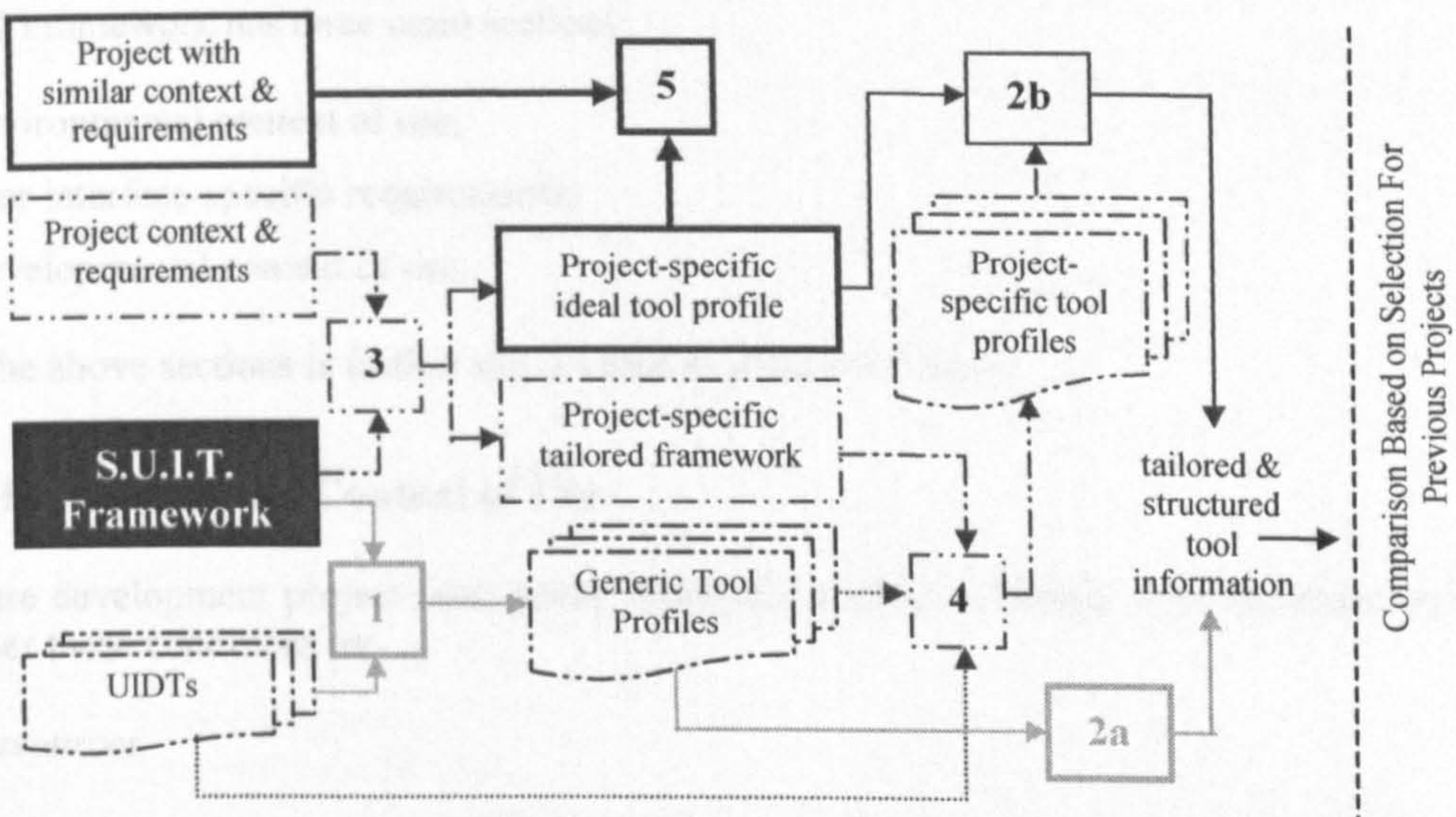


Figure 5 – *Selecting a project-specific UIDT where selection information is available for similar preceding projects*

Figure 5 highlights the path through the SUIT methodology which would be followed if selecting a tool for a project which has ancestors with close similarities. The new project requirements and context are examined in order to ascertain whether or not they would generate a project-specific ideal tool profile which would exactly match with an existing project-specific ideal tool profile - see step 5 in Figure 5. If this is the case, the tool recommendation as made for the previous project would also be the best-fit for the new project. For example, imagine that a web page had been developed using a tool which was selected according to SUIT. If a practically identical web page was to be developed by the same team members, then the ideal tool profile for the two projects is likely to be realistically identical and so the tool recommendation from the previous project would stand for the new project. Hence, the work done to select the most appropriate tool for the previous project can benefit the new project.

If there were only slight differences between the ideal tool profile for a new project and an existing project, the project-specific tailored framework and ideal tool profile for the existing project could simply be tweaked (that is, minor changes made) and then used to complete the selection process as described for steps 4 and 2b

in section 1.2. In this way, the ability to identify similarities between projects saves considerably on the time taken to make an effective selection of the best-fit tool.

For example, again imagine that an interface had been developed using a tool which was selected according to the SUIT methodology and framework. If a new interface was to be developed which differed only slightly in terms of the functionality it utilised and the experience of the team members, then the project-specific tailored framework and ideal tool profile for the previous project would be a good starting point for the generation of a project-specific tailored framework and ideal tool profile for the new project. They could simply be tweaked and then used to complete a project-specific tool selection which would be closely guided by the findings of the previous project. Indeed, if the two projects were close enough in context and requirements, the project-specific tool profiles for the previous project could also be used during the selection process for the new project allowing a new project to greatly benefit from the work completed for an existing project.

Given either of the above scenarios, advantage is made of the time and effort expended on tool selection for preceding projects, therefore reducing the cost of tool selection for new projects.

Refer to section 5 for detailed guidance on using SUIT to select a UIDT for a project based on similarities with previous projects.

2 THE SUIT FRAMEWORK

For familiarisation purposes prior to its use, this section describes the overall layout and organisation of the SUIT Framework.

The SUIT Framework has three main sections:

- environmental context of use;
- user interface-specific requirements;
- developmental context of use.

Each of the above sections is further sub-divided as discussed below.

2.1 Environmental Context of Use

A software development project takes place within the context of certain *environmental* conditions. SUIT categorises these conditions as:

human resources

Software is usually developed by teams of people. As individuals, the members of the team will have varied (and perhaps very specific) skills, will have differing experiences of UIDT use, and will work on varying amounts of the project as a whole. Therefore, as a group, they will inform a complex set of human related constraints which need to be carefully considered when selecting the most suitable UIDT for the specific project. SUIT provides the means by which the history or experience of each team member can be catalogued for inclusion in the selection process. In order to accommodate the differing roles of individuals in the team, their experience of rapid prototyping and of end-product implementation is treated separately. In each case, the following information is recorded:

- the interaction mechanism(s) with which the team member is familiar (including the specific programming language(s) if appropriate);
- the degree to which the team member's role in the current project is routine;
- the level of semantic knowledge the team member possesses regarding the domain for which the application is being developed;
- the level of stress under which the team member is placed whilst working on the project;

- the weight of the work load of the team member.

institutional goals or constraints

A software development company may have specific long-term institutional goals. These goals will determine levels of constraints on the purchase of a UIDT and must therefore be take into consideration during the selection process. SUIT captures constraining information under the following categories:

- the programming language which is supported by the UIDT;
- the staff training which is possible upon purchase of the UIDT;
- the universe of UIDTs from which the selection can take place;
- the financial costs involved with purchase and installation of the UIDT;
- the development platform(s) on which the UIDT must operate;
- and the speed with which the development must take place.

2.2 User Interface-Specific Requirements

The purpose for which an application is being developed and the design of the interface to that application will generate certain concrete user interface-specific requirements – for example, the visual features which are included in the user interface and the external devices with which the application must communicate. If a UIDT is to be complete in its support of the user interface development for a project, each of these requirements must be catalogued and then checked against the service provision of each potential UIDT.

SUIT imposes a hierarchical categorisation on user interface-specific aspects. The list below demonstrates the two highest levels of classification. Reference should be made to the appropriate sections of the framework in order to examine the lowest level of classification – that is, the actual concrete components or features.

The SUIT categorisation of user interface-specific aspects:

- *General Properties Of The User Interface:*
  - Screen Navigation;
  - Dialogue;
  - Active Devices;
  - Miscellaneous.
- *Input/Output Devices:*
  - Input Devices;
  - Output Devices;
  - Target Monitor;
  - Other Hardware Devices.
- *Output Presentation Types:*
  - Text;
  - Graphics;
  - Animation;
  - Audio;
  - Video;
  - Data Driven/Dynamic Objects.
- *Combinations Of Output Presentation Types:*
  - One Screen/Different Windows;
  - One Screen/Same Window.
- *User Interface Features:*
  - Boxes;
  - Menus;
  - Forms;
  - Windows;
  - Text Area.

2.3 Developmental Context Of Use

In addition to the environmental constraints mentioned in section 2.1, UIDT use is also constrained by aspects of the developmental context into which the UIDT must be integrated. These contextual aspects are primarily concerned with the integration of the UIDT into existing working practice and the ways in which the UIDT can enhance or support these practices. They include the design and development methodology adopted by the organisation for the completion of the project and the technological support which must be provided by the UIDT.

SUIT imposes a hierarchical categorisation on the developmental contextual aspects. As in section 2.2, the following list shows the highest level of classification and reference should be made to the appropriate sections in the framework in order to view the details.

The SUIT categorisation of developmental contextual aspects:

- User Interface Design & Development Steps;
- Evaluation Assistance;
- Development Aids;
- Project Management;
- Tool Characteristics;
- Quality Attributes of the Target System.

2.4 General Layout Of The SUIT Framework

The SUIT Framework is organised around tables. There is a separate table for each of the categories discussed in sections 2.1 to 2.3. The rows in the tables correspond to the various components within each category. The label for each of these components is preceded by a small box which, when checked (☑), flags inclusion of the component within the tailored framework. The columns in the tables display the criteria against which the components should be measured.

With the exception of the environmental contextual components, all tables are preceded by a single row which can be used to:

- summarise the data across all rows in the table;
- compare the collected data at the level of the summary rather than the individual components.

The tables on pages I and II deal with the environmental contextual aspects of UIDT selection. Pages 1 to 33 cover the user interface-specific requirements, and the developmental contextual aspects are dealt with on pages 33 to 39.

For each page in the SUIT Framework there is a corresponding glossary page. These are annotated versions of the actual framework tables. They describe what is meant by every term in each table. The glossary pages are located opposite the corresponding framework page for ease of reference whilst using the framework. The glossary page numbers match those of the framework pages but are prefixed with a 'G'.

3 GENERIC COMPARISON OF UIDTS

As mentioned in section 1.1 SUIT can be used to perform a generic comparison of UIDTs. To recap on the conditions under which it is possible to perform these comparisons and the interpretative context of the results obtained, refer back to section 1.1.

In essence, the SUIT framework is used as a reference model against which the UIDTs are evaluated and the following methodology should be adopted.

Notes:

- the term **select** refers to putting a ☑ in the appropriate box;
- terms which are used in the tables are written in *italics*;
- for clarification of the meaning of any term, refer to the corresponding glossary page;
- the appropriate page number is given in brackets – e.g. [1].

3A Prepare The SUIT Framework

Complete the following steps to prepare the SUIT framework for use during data-collection.

3A.1 ENVIRONMENTAL CONTEXT OF USE

3A.1.1 HUMAN RESOURCES [I]

The absence of project-specificity means this table can be ignored.

3A.1.2 INSTITUTIONAL GOALS & CONSTRAINTS [II]

The absence of project-specificity means that this table can also be ignored. However, it may be of benefit to record some of the listed information for purposes of reference – the information will not be used directly in the UIDT comparison process.

3A.2 USER INTERFACE SPECIFIC REQUIREMENTS

The steps indicated below should be followed to include all user interface features in the prepared framework. In each of the following categories, the *Other* rows should be ignored.

3A.2.1 GENERAL PROPERTIES OF THE USER INTERFACE

☑ Select *Screen Navigation* [1]
  ☑ Select *Scrolling*; *Paging;* and *Key-Combinations*
☑ Select *Dialogue* [2]
  ☑ Select *End-User Customisable*; *Adaptive*; *Multi-Thread*; *Asynchronous*; and *End-User Interruptible*
☑ Select *Active Devices* [3]
  ☑ Select *Multiple Active Devices* and *Single Active Devices*
☑ Select *Miscellaneous* [4]
  ☑ Select *Highlighting* and *Target System Help*

3A.2.2 INPUT/OUTPUT DEVICES

☑ Select *Input Devices* [5-6]
  ☑ Select *Keyboard*; *Mouse*; *Function/Command Keys*; *Arrow Keys*; *Voice Input*; *Touch Screen*; *Graphics Tablet*; *Joystick*; *Trackball*; *Footpedal*; *Lightpen*; *Dial/Valuator*; *Button Box*; *Finger Mouse*; *Head Pointer*; *Eye Tracker*; *Finger/Hand Pointer*; *Data Glove*; *Fingerprint Scanner*; and *Iris Scanner*.
☑ Select *Output Devices* [7]
  ☑ Select *Voice Output*; *Tonal Output*; and *Print/Hard Copy*
☑ Select *Target Monitor* [8]

- ☑ Select *Standard Desktop Monitor (colour)*; *Standard Desktop Monitor (monochrome)*; *Bit Mapped/High Resolution Graphics Monitor*; *Alpha-Numeric Text Only Monitor*; *Palmtop*; and *Laptop*.

☑ Select *Other Hardware Devices* [9]

- ☑ Select *Videodisk Player*; *Videotape Player*; *CD ROM Drive*; *Audio CD Player*; *Microphone*; *External Speakers*; *Midi Controller*; and *Scanner*.

3A.2.3 OUTPUT PRESENTATION TYPES

☑ Select *Text* [10]

☑ Select *Graphics* [11-12]

- ☑ Select *Bit-Mapped*; *Vector*; *Imported Images*; *Geographical*; *Pie Charts*; *X-Y Charts*; and *3D Graphics*;

☑ Select *Animation* [13]

☑ Select *Audio* [14-16]

- ☑ Select *Audio Midi*
- ☑ Select *Audio Synthesised Speech*
  - ☑ Select *From Text Strings*; *From File*; *Male Voice*; *Female Voice*; *Natural Tone*; *Robotic Tone*; *Pitch Control*; and *Rate Control*
- ☑ Select *Audio Non-Speech Sounds*
- ☑ Select *Auditory Icons*

☑ Select *Video* [17-19]

- ☑ Select *Videodisk*; *Videotape*; *Frame Grabber*; *Gen Lock*; *Imported*; *Freeze Frame*; *Special Effects*; *Frame Editor*; *Multiple Concurrent Videos*; and *Control*.

☑ Select *Data Driven/Dynamic Objects* [20-21]

- ☑ Select *Discrete Objects*; *Bar Graphs*; *Line Graphs*; *3D Graphs*; *Pie Charts*; *Histograms*; *Rotators*; *Dials*; *Scales*; and *Text*.

3A.2.4 COMBINATIONS OF OUTPUT PRESENTATION TYPES

☑ Select *One Screen/Different Windows* [22]

☑ Select *One Screen/Same Window* [22]

3A.2.5 USER INTERFACE FEATURES

☑ Select *Boxes* [23]

- ☑ Select *List Box*; *Data Entry Box – Single Line*; *Data Entry Box – Multiple Line*; *Message Box*; *Dialogue Box*; and *Default Boxes*.

☑ Select *Menus* [24-25]

- ☑ Select *Binary*; *Extended*; *Permanent*; *Pull-Down*; *Pop-Up*; *Tear-Off*; *Embedded*; *Radio Buttons*; *Push Buttons*; *Check Buttons*; *Palette*; *Pie*; and *Default*.

☑ Select *Forms* [26-28]

- ☑ Select *Non-Enumerated Input*; *Enumerated Input*; *Formatted Fields*; *Optional Fields*; *Required Fields*; *Data Dependent Fields*; *Default Value Fields*; *Single Page Form*; and *Multiple Page Form*.
- ☑ Select *Mouse Pick*; *Arrow Keys – Bi-directional*; *Arrow Keys – Wrap Around*; *Tab Key – Bi-directional*; and *Tab Key – Wrap Around*.

☑ Select *Windows* [29-32]

- ☑ Select *Overlapping Windows*; *Tiled Windows*; *Primary Windows*; *Secondary Windows*; and *Default Standards*.
- ☑ Select *Window Transition*
  - ☑ Select *None*; *Blinds*; *Collapse To Centre*; *Diagonal Contract/Expand*; *Dissolve*; *Expanding Quadrants*; *Fade from Colour*; *Horizontal Fold/Unfold*; *Slats*; and *Wipe*.

☑ Select *Text Area* [33]

- ☑ Select *Simple Typed Input*; *Input With Required Arguments*; *Input With Optional and/or Required Arguments*; and *Input Language With Arbitrary Grammatical Structure*.

3A.3 DEVELOPMENTAL CONTEXT OF USE

☑ Select *UI Design & Development Steps* [34]

- ☑ Select *Systems Analysis*; *User Interface Software Design*; *User Interface Interaction Design*; *User Interface Software Implementation*; *Formative User Based Evaluation*; *User Based Evaluation*; and *User Based Testing of the Interface Software.*
- ☑ Select *Rapid Prototyping*
  - ☑ Select *Horizontal Prototypes*; *Vertical Prototypes*; *Global Prototypes*; *Local Prototypes*; and *Paper Based Prototypes.*
- ☑ Select *Evaluation Assistance* [35]
  - ☑ Select *Event Level Data Collection*; *Selective Data Collection*; *Data Analysis*; *Data Management*; and *Session Replay.*
- ☑ Select *Development Aids* [36]
  - ☑ Select *Code Editor*; *Interpreter*; *Compiler*; *Step Through Execution*; *Debugger*; *Property List*; *Hierarchy Editor*; *Class Browser*; and *File Comparison.*
- ☑ Select *Project Management* [37]
  - ☑ Select *Configuration/Version Control*; *Documentation*; *Design Representation*; *Data Dictionary*; *Approval/Signing Off Process*; and *Quality Control/Measurement.*
- ☑ Select *Tool Characteristics* [38]
  - ☑ Select *Reuse of Tool Output*; *Extension of Tool Features*; *Modification of Tool Features*; *Observance of Standards*; *Portability of Tool Output*; *Integration With DBMS*; and *Integration With Other Software.*
- ☑ Select *Quality Attributes Of Target System* [39]
  - ☑ Select *Modifiability of Target System Code*; *Compatibility of Target System Code*; *Reusability of Target System Code*; and *Response Time of Target Code.*

3B Gather Data

Complete the following steps *for each* of the UIDTs which are to be included in the comparison

3B.1 PRODUCE COPIES OF THE PREPARED FRAMEWORK

Generate a copy of the prepared framework (pages 0 and 1 - 39). Complete sections A and C on page 0 to record the evaluation and high-level tool details respectively.

3B.2 RECORD INFORMATION

For each component in the prepared framework [pages 1 – 39] record the following information by selecting the box(es) in the appropriate column(s):

3B.2.1 INTERACTION MECHANISM(S) USED TO INCLUDE A COMPONENT

Where applicable, the activities involved in the inclusion (or use) of a component are broken down into the instantiation of the component, the specification of its layout, and configuration of its properties. In each case, select the interaction mechanism(s) which are provided by the UIDT in order to complete the inclusion (or use) of the component.

3B.2.2 INTERACTION ASSISTANCE PROVIDED

Select the style(s) of interaction assistance which the UIDT provides to guide or inform the performance of each of the steps discussed above.

3B.2.3 COGNITIVE DEMANDS

This refers to the amount of mental effort demanded of the UIDT user whilst performing the tasks discussed above. The aim of the UIDT should be to minimise the cognitive demands. Therefore, if the level of mental effort required of the UIDT user is high, place a "-" in the first box under this heading, and record an appropriate explanatory comment. Conversely, if performance of the task requires little mental effort, place a "+" in the first box and record an explanation. Note the inverse correspondence between the symbol allocation and the level of cognitive demands. Low cognitive demands are viewed as an advantage to interaction with the UIDT and are therefore signalled with a "+" whereas high cognitive demands are a disadvantage and are marked with a "-".

3B.2.4 QUALITY OF FEEDBACK

If the UIDT provides good feedback to the user when performing each of the tasks discussed previously, place a "+" in the first of the two boxes under this heading and record an explanatory comment. If feedback is poor mark the first box with a "-" and explain why.

3B.2.5 MODIFIER/OTHER COMMENTS

This section allows miscellaneous comments to be recorded regarding the actual component or the performance of the associated task – in other words it is a catch all section for recording peculiarities (good, bad, or unusual) of the UIDT. If the UIDT does something particularly well, signal this with a "+" and indicate the nature of the advantage with a descriptive comment. In contrast, if the UIDT makes something particularly difficult to achieve or the associated component is inadequate, mark a "-" and provide an explanation.

3B.3 ADDITIONAL COLUMNS

The nature of some of the components demanded that additional columns be included in the associated tables. For each table, explanation regarding the meaning of these extra columns is provided on the affiliated glossary page. For each component (or row) in these tables, if the UIDT supports the features represented by these additional columns, the appropriate box(es) should be selected. For example, consider the table for Dialogue on page 2. Two additional columns have been introduced to record whether the specific styles of dialogue (e.g. adaptive) can have a linear or a hierarchical organisation. For each style of dialogue, if one or both of these options is available, the appropriate box or boxes should be selected.

3C Compare Results

Once the data has been gathered for each of the UIDTs, they can be compared. Given that generic comparisons are not based on an underlying set of requirements little in the way of guidance can be offered during this stage of evaluation. However, as a means to assist, a summarisation facility is provided. It is achieved by following these steps for each completed framework:

3C.1 ELIMINATE REDUNDANT INFORMATION

Strike-through every row in each table for which the UIDT makes no provision. For example, if the UIDT does not provide *Tear-Off Menus* [page 24], put a line through each of the associated rows to easily identify that the rows need not be considered during the comparisons (see Figure 6).

Figure 6 – *Elimination of redundant information*

3C.2 COMPLETE THE SUMMARY ROW

Each table is preceded by a summary row. It reflects the columns in the associated table and enables the information obtained for that category to be viewed at a glance, albeit in less detail than would be obtained by examining each row in the table in turn. The summary row is completed as follows:

3C.2.1 INTERACTION MECHANISM

For each of the columns reflecting interaction mechanisms, count the number of selections which have been made – in other words the number of rows in which the column has been selected. The interaction mechanism with the greatest count is the predominant interaction mechanism for that specific category of the UIDT. The corresponding column should be selected in the summary row. If the counts for two or more interaction mechanisms are equal or very close, each of the corresponding columns in the summary row can be selected to reflect, at a glance, the spectrum of interaction mechanisms used by the UIDT. Figure 7 shows this being done for *Screen Navigation*.

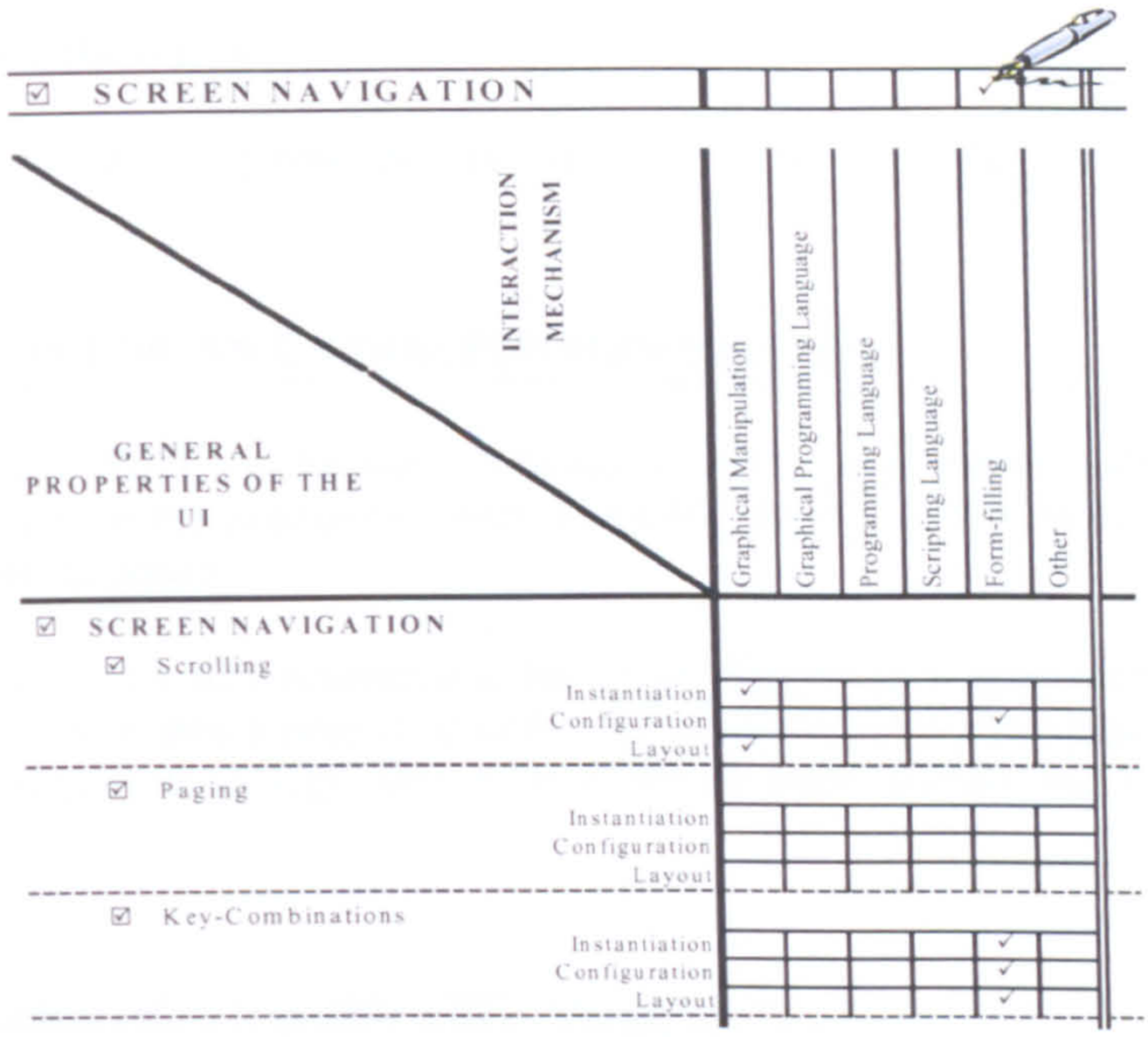


Figure 7 - *Completing the summary row*

3C.2.2 INTERACTION ASSISTANCE

For each of the columns reflecting interaction assistance, count the number of selections which have been made. Again, the column(s) with the highest count(s) indicate the most employed interaction assistance mechanism(s) for components in the associated table. Select the corresponding column(s) in the summary row.

3C.2.3 COGNITIVE DEMANDS

To summarise the cognitive demands is less straight forward than the summation process described in the previous two sections. Given that a "-" indicates a negative point concerning a component and a "+" indicates something positive, it is possible to count the number of negative and the number of positive points and reflect the greater of the two with the appropriate symbol and summative comment in the corresponding columns of the summery row (if the counts are equal, a neutral "0" can be recorded).

However, given that the severity or significance of each positive or negative aspect will not have the same weighting, this kind of summation is of limited value. Instead, the discretion of the evaluator should be called upon to weigh-up the positive and negative aspects in light of their accompanying comments and then insert the most appropriate or accurate summary in the summary row.

3C.2.4 QUALITY OF FEEDBACK

The summation of this aspect should be treated in the same way as the cognitive demands. That is, the most appropriate symbol (-,0,+) and accompanying comment should be left to the discretion of the evaluator based on the individual entries for each row in the table.

3C.2.5 MISCELLANEOUS COMMENTS

Although more difficult to summarise a collection of comments with potentially random focal points, the summation of this information should be treated in the same fashion as cognitive demands and quality of feedback.

3C.2.6 ADDITIONAL COLUMNS

For each of the tables that include additional columns (see section 3B.3), complete the corresponding columns in the summary row using the appropriate method as described in section 3C.2.

3C.3 COMPARE RESULTS

Using a combination of the summary rows and the data held within the tables, compare the provision of the UIDTs examined.

4 UIDT SELECTION FOR AN UNPRECEDENTED PROJECT

Section 1.2 discussed how SUIT can be used to select a UIDT for a specific unprecedented project. It provides a reference point for the conditions under which this kind of selection process is possible and the significance of the results obtained.

When performing this kind of UIDT comparison, the SUIT framework is tailored to the needs of the given project. In essence this generates a project-specific reference model against which the various UIDTs are assessed. The following methodology should be observed when performing a UIDT selection for an unprecedented project.

Notes:

- the term **select** refers to putting a ☑ in the appropriate box;
- terms which are used in the tables are written in *italics*;
- for clarification of the meaning of any term, refer to the corresponding glossary page;
- the appropriate page number is given in brackets – e.g. [1].

4A Tailor The SUIT Framework

With reference to the given project, complete the following steps to tailor the SUIT framework for use during data-collection. It should be remembered that, as mentioned in section 2.2, it is possible to complete this tailoring in an iterative process as more detailed information becomes available during the lifetime of the project.

4A.1 ENVIRONMENTAL CONTEXT OF USE

These aspects of comparison ensure that the needs of the project team members and the higher level organisational goals are given due consideration during the selection of a UIDT – see section 3.1 for greater discussion of their importance. Note that this section of the framework preparation actually involves data-collection since the information obtained is used later in the tailoring process.

4A.1.1 HUMAN RESOURCES [I]

If the project team has been decided, for each member who will be required to utilise the selected UIDT, gather and then fill in the following information. If there are more than five members in the team, simply duplicate page I as many times as required. If the team has yet to be assigned, the completion of this section should be postponed until such time as the information is available.

4A.1.1.1 Rapid Prototyping

The following steps refer to the member's experience in terms of rapid prototyping. If the member has no such experience, no selections should be made.

☑ *Interaction Mechanisms*

☑ Select all *interaction mechanisms* with which the member has experience. If the member has experience of rapid prototyping using a programming language, select the first of the two boxes under this heading and use the second to specify which language(s).

[+/-] *Routine Work*

[+/-] If rapid prototyping forms part of the routine work of the member, put a "+" in the first of the two boxes; otherwise, put a "-" in the first box. In either case, provide a short explanation in the second of the two boxes.

4A.1.1.2 End-Product Implementation

The following steps refer to the member's experience in terms of implementing the end product. Again, if the member has no such experience, there should be no selections made.

☑ *Interaction Mechanisms*

☑ Select all *interaction mechanisms* with which the member has experience. If the member has experience of implementation using a programming language, select the first of the two boxes under this heading and use the second to specify which language(s).

[+/-] *Routine Work*

[+/-] If end product implementation forms part of the routine work of the member, put a "+" in the first of the two boxes; otherwise, put a "-" in the first box. In either case, provide a short explanation in the second of the two boxes.

4A.1.1.3 Generic User Information

The last three points concerning human resources are generic in that they do not attempt to capture information specific to rapid prototyping or implementation. Instead, they are concerned with ascertaining the levels of understanding and stress under which the project member operates since these are factors which could influence the manner in which the member adopts the selected UIDT.

[+/-] *Semantic Knowledge of the Domain*

[+/-] If the member has in-depth understanding or knowledge of the domain for which the application is begin developed (e.g. a good understanding of the terminology which is to be used), put a "+" in the first of the two boxes; otherwise, put a "-" in the first box. In either case, quantify the allocation in the second of the two boxes.

[+/-] *Stress Levels*

[+/-] If the member is under very little stress at work, put a "+" in the first of the two boxes; otherwise, put a "-" in the first box. In either case, provide a short explanation in the second of the two boxes. Note the inversion of the use of the symbols in this instance – low levels of stress are taken to be positive aspects of the human resource and so are flagged with a "+" whereas high stress levels are a negative aspect as indicated by the "-".

[+/-] *Work Load*

[+/-] If the member has a very low work load at the time of the project (e.g. the given project is the only project in which he is a member), put a "+" in the first of the two boxes; otherwise, put a "-" in the first box. In either case, provide a short explanation in the second of the two boxes. Again note the inversion of the use of the symbols in this instance – low work loads are viewed as positive aspects of the human resource and so are flagged with a "+" whereas high work loads are a negative aspect as indicated by the "-".

4A.1.2 INSTITUTIONAL GOALS/CONSTRAINTS [II]

Where the institution has specific goals or constraints at either the level of the organisation or at the level of the project, complete the following steps.

[+/-] *Programming Language*

[+/-] If the organisation is restricting the programming language underlying the UIDT to a group of languages or even a single programming language, place a "-" in the first box and provide an explanation which includes the possible choice(s). Note that a restriction may result from institutional constraints (e.g. financial) or may be the result of an institutional goal (e.g. to increase or specialise the language knowledge base with the organisation). Conversely, if the organisation is placing no restriction on the choice of programming language, place a "+" in the first box and a comment in the second.

[+/-] *Staff Training/Experience*

[+/-] If the organisation is restricted by the experience of its staff and cannot afford to invest in staff training, place a "-" in the first box. If the organisation wishes to use the introduction of the UIDT as an incentive for investment in staff training or to increase

the staff experience, place a "+" in the first box. In either case, provide an explanatory comment.

[+/-] *Tool Use*

[+/-] If the organisation is restricting the UIDT selection set to an existing internal library of tools or to a subset of the potential tools (e.g. based on programming language, cost, experience etc), indicate this with a "-" in the first box. Conversely, if the universe of potential UIDTs is open, place a "+" in the first box. Include an explanation in either case, itemising restrictions in the former.

[+/-] *Cost*

[+/-] If the organisation has to restrict the selection of UIDTs on the basis of cost, place a "-" in the first box and an explanation with actual cost restrictions in the second. If the cost of the UIDT is not a restrictive factor in the selection of a tool, indicate this with a "+" and comment this decision.

[+/-] *Development Platform*

[+/-] If the organisation is limited to specific platforms for development, the UIDT will have to be operational on one of these platforms and so selection of a UIDT will be restricted to those which can conform. This should be marked with a "-" and the accompanying explanation should itemise the platform restrictions. On the other hand, if the organisation has the ability to set-up any required platform – be it through purchase of new equipment or use of existing equipment and is under no contractual obligation to a specific development platform, then mark this with a "+" and provide an explanatory comment.

[+/-] *Development Time Limits*

[+/-] If the organisation requires a specific development turn-around time, then selected UIDT must make this possible my providing rapid development, and so the set of potential UIDTs may therefore be restricted. Mark this with a "-" and explain why. Inversely, there may be no developmental time limits that would be of significance to the choice of UIDT and as such this can be flagged with a "+" and commented.

[+/-] *Other*

[+/-] If there are any other restrictions on the choice of UIDT, use this catch-all row to itemise, quantify, and explain them.

4A.2 USER INTERFACE-SPECIFIC REQUIREMENTS

The level of detail at which the user interface features of the framework can be completed is dictated by the stage of design or development of the project. Each of the following steps should be performed with the project-specific requirements in mind. Where the requirements of the project are yet to be decided, the categories can be temporarily completed at an abstract level and then refined as detail becomes available – as described in each section.

For each of the listed categories relating to general properties of the user interface, adopt one of the following two approaches.

Approach 1: if no detailed information is available about the need for any of the components in the category, simply select the category – in other words select the first row in the table which will bear the title of the category. For example ☑ Screen Navigation on page 1.

Approach 2: if it is known which components (or likely components) will be needed in the user interface, select only those components from each category and leave the redundant components unselected. Note that if any category-specific component is not listed by the framework, use the *other* row to make it an active part of the selection process.

Note that if it is known that a category of components will definitely not be needed, the entire category can be ignored – that is, left unselected and therefore omitted from the data-collection process.

4A.2.1 GENERAL PROPERTIES OF THE USER INTERFACE

☑ Select *Screen Navigation* [1]
or ☑ Select one or more of: *Scrolling*; *Paging*; *Key-Combinations*; and *other*.
☑ Select *Dialogue* [2]
or ☑ Select one or more of: *End-User Customisable*; *Adaptive*; *Multi-Thread; Asynchronous*; *End-user Interruptible*; and *other*.
☑ Select *Active Devices* [3]
or ☑ Select one or more of: *Multiple Active Devices*; *Single Active Devices*; and *other*.
☑ Select *Miscellaneous* [4]
or ☑ Select one or more of: *Highlighting*; *Target System Help*; and *other*.

4A.2.2 INPUT/OUTPUT DEVICES

☑ Select *Input Devices* [5-6]
or ☑ Select one or more of: *Keyboard*; *Mouse*; *Function/Command Keys*; *arrow Keys*; *Voice Input*; *Touch Screen*; *Graphics Tablet*; *Joystick*; *Trackball*; *Footpedal*; *Lightpen*; *Dial/Valuator*; *Button Box*; *Finger Mouse*; *Head Pointer*; *Eye Tracker*; *Finger/Hand Pointer*; *Data Glove*; *Fingerprint Scanner*; *Iris Scanner*; and *other*.
☑ Select *Output Devices* [7]
or ☑ Select one or more of: *Voice Output*; *Tonal Output*; *Print/Hard Copy*; and *other*.
☑ Select *Target Monitor* [8]
or ☑ Select one or more of: *Standard Desktop Monitor (colour)*; *Standard Desktop Monitor (monochrome)*; *Bit Mapped/High Resolution Graphics Monitor*; *Alpha-Numeric Text Only Monitor*; *Palmtop*; *Laptop*; and *other*.
☑ Select *Other Hardware Devices* [9]
or ☑ Select one or more of: *Videodisk Player*; *Videotape Player*; *CD ROM Drive*; *Audio CD Player*; *Microphone*; *External Speakers*; *Midi Controller*; *Scanner*; and *other*.

4A.2.3 OUTPUT PRESENTATION TYPES

☑ Select *Text* [10]
☑ Select *Graphics* [11-12]
or ☑ Select one or more of: *Bit-Mapped*; *Vector*; *Imported Images*; *Geographical*; *Pie Charts*; *X-Y Charts*; *3D Images*; and *other*.
☑ Select *Animation* [13]
☑ Select *Audio* [14-16]
or ☑ Select *Audio Midi*
and/or ☑ Select *Audio Synthesised Speech*
or ☑ Select one or more of: *From Text Strings*; *From File*; *Male Voice*; *Female Voice*; *Natural Tone*; *Robotic Tone*; *Pitch Control*; *Rate Control*; and *other*.
and/or ☑ Select *Audio Non-Speech Sounds*
and/or ☑ Select *Audio Auditory Icons*
and/or ☑ Select *Audio Other*
☑ Select *Video* [17-19]
or ☑ Select one or more of: *Videodisk*; *Videotape*; *Frame Grabber*; *Gen Lock*; *Imported*; *Freeze Frame*; *Special Effects*; *Frame Editor*; *Multiple Concurrent Videos*; *Control*; and *other*.
☑ Select *Data Driven/Dynamic Objects* [20-21]
or ☑ Select one or more of: *Discrete Objects*; *Bar Graphs*; *Line Graphs*; *3D Graphs*; *Pie Charts*; *Histograms*; *Rotators*; *Dials*; *Scales*; *Text*; and *other*.

4A.2.4 COMBINATIONS OF OUTPUT PRESENTATION TYPES

☑ Select *One Screen/Different Windows* [22]
☑ Select *One Screen/Same Window* [22]

4A.2.5 USER INTERFACE FEATURES

☑ Select *Boxes* [23]
or ☑ Select one or more of: *List Box*; *Data Entry Box – Single Line*; *Data Entry Box – Multiple Line*; *Message Box*; *Dialogue Box*; *Default Boxes*; and *other*.

☑ Select *Menus* [24-25]

or ☑ Select one or more of: *Binary*; *Extended*; *Permanent*; *Pull-Down*; *Pop-Up*; *Tear-Off*; *Embedded*; *Radio Buttons*; *Push Buttons*; *Check Buttons*; *Palette*; *Pie*; *Default*; and *other*.

☑ For each of the selected menus, select one or more of the following from the first six columns: *Static Items*; *Dynamic Items*; *Linear Sequence*; *Hierarchical Sequence*; *Textual Items*; *Graphical Items*; and *Combination Text/Graphic*.

Note that these pages can be duplicated if it is necessary to treat each combination of menu type and menu content/organisation independently. For example Binary menus with static items in a linear sequence could be selected on one duplication and on another, Binary menus with dynamic items in a hierarchical sequence could be selected such that they are dealt with independently. This is entirely at the discretion of the evaluator.

☑ Select *Forms* [26-28]

or ☑ Select one or more of: *Non-Enumerated Input*; *Enumerated Input*; *Formatted Fields*; *Optional Fields*; *Required Fields*; *Data Dependent Fields*; *Default Value Fields*; *Single Page Form*; *Multiple Page Form*; and *other*.

and ☑ Select one or more of: *Mouse Pick*; *Arrow Keys – Bi-directional*; *Arrow Keys – Wrap Around*; *Tab Keys – Bi-directional*; *Tab Keys – Wrap Around*; and *other*.

☑ Select *Windows* [29-32]

or ☑ Select one or more of: *Overlapping Windows*; *Tiled Windows*; *Primary Windows*; *Secondary Windows*; *Default Standards*; and *other*.

and ☑ Select *Window Transition*

or ☑ Select one or more of: *None*; *Blinds*; *Collapse To Centre*; *Diagonal Contract/Expand*; *Dissolve*; *Expanding Quadrants*; *Fade from Colour*; *Horizontal Fold/Unfold*; *Slats*; *Wipe*; ; and *other*.

☑ Select *Text Area* [33]

or ☑ Select one or more of: *Simple Typed Input*; *Input With Required Arguments*; *Input With Optional and/or Required Arguments*; *Input Language With Arbitrary Grammatical Structure*; and *other*.

4A.3 DEVELOPMENTAL CONTEXT OF USE

The following steps should be performed with consideration of the working practice into which the UIDT is expected to fit and required standards of the product of the tool. This includes the support systems used, the developmental processes adopted by the organisation (in particular for the given project) and specific properties of the UIDT itself.

For each of the listed categories relating to the developmental context of use, adopt one of the following two approaches.

Approach 1: if no detailed information is available about the need for the components in the category, simply select the category – in other words select the first row in the table which will bear the title of the category. For example ☑ Evaluation Assistance on page 35.

Approach 2: if it is known what developmental components (or likely components) will be needed, select those components from each category and leave the redundant components unselected. Note that if a category-specific component is not included by the framework, use the *other* row to make it an active part of the selection process.

As with the user interface-specific requirements, if it is known that a category of components will definitely not be needed, the entire category can be ignored – that is, left unselected and therefore omitted from the data-collection process.

☑ Select *UI Design & Development Steps* [34]

or ☑ Select one or more of: *Systems Analysis*; *User Interface Software Design*; *User Interface Interaction Design*; *User Interface Software Implementation*; *Formative User Based Evaluation*; *User Based Evaluation*; *User Based Testing of the Interface Software*; and *other*.

and/or ☑ Select *Rapid Prototyping*

or ☑ Select one or more of: *Horizontal Prototypes*; *Vertical Prototypes*; *Global Prototypes*; *Local Prototypes*; *Paper Based Prototypes*; and *other*.
☑ For each of the selected prototypes, select one or both of the following from the appropriate additional columns: *Evolutionary*; and *Revolutionary*.

☑ Select *Evaluation Assistance* [35]
or ☑ Select one or more of: *Event Level Data Collection*; *Selective Data Collection*; *Data Analysis*; *Data Management*; *Session Replay*; and *other*.
☑ Select *Developmental Aids* [36]
or ☑ Select one or more of: *Code Editor*; *Interpreter*; *Compiler*; *Step Through Execution*; *Debugger*; *Property List*; *Hierarchy Editor*; *Class Browser*; *File Comparison*; and *other*.
☑ Select *Project Management* [37]
or ☑ Select one or more of: *Configuration/Version Control*; *Documentation*; *Design Representation*; *Data Dictionary*; *Approval/Signing-Off Process*; *Quality Control/Measurement*; and *other*.
☑ Select *Tool Characteristics* [38]
or ☑ Select one or more of: *Reuse of Tool Output*; *Extension of Tool Features*; *Modification of Tool Features*; *Observance of Standards*; *Portability of Tool Output*; *Integration With DBMS*; *Integration With Other Software*; and *other*.
☑ Select *Quality Attributes Of Target System* [39]
or ☑ Select one or more of: *Modifiability of Target System Code*; *Compatibility of Target System Code*; *Reusability of Target System Code*; *Response Time Of Target System Code*; and *other*.

4B Generate Ideal UIDT Profile

Once the information about the human resources for the project has been collected and the required UIDT components have been selected, these two aspects of UIDT use can be integrated in order to generate a profile of the ideal UIDT for the given project. Essentially, the ideal profile is a tailored framework for which the interaction mechanisms have been quantified in consideration of the project's human resources and interaction mechanisms best suited to the tasks to be performed. It is against this profile that the data gathered using the tailored framework created in 4A will be compared.

The ideal profile is generated following a two-step process as outlined below. The parts labelled (a) and (b) in each case discuss the option to increase the complexity and accuracy of the ideal profile.

4B.1 Analyse The Human Resource Information

Refer to the human resource information collected in steps 4A.1.1.1 and 4A.1.1.2 – Rapid Prototyping and End-Product Implementation respectively.

a) For each of the interaction mechanisms, count the number of team members who have experience using that mechanism for either rapid prototyping or end product implementation. If there is one interaction mechanism which has clearly been used more than any other, it should be considered the generic interaction mechanism most appropriate for the whole project team across all activities. If there are several interaction mechanisms all of which have similar levels of use, they should all be considered as the most appropriate mechanisms.

b) Although not essential (see section 4B.2), it may also be potentially beneficial to separately analyse and calculate the generic mechanism or appropriate mechanisms for Rapid Prototyping and for End Product Implementation independently.

4B.2 Integrate The Information

The actions listed below should be taken for each of the tables [pages 1 – 39] in which there are columns corresponding to interaction mechanisms. Reminder: select action (a) or (b) based on the required level of accuracy.

a) For each of the rows corresponding to selected components (in other words, rows for which the component has been specified as required for the given project - or every row in the case of a generic category selection), select the box or boxes reflecting the generic interaction mechanism or appropriate interaction mechanisms as calculated in section 4B.1(a). For example, if analysis of the human resources revealed that graphical manipulation and form-filling were the most appropriate interaction mechanisms with respect to the project team members, and it had been identified that list boxes were required in the user interface being built, the three rows for instantiation, configuration and layout of list boxes should have the boxes in the columns corresponding to graphical manipulation and form-filling selected in each case.

b) For each selected component in the tailored framework, if it is known that the component will only be specifically used in *either* rapid prototyping *or* implementation, the appropriate interaction mechanism(s) as analysed in 4B.1(b) can be used instead of the more generic human resource analysis. For example, if it is known that Data Analysis [page 35] will only be used during rapid prototyping, and that the most common (i.e. generic) interaction mechanism for rapid prototyping (only) is graphical manipulation, then the boxes corresponding to graphical manipulation for instantiation and configuration of data analysis should be selected as the ideal interaction mechanism in that instance. If this knowledge is not known for any component, then approach (a) can be adopted for that component.

Finally, if it is possible to identify which team members will perform specific (or isolated) tasks in the project development, the ideal interaction mechanism(s) for that specific task or tasks should be specified as the interaction mechanism with which the individual team member is most familiar. For example, if it is known that Team Member 1 (and only Team Member 1) will perform all implementation and is accustomed to graphical manipulation for user interface implementation, then the ideal interaction mechanism for all implementation-specific tasks can be listed as corresponding to his experience - i.e. as graphical manipulation.

c) If it is possible to identify interaction mechanism(s) that are best suited to any specific task related to any specific required functional component, these interaction mechanism(s) should either be included with the ideal interaction mechanisms listed during steps (a) and (b) for the corresponding component or should replace those listed in steps (a) and (b).

Once the above tasks have been completed, the project-specific ideal UIDT profile exists and is ready for use in comparing the results of the data gathered using the tailored framework - see section 4C.

4C Gather Data

The following steps should be completed *for each* of the UIDTs being considered in the comparison. Remember to consult page II (institutional goals/constraints) to acknowledge any restrictions on the universe of potential UIDTs for inclusion in the comparison.

4C.1 Produce Copies Of The Tailored Framework

Generate a copy of the tailored framework (pages 0 and 1 - 39). Complete all sections on page 0 to record high-level information about the evaluation, project and tool.

4C.2 Record Information

The information required to complete the following steps can be obtained either directly from use of the UIDT or, if one exists, from a completed tool template (or framework) for that UIDT. For each selected component in the tailored framework, record the required information by selecting the box in the appropriate column(s):

4C.2.1 Interaction Mechianism(s) Used To Include A Component

Where applicable, the activities involved in the inclusion (or use) of a component are broken down into the instantiation of the component, the specification of its layout, and configuration of its

properties. In each case, select the interaction mechanism(s) which are provided by the UIDT in order to complete the inclusion (or use) of the component.

4C.2.2 INTERACTION ASSISTANCE PROVIDED

Select the style(s) of interaction assistance which the UIDT provides to guide or inform the performance of each of the steps required for the inclusion (or use) of the selected component.

4C.2.3 COGNITIVE DEMANDS

This refers to the amount of mental effort demanded of the UIDT user whilst performing the tasks discussed above. The aim of the UIDT should be to minimise the cognitive demands. Therefore, if the level of mental effort required of the UIDT user is high, place a "-" in the first box under this heading, and record an appropriate explanatory comment. Conversely, if performance of the task requires little mental effort, place a "+" in the first box and record an explanation. Note the inverse correspondence between the symbol allocation and the level of cognitive demands. Low cognitive demands are viewed as an advantage to interaction with the UIDT and are therefore signalled with a "+" whereas high cognitive demands are considered a disadvantage and are marked with a "-".

4C.2.4 QUALITY OF FEEDBACK

If the UIDT provides good feedback to the user when performing each of the tasks discussed previously, place a "+" in the first of the two boxes under this heading and record an explanatory comment. If feedback is poor mark the first box with a "-" and explain why.

4C.2.5 MODIFIER/OTHER COMMENTS

This section allows miscellaneous comments to be recorded regarding the actual component or the performance of the associated task – in other words it is a catch all section for recording peculiarities (good, bad, or unusual aspects) of the UIDT with regards the selected component. If the UIDT does something particularly well, signal this with a "+" and indicate the nature of the advantage with a descriptive comment. In contrast, if the UIDT makes something particularly difficult to achieve or the associated component is inadequate, mark a "-" and provide an explanation.

4C.3 ADDITIONAL COLUMNS

The nature of some of the components demanded that additional columns be included in the associated tables. For each table, explanation regarding the meaning of these extra columns is provided on the affiliated glossary page. For each selected component (or row) in these tables, if the UIDT supports the features represented by these additional columns, the appropriate box(es) should be selected. For example, consider the table for Dialogue on page 2. Two additional columns have been introduced to record whether the specific styles of dialogue (e.g. adaptive) can have a linear or a hierarchical organisation. For each selected style of dialogue, if one or both of these options is available and has been specified as required, the appropriate box or boxes should be selected.

4D Compare Results

Once the data has been gathered for each of the UIDTs, they can be compared with respect to the ideal profile generated in section 4B. The steps detailed in sections 4D.1 and 4D.2 outline the means by which the data collected can be summarised and compacted. These steps are not compulsory since the summarised data will not be used explicitly in the comparison process. The summary information is, however, useful for obtaining sanitised information quickly or can be useful for performing a 'speed' comparison where a more detailed comparison will be performed at a later date but initial data is required fast.

4D.1 HIGHLIGHT UNSUPPORTED FUNCTIONALITY

Strike-through every selected row in each table for which the UIDT makes no provision. For example, if the tailored framework includes *Tear-Off Menus* [page 24] but the UIDT does not provide them, put a line through each of the associated rows to easily identify that the UIDT does not provide the required component (see Figure 8).

Figure 8 – *Highlight unsupported functionality*

4D.2 COMPLETE THE SUMMARY ROW

Each table is preceded by a summary row. It reflects the columns in the associated table and enables the information obtained for that category to be viewed at a glance, albeit in less detail than would be obtained by examining each selected row in the table in turn. The summary row is completed as follows:

4D.2.1 INTERACTION MECHANISM

For each of the columns reflecting interaction mechanisms, count the number of selections which have been made – in other words the number of rows in which the column has been selected. The interaction mechanism with the greatest count is the predominant interaction mechanism for that specific category of the UIDT. The corresponding column should be selected in the summary row. If the counts for two or more interaction mechanisms are equal or very close, each of the corresponding columns in the summary row can be selected to reflect, at a glance, the spectrum of interaction mechanisms used by the UIDT. Figure 9 shows this being done for *Screen Navigation.* Notice that three types of screen navigation have been included in the tailored framework but the UIDT does not support *Paging* and so the absence of this functionality has been highlighted.

4D.2.2 INTERACTION ASSISTANCE

For each of the columns reflecting interaction assistance, count the number of selections which have been made. Again, the column(s) with the highest count(s) indicate the most employed interaction assistance mechanism(s) for components in the associated table. Select the corresponding column(s) in the summary row.

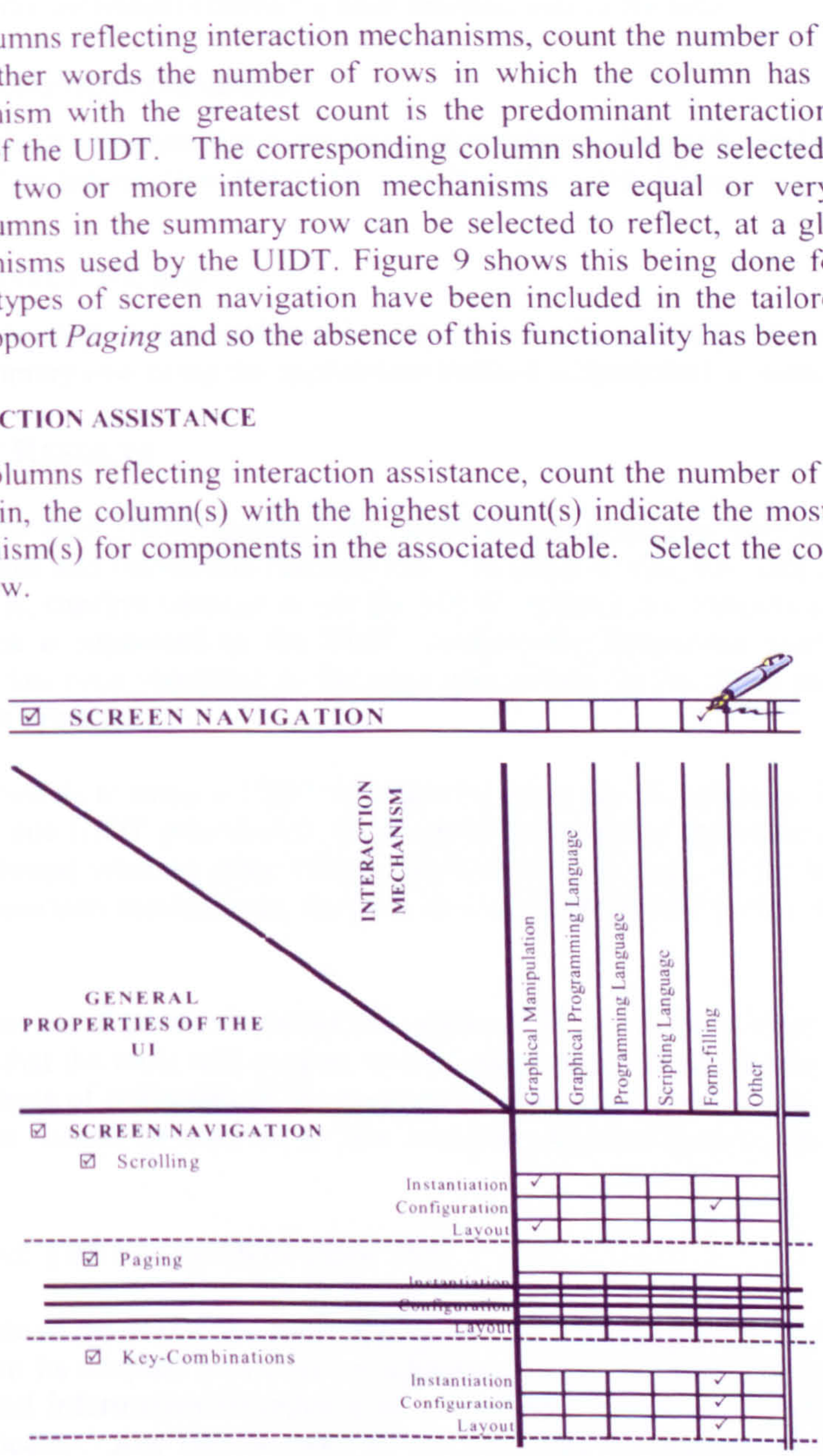


Figure 9 – *Completing the summary row*

4D.2.3 COGNITIVE DEMANDS

To summarise the cognitive demands is less straight forward than the summation process described in the previous two sections. Given that a "-" indicates a negative point concerning a component and a "+" indicates something positive, it is possible to count the number of negative and the number of positive points and reflect the greater of the two with the appropriate symbol and summative comment

in the corresponding columns of the summery row (if the counts are equal, a neutral "0" can be recorded).

However, given that the severity or significance of each positive or negative aspect will not have the same weighting, this kind of summation is of limited value. Instead, the discretion of the evaluator should be called upon to weigh-up the positive and negative aspects in light of their accompanying comments and with respect to the project and then insert the most appropriate or accurate summary in the summary row.

4D.2.4 QUALITY OF FEEDBACK

The summation of this aspect should be treated in the same way as the cognitive demands. That is, the most appropriate symbol (-,0,+) and accompanying comment should be left to the discretion of the evaluator based on the individual entries for each selected row in the table.

4D.2.5 MISCELLANEOUS COMMENTS

Although more difficult to summarise a collection of comments with potentially random focal points, the summation of this information should be treated in the same fashion as cognitive demands and quality of feedback.

4D.2.6 ADDITIONAL COLUMNS

For each of the tables that include additional columns (see section 3B.3), complete the corresponding columns in the summary row using the appropriate method as described in section 4D.2.

4D.3 COMPARE RESULTS

On a table by table basis for each UIDT, compare the pattern of box selections with those in the ideal profile - that is, for the components and interaction mechanisms. In other words, for each required component as selected in the ideal profile, confirm whether or not the UIDT supports the component. Similarly, for each required component which *is* supported by the UIDT, compare the interaction mechanism utilised by the UIDT against that which has been identified as the most appropriate for the given project's human resource (as indicated in the project profile).

At this stage it may be possible to make a UIDT selection based on the closeness of its match with the ideal profile. For example, if one UIDT provided all the required components and made predominant use of the ideal interaction mechanism(s) whereas other UIDTs failed to support many of the required components *or* utilised inappropriate interaction mechanisms, then the former UIDT would be the most obvious candidate for purchase.

However, it is unlikely that the disparities between the various UIDTs will be as clear cut as described above. Instead it is more likely that the tools will support similar percentages of the required components and will match in similar percentages of utilisation of the appropriate interaction mechanisms. This being the case, the following approaches should be adopted to fine tune the selection process and discover the 'best-fit' UIDT.

4D.4 FINE TUNE THE SELECTION PROCESS

If it has not been possible to determine the most appropriate tool by comparing the functional provision of each tool (with respect to its adopted interaction mechanisms) with the ideal tool profile – as described in step 4D.3 – the additional information recorded during the evaluation process needs to be drawn upon to inform the selection process. Any one or more of the following can be performed in order to make an appropriate tool selection. It is, however, recommended that each of the following be considered in collaboration in order that the most informed decision can be made.

4D.4.1 EXAMINE INTERACTION ASSISTANCE PROVIDED

For each tool examine the nature of the interaction assistance which is provided. Where possible, eliminate from consideration those tools which do not provide interaction assistance which is fitting for the identified team members. It is recommended that this decision is made in conjunction with examination of the cognitive demands placed upon the users and the quality of the feedback given to

the users in order that a balanced view of the tool can be obtained. Although it may regularly be the case that the better tool will provide the more appropriate interaction assistance, this is not necessarily always the case. A tool may provide little or no interaction assistance, or even interaction assistance that is not best matched to the team members, but may place such minimal cognitive demands on its users and support their interaction with sufficiently high quality feedback that interaction assistance is of lesser importance than in another tool where the cognitive demands are so high and the quality of feedback so low that the interaction assistance – and in particular its specific nature – is a vital component in the successful use of the tool. At a simpler level, where tools provide almost identical levels of interaction assistance, it may be necessary to examine these additional criteria in order to differentiate between them.

4D.4.2 EXAMINE COGNITIVE DEMANDS

For each tool, examine the cognitive demands which are placed upon the tool users. In light of the experience and requirements of the team members, where possible, eliminate those tools which place an inappropriate mental demand upon their users. As with interaction assistance, cognitive demands should be considered in conjunction with the interaction assistance and quality of feedback provided by the tool. It may be the case that one tool places high mental demands on its users but counters those demands with excellent (and appropriate) interaction assistance thus reducing its complexity in contrast to another tool which places slightly lower mental demands on its users but fails to support its users with appropriate interaction assistance. Only collaborative examination of this information will highlight the most suitable tools. Similarly, examination of additional criteria may be required to decide between tools which are closely equalled in terms of their cognitive demands.

4D.4.3 EXAMINE QUALITY OF FEEDBACK

For each tool, examine the quality of feedback provided for the users of the tool. Eliminate, where possible, those tools which provide poor levels of feedback to users. It is perhaps more obvious that a tool which falls short in terms of user feedback is likely to be a less appropriate tool than one which gives excellent feedback to its users. However, it is still important to examine the interaction assistance provided for each tool in conjunction with its feedback in order to perhaps differentiate tools which are equally matched in terms of their quality of feedback.

4D.4.4 EXAMINE MISCELLANEOUS COMMENTS

For each tool, take note of the miscellaneous comments recorded during evaluation of the tool. The positive or negative nature of the comments should inform and influence discrimination between the various tools. These comments can be considered in conjunction with any of the other criteria mentioned above.

4D.4.5 EXAMINE INSTITUTIONAL GOALS/CONSTRAINTS

For each tool, consider its suitability with respect to the recorded institutional goals and constraints. Although these factors will have been taken into account during the initial selection of tools for consideration, re-examination of the remaining tools in light of the institutional goals and constraints may determine if any can be eliminated from consideration based on their level of conformance with the criteria. For example, if all remaining tools are practically identical in all other respects, their conformance with cost constraints may provide the determining factor in the decision making process.

4D.4.6 EXAMINE ADDITIONAL HUMAN RESOURCE INFORMATION

For each tool, consider its suitability with respect to the additional information recorded for the project team members. Consider whether or not the tool will assist in lowering the stress levels felt by the team members. Similarly, consider whether the effect it will have upon the work load of the team members. Although these aspects of information are less concrete than other data considered during this selection process, they may be able to eliminate tools from consideration and should therefore be taken into account if a selection has not been possible at this stage.

Hence, an examination of all ancillary information can determine the most appropriate tool within the context of the project for which the tool is being selected.

5 UIDT Selection Based On Comparison With Previous Projects

Section 1.3 discussed how SUIT can be used to select a UIDT for a specific project based on similarities between the current project and previous projects for which SUIT was used to select a UIDT. Refer back to this section for required clarification of the conditions under which this type of selection process is suitable.

The following methodology should be observed when performing a UIDT selection based on the selection results obtained for previous projects.

5A Identify A Similar Project Profile

From the existing project profiles (in other words the ideal tool profiles for the previous projects) which have been constructed, identify which is the closest match to the profile which would be generated for the current project. This can be done in a three step process as described below. Any disparities between a previous project profile and the requirements of the current project should be recorded for ease of reference at the next stage in tool selection.

5A.1 Environmental Context of Use

These aspects of comparison ensure that the needs of the project team members and the higher level organisational goals are given due consideration during the selection of a UIDT – see section 3.1 for greater discussion of their importance. Note that this section of the profile comparisons may involve data-collection for the current project in order to perform an accurate comparison of the project requirements.

5A.1.1 HUMAN RESOURCES

For each member on the current project who will be required to utilise the selected UIDT, gather and then compare the following information with that recorded on the profiles for previous projects. If there are more than five members in the team, simply duplicate page I as many times as required.

Perform the following two steps for the members' experience in terms of rapid prototyping and end product implementation.

- *Interaction Mechanisms*

  Perform a general comparison of all *interaction mechanisms* with which the project team members have experience. Identify whether there is a similar collective pool of experience for the team members on previous projects compared with the team members on the current project.

- *Routine Work*

  Consider the extent to which the work of the team members on the previous projects was routine in comparison to that of the team members on the current project. For each of the team members on previous projects, if the work was routine, this will be indicated with a "+". Conversely, if the work was not routine, this will be marked with a "-".

5A.1.1.1 Generic User Information

The last three points concerning human resources are concerned with ascertaining the levels of understanding and stress under which the project members operate since these are factors which could influence the manner in which the team members adopt the selected UIDT.

- *Semantic Knowledge of the Domain*

  Compare the level of understanding or knowledge of the domain held by the team members on previous projects with that held by the team members on the current project. For each team member on the previous projects, a good understanding or knowledge of the domain will be marked with a "+"and conversely, lack of familiarity will be indicated with a "-".

- *Stress Levels*

  Contrast and compare the stress levels felt by the team members on previous projects with those expected for the team members on the current project. For each team member on the previous projects, low stress levels will have been recorded using a "+"; high stress levels will be indicated with a "-". Note the inversion of the use of the symbols in this instance – low levels of stress are taken to be positive aspects of the human resource and so are flagged with a "+" whereas high stress levels are a negative aspect as indicated by the "-".

- *Work Load*

  Compare the work load placed upon the team members on the previous projects with that on the team members of the current project. Once again, a low work load will have been indicated with a "+" and a high work load with a "-".

5A.1.2 INSTITUTIONAL GOALS/CONSTRAINTS

Where the institution has specific goals or constraints at either the level of the organisation or at the level of the project, compare the following comparisons.

- *Programming Language*

  Compare programming language restrictions imposed on the previous projects with those on the current project. Where programming languages have been restricted, this will be indicated with a "-" and conversely, where there have been no restrictions imposed, this will be marked with a "+".

- *Staff Training/Experience*

  Compare the restrictions pertaining to staff training or experience for the previous projects with those for the current project. If the previous project was subject to restrictions in this regard, this will be indicated with a "-"; lack of restrictions or the intention to increase staff training and experience as a result of choosing a tool will be indicated with a "+".

- *Tool Use*

  Compare the restrictions on the set of UIDTs from which the previous projects could make a selection to those being imposed on the current project. Where a previous project was under no such restrictions or was actively encouraged to increase the library of UIDTs held by the organisation, this will be indicated with a "+"; where selection was restricted to a small subset of tools, a "-" will have been used.

- *Cost*

  Compare the cost constraints placed on the previous projects with those on the current project. Where there was no such limitation placed on the previous projects, a "+" will have been used to record the fact; otherwise, a "-" will be used.

- *Development Platform*

  Compare the development platform restrictions imposed on the previous projects with those in place for the current project. Once again, limitations on the platform will be indicated with a "-" and freedom of choice will be marked with a "+".

- *Development Time Limits*

  Compare the time limitations under which the previous projects were working with those imposed upon the current project. Where time scheduling was not a major limiting factor for previous projects a "+" will have been used. Restricted time limits will be indicated with a "-".

- *Other*

  Compare any additional constraints placed on the previous projects with those placed on the current project.

5A.2 USER INTERFACE-SPECIFIC REQUIREMENTS

Consider the user interface-specific requirements recorded for each of the previous projects – as shown by their profiles. Compare these recorded requirements with the requirements of the current project. If a component has been included in the profile for the previous projects this will be indicated with a selected box (☑). Where a previous project has considered all components within a specific category, this will have been indicated with a selected box for the whole category (located next to the category label).

5A.3 DEVELOPMENTAL CONTEXT OF USE

Compare the working practice of the previous projects with that of the current project. Consider also the similarities between the standards to be met by the tools for previous projects and the current project. Finally compare the support systems used, the developmental processes adopted by the organisation (in particular for the given project) and specific properties of the UIDT itself as recorded for the previous projects with those stipulated for the current project. Once again, the inclusion of a component within a previous project profile will be indicated with a selected box.

5B Adjust Previous Profiles

If step 5A failed to identify a project profile which was a significantly close match to that which would be generated for the current profile one of two actions can be taken:

- if there are only minor discrepancies between an existing profile and the requirements of the current project, the closest matching existing profile can be 'tweaked' to cater for the differences;

- if the discrepancies between the existing profiles and the requirements of the current project are too great, the selection process for an unprecedented project should be followed – in other words, comparison of the current project with previous projects uncovered no project precedence and so the alternative route through the methodology should be adopted (as described in section 4).

The remainder of this section assumes that a suitable profile has been identified. If the profile is practically identical to that required for the current project, the remaining steps described in section 5 can be ignored and **the tool chosen for the previous project can be assumed suitable for the current project.** If the profile is close but requires some tweaking, this and the following steps should be completed in order to identify the most suitable tool for the current project.

5B.1 TWEAK THE CLOSEST PREVIOUS PROJECT PROFILE

Select the profile from a previous project which most closely matches the requirements for this project. Produce a copy of this profile in order to preserve the original and complete the following steps using the copy.

For each of the discrepancies recorded during the comparison process described in section 5A, adjust the profile accordingly. Adjustments can take one of two forms:

- introduction of a requirement or information where none already exists;
- change to an existing selection or recording of requirements or information.

If the previous profile has left blank a specific aspect of information or has not selected a particular component which is required by the current project, the appropriate steps from sections 4A and 4B should be followed in order to introduce the data to the profile for the current project. If information recorded for the previous project is inappropriate for the current project, the existing information should be deleted and again the appropriate steps from sections 4A and 4B should be followed in order to insert the correct information.

5C Gather Additional Information

Given a suitably tweaked profile, the required information should be gathered. Where the components of the current profile remain unaltered, the information gathered for the previous project can be used directly. If, however, the current project profile has altered the previous project profile, new information needs to be collated. This should be done according to the process outlined in section 4C. Similarly, if the current project wishes to consider the adoption of a tool which was not examined for the previous project, section 4C should be followed in its entirety.

5D Compare Results

Once the additional information has been gathered, the tools can be compared with respect to the ideal profile for the current project. Although this comparison process is essentially the same as that described in section 4D, it can benefit from the comparative process already conducted for the previous project (provided no additional tools are being considered). For all unaltered components in the current project profile, the comparisons hold for the previous project. For altered or additional components, the process outlined in section 4D should be followed in order to contrast the components which differ between the two profiles. Indeed for information pertaining to additional tools, the whole of step 4D should be completed in order to complete the comparison.

Where the discrepancies between the previous profiles and the current profile are sufficiently insignificant that they are *unlikely* but not certain to affect the outcome of the selection process, it may be most beneficial to conduct the comparison of the gathered information with respect to the selected tool. In other words, where a component of the profile has been altered or added and the appropriate data gathered, the question should be asked: *"Does this new information invalidate the previous tool selection?"*. Unless the answer to the question is *"yes"* it should be possible to retain the previous tool selection. For example, consider the situation where a specific user interface feature is added to the profile for the current project. If the tool selected for the previous project meets the additional functional requirement, the tool can still be considered most appropriate for the current project. However, if the tool fails to meet this new requirement its suitability as the best-fit tool for the current project becomes questionable and an alternative from the other selected tools may be required.

Hence, the information obtained and comparative process undertaken for previous projects can radically reduce the amount of effort required for the selection of a tool which meets the needs of a new project which has similar requirements.

The remainder of this appendix contains the SUIT Framework (and associated glossary pages). Due to formatting restrictions on this thesis, the SUIT Framework has had to be reduced in size. A full size copy can be obtained, upon request, from the author.

S.U.I.T.

Section A - Evaluation Details

This framework has been completed as part of a. ☐ Generic Comparison of UIDTs ☐ Project Specific Selection of UIDTs Date _____

Evaluation performed by _____ Evaluation Ref No _____

Section B - Project Details

Project Title _____ Project Ref No _____

Project/Team Leader _____

Project Description

Section C - UIDT Details

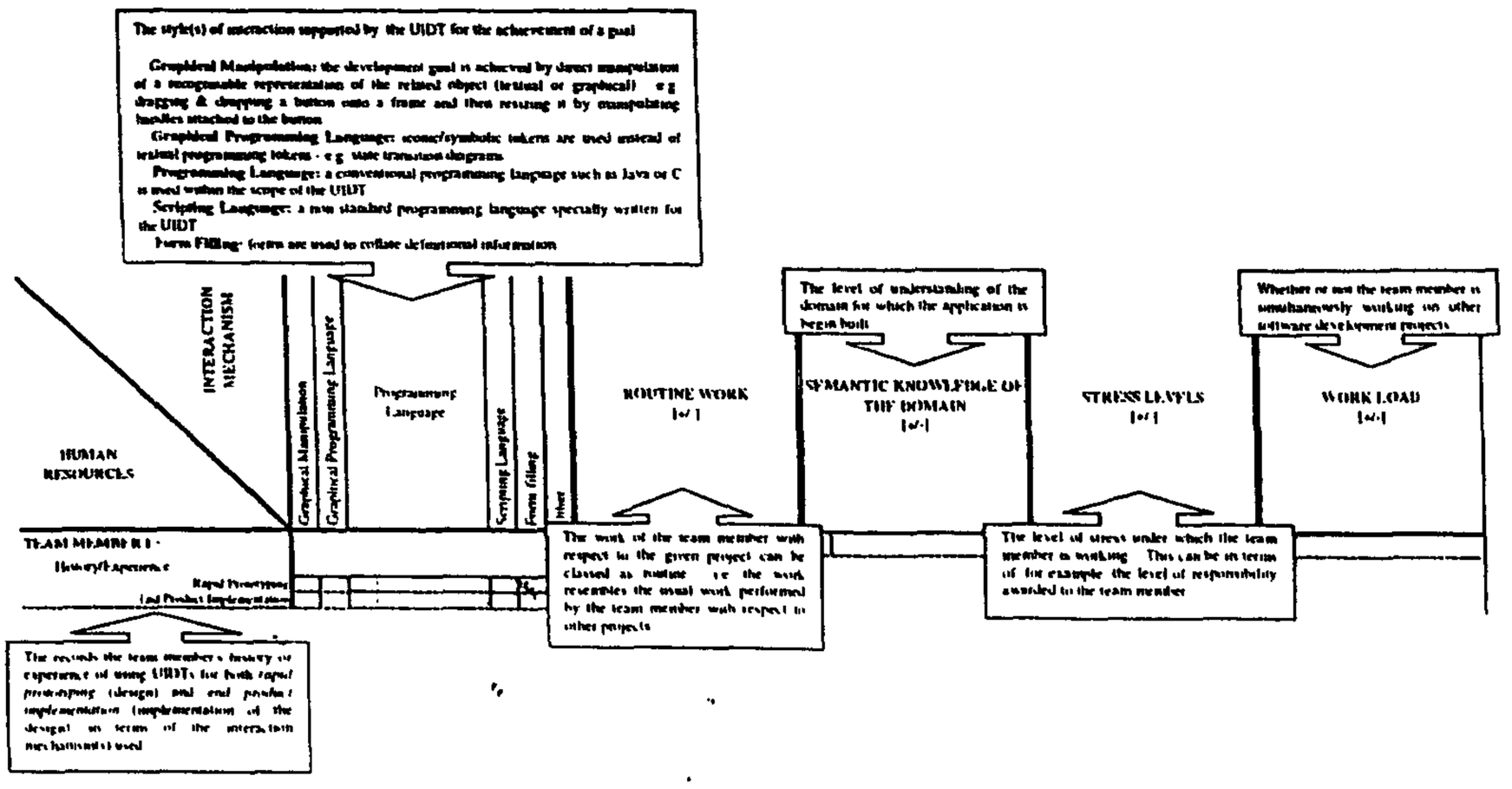
UIDT Title _____ Version _____

UIDT Description Cost _____

Programming Language(s) Used _____ Platforms _____

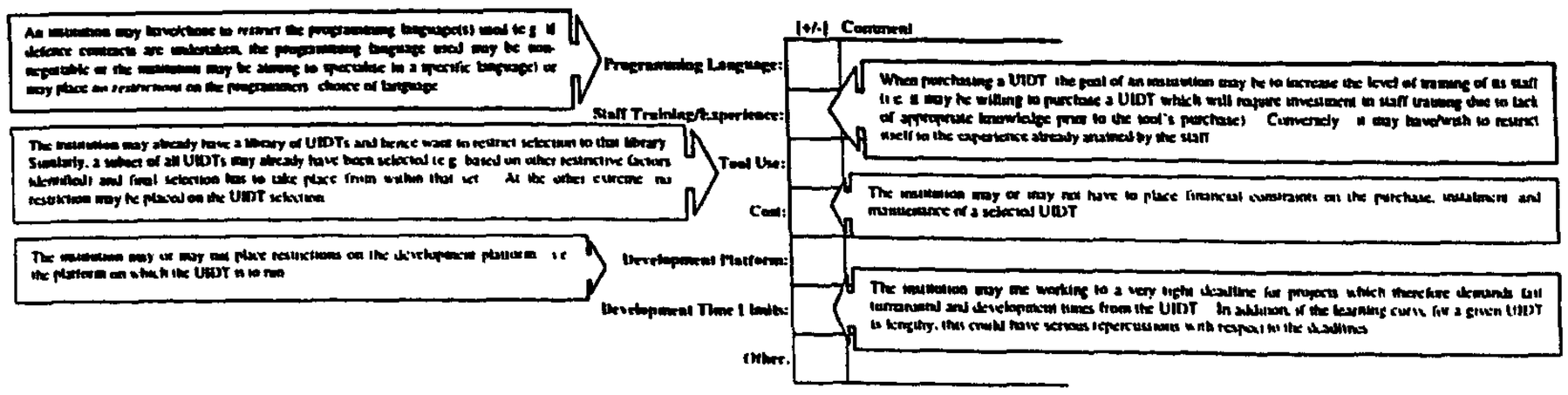
Technical Support Provision _____

HUMAN RESOURCES is concerned with information regarding the actual members of the development team for a given project. The information will be used to ensure that the UIDT selected is best suited to the particular attributes of the team members. For each member of the team, the following information is collated·



HUMAN RESOURCES \ INTERACTION MECHANISM	Graphical Manipulation	Graphical Programming Language	Programming Language	Scripting Language	Form-filling	Other	ROUTINE WORK [+/-]	SEMANTIC KNOWLEDGE OF THE DOMAIN [+/-]	STRESS LEVELS [+/-]	WORK LOAD [+/-]
TEAM MEMBER 1 :										
History/Experience										
Rapid Prototyping										
End Product Implementation										
TEAM MEMBER 2 :										
History/Experience										
Rapid Prototyping										
End Product Implementation										
TEAM MEMBER 3 :										
History/Experience										
Rapid Prototyping										
End Product Implementation										
TEAM MEMBER 4 :										
History/Experience										
Rapid Prototyping										
End Product Implementation										
TEAM MEMBER 5 :										
History/Experience										
Rapid Prototyping										
End Product Implementation										
Total Rapid Prototyping										
Total End Product Implementation										

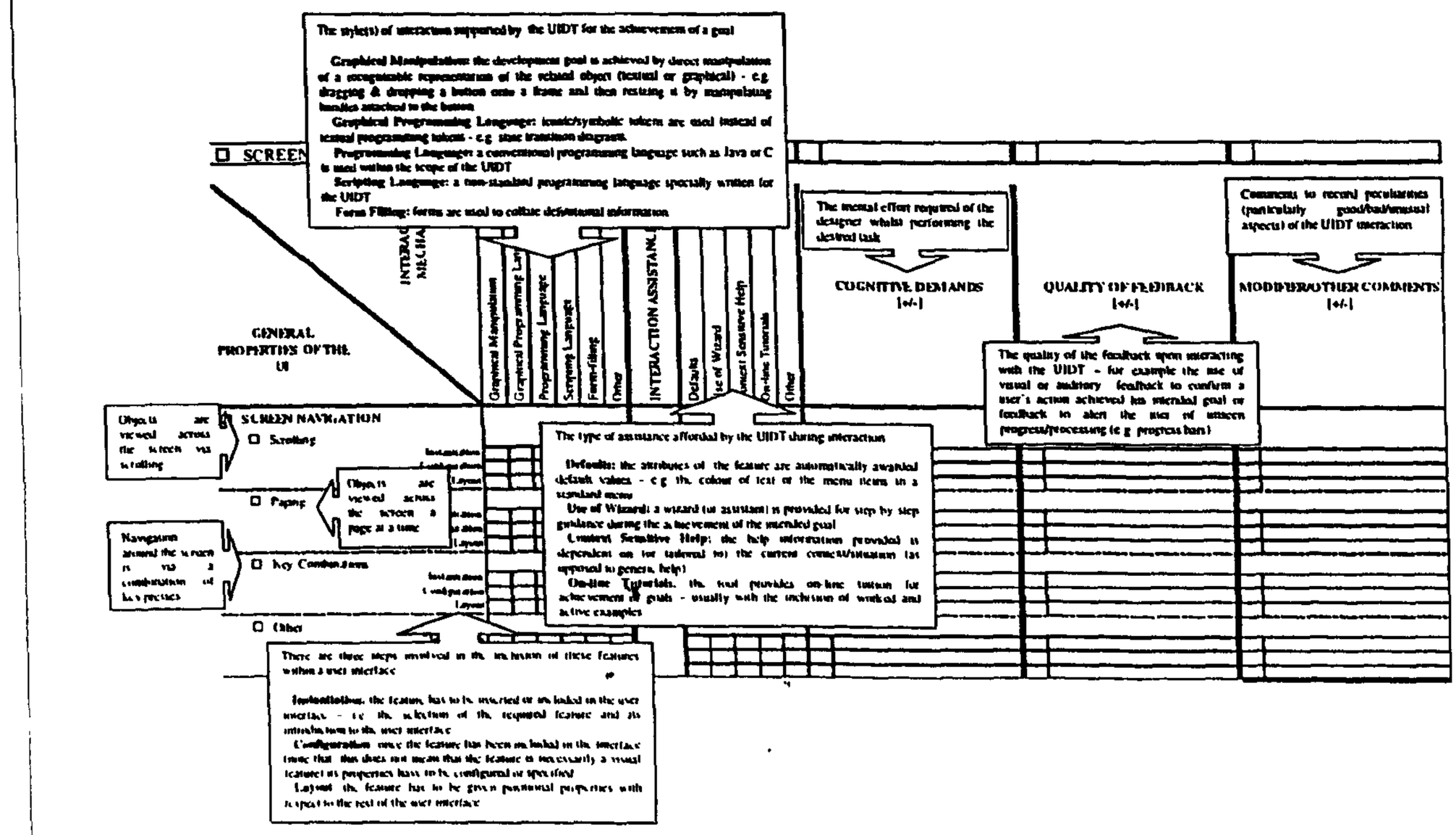
INSTITUTIONAL GOALS/CONSTRAINTS refers to the long or short term aims of the company with respect to investment in selected aspects of software development, and also to the constraints imposed upon the process of software development. The following categories attempt to capture such information.

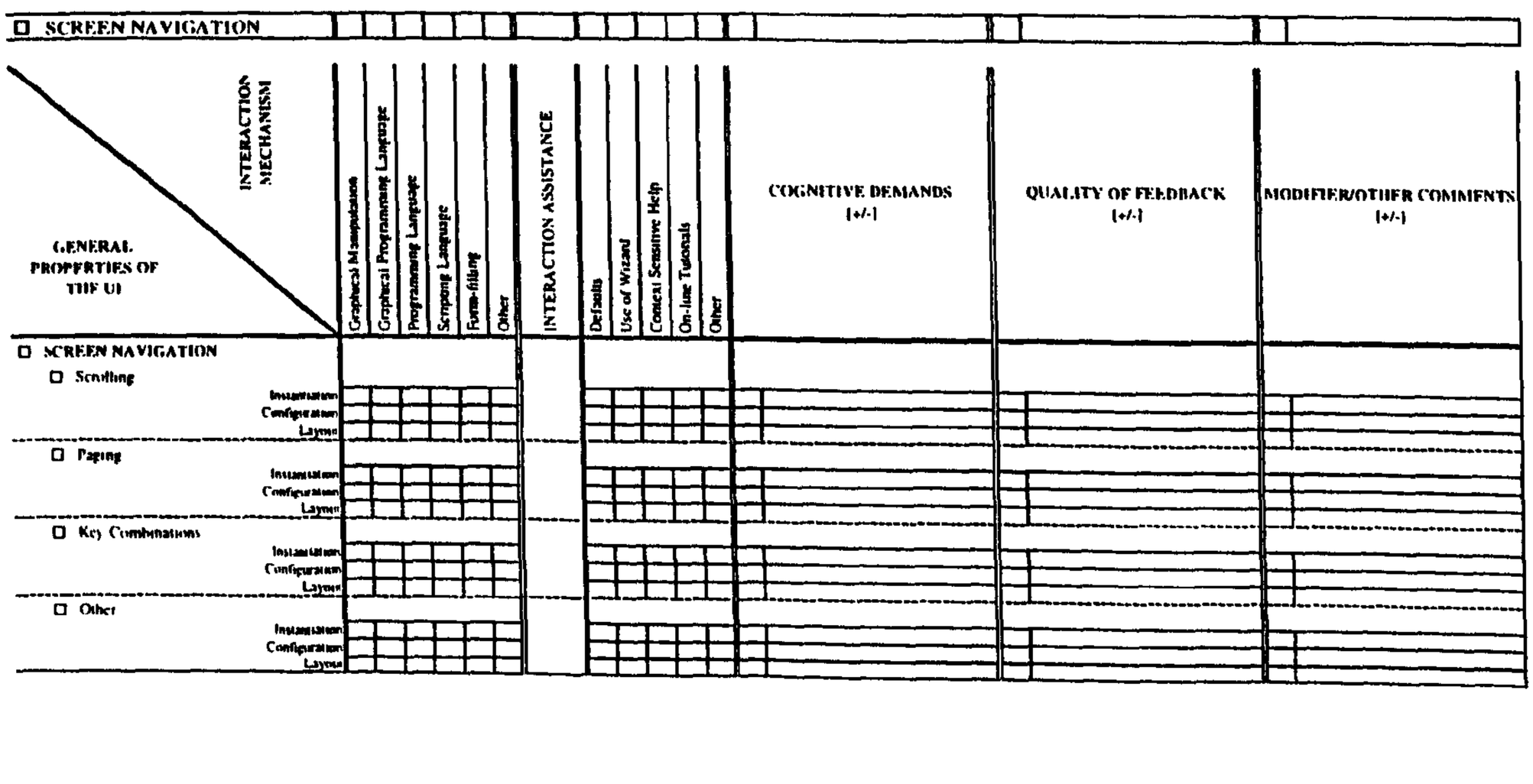


INSTITUTIONAL GOALS/CONSTRAINTS

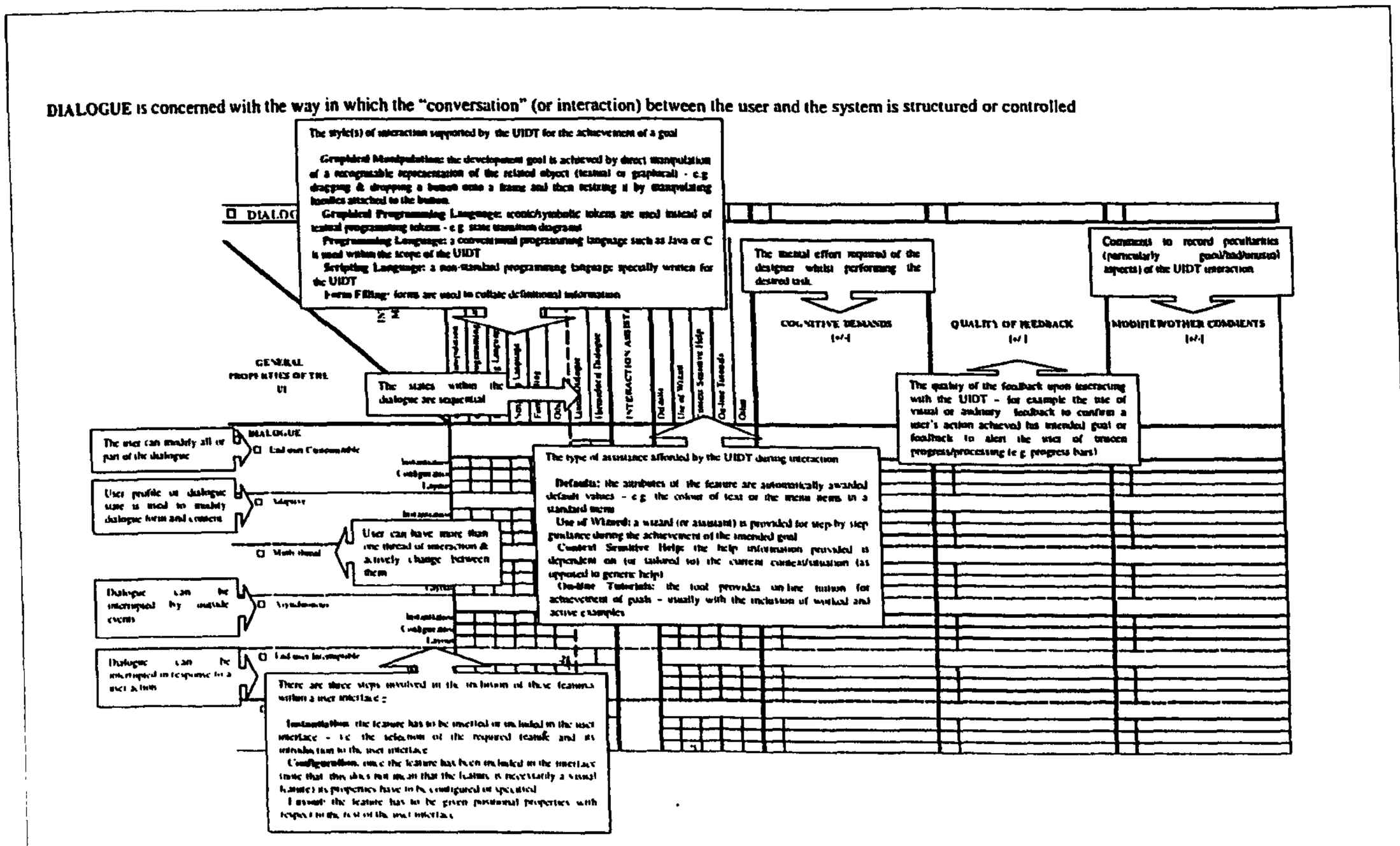
	(+/-)	Comment
Programming Language:		
Staff Training/Experience:		
Tool Use:		
Cost:		
Development Platform:		
Development Time Limits:		
Other:		

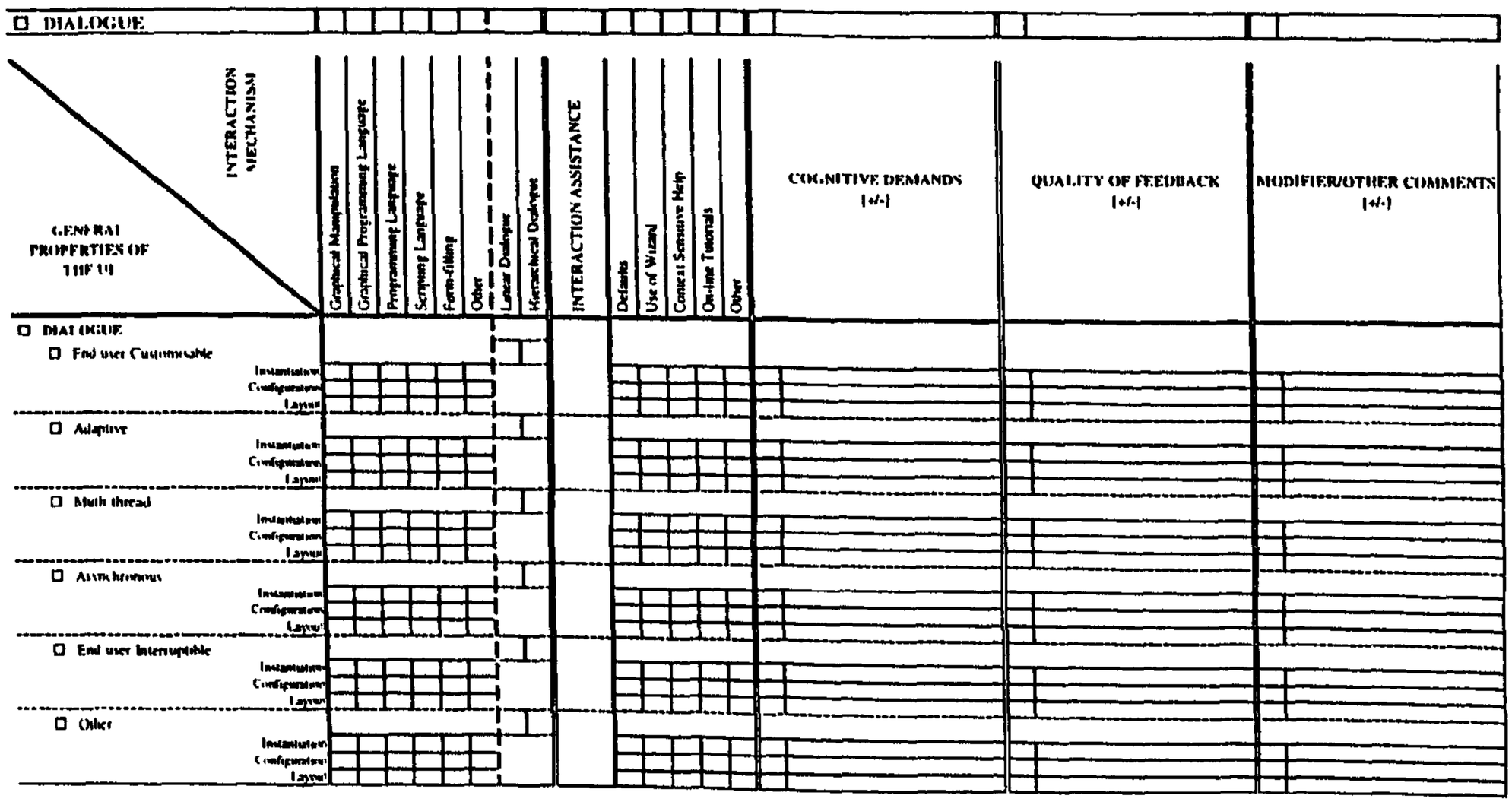
SCREEN NAVIGATION is concerned with the mechanisms provided for navigation around a single display in a user interface. Note that the means of navigation within a form are dealt with independently on page 27.



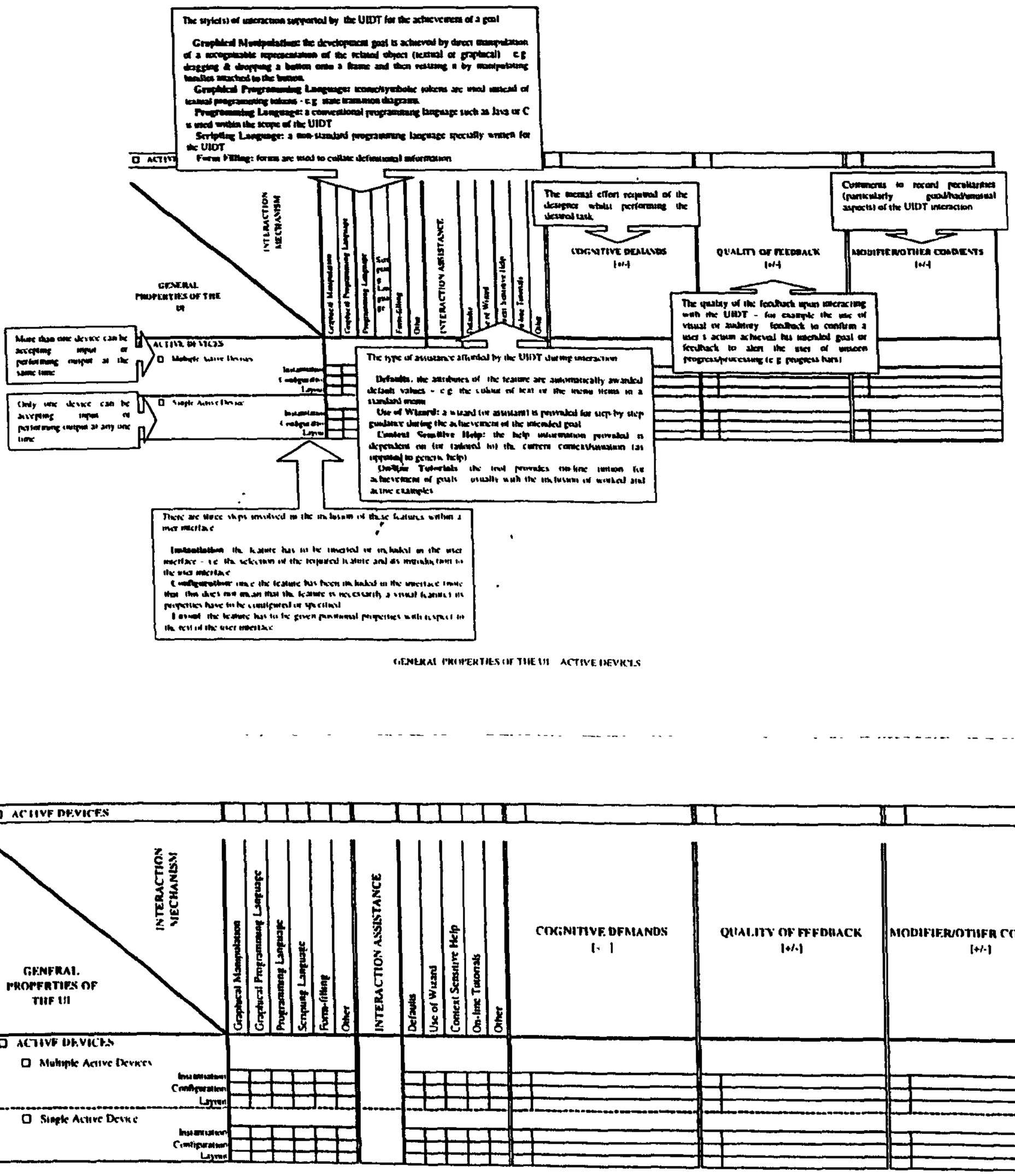


SUIT
GENERAL PROPERTIES OF THE UI - SCREEN NAVIGATION
Page 1

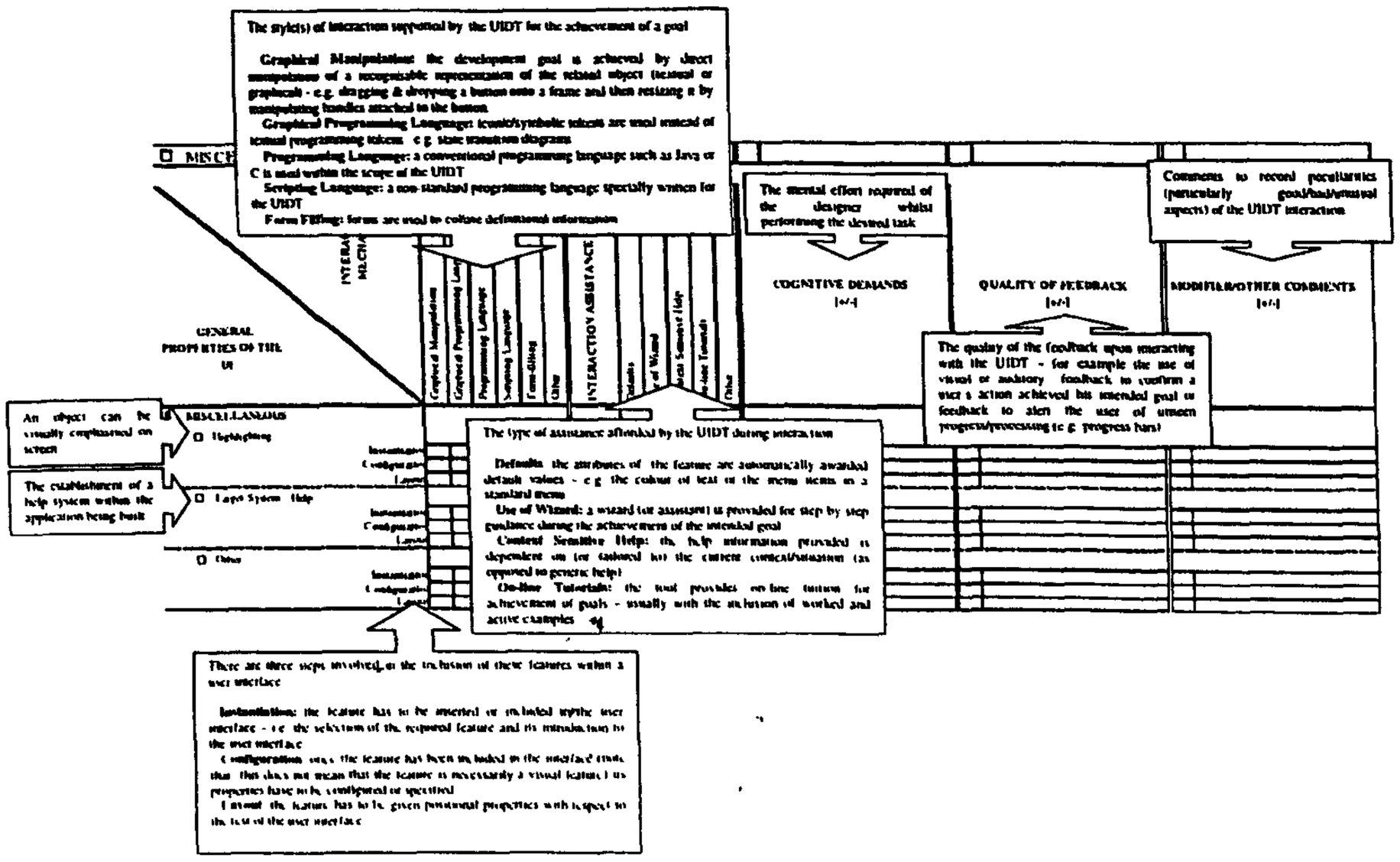

DIALOGUE is concerned with the way in which the "conversation" (or interaction) between the user and the system is structured or controlled
The style(s) of interaction supported by the UIDT for the achievement of a goal
Graphical Manipulations: the development goal is achieved by direct manipulation of a recognisable representation of the related object (textual or graphical) - e.g dragging & dropping a button onto a frame and then resizing it by manipulating handles attached to the button
Graphical Programming Language: iconic/symbolic tokens are used instead of textual programming tokens - e.g state transition diagrams
Programming Language: a conventional programming language such as Java or C is used within the scope of the UIDT
Scripting Language: a non-standard programming language specially written for the UIDT
Form-Filling: forms are used to collate definitional information
The mental effort required of the designer whilst performing the desired task.
COGNITIVE DEMANDS [+/-]
Comments to record peculiarities (particularly good/bad/unusual aspects) of the UIDT interaction
MODIFIER/OTHER COMMENTS [+/-]
QUALITY OF FEEDBACK [+/-]
The quality of the feedback upon interacting with the UIDT - for example the use of visual or auditory feedback to confirm a user's action achieved (as intended) goal or feedback to alert the user of unseen progress/processing (e.g progress bars)
GENERAL PROPERTIES OF THE UI
The states within the dialogue are sequential
INTERACTION ASSIST
The user can modify all or part of the dialogue
DIALOGUE
End user Customisable
User profile or dialogue state is used to modify dialogue form and content
Adaptive
User can have more than one thread of interaction & actively change between them
Multi-thread
Dialogue can be interrupted by outside events
Asynchronous
Dialogue can be interrupted in response to a user action
End user Interruptible
The type of assistance afforded by the UIDT during interaction
Defaults: the attributes of the feature are automatically awarded default values - e.g the colour of text or the menu items in a standard menu
Use of Wizard: a wizard (or assistant) is provided for step-by-step guidance during the achievement of the intended goal
Context Sensitive Help: the help information provided is dependent on (or tailored to) the current context/situation (as opposed to generic help)
On-line Tutorials: the tool provides on-line tuition for achievement of goals - usually with the inclusion of verbal and active examples
There are three steps involved in the inclusion of these features within a user interface :
Instantiation: the feature has to be invoked or included in the user interface - i.e the selection of the required feature and its introduction to the user interface
Configuration: once the feature has been included in the interface (note that this does not mean that the feature is necessarily a visual feature) its properties have to be configured or specified
Layout: the feature has to be given positional properties with respect to the rest of the user interface

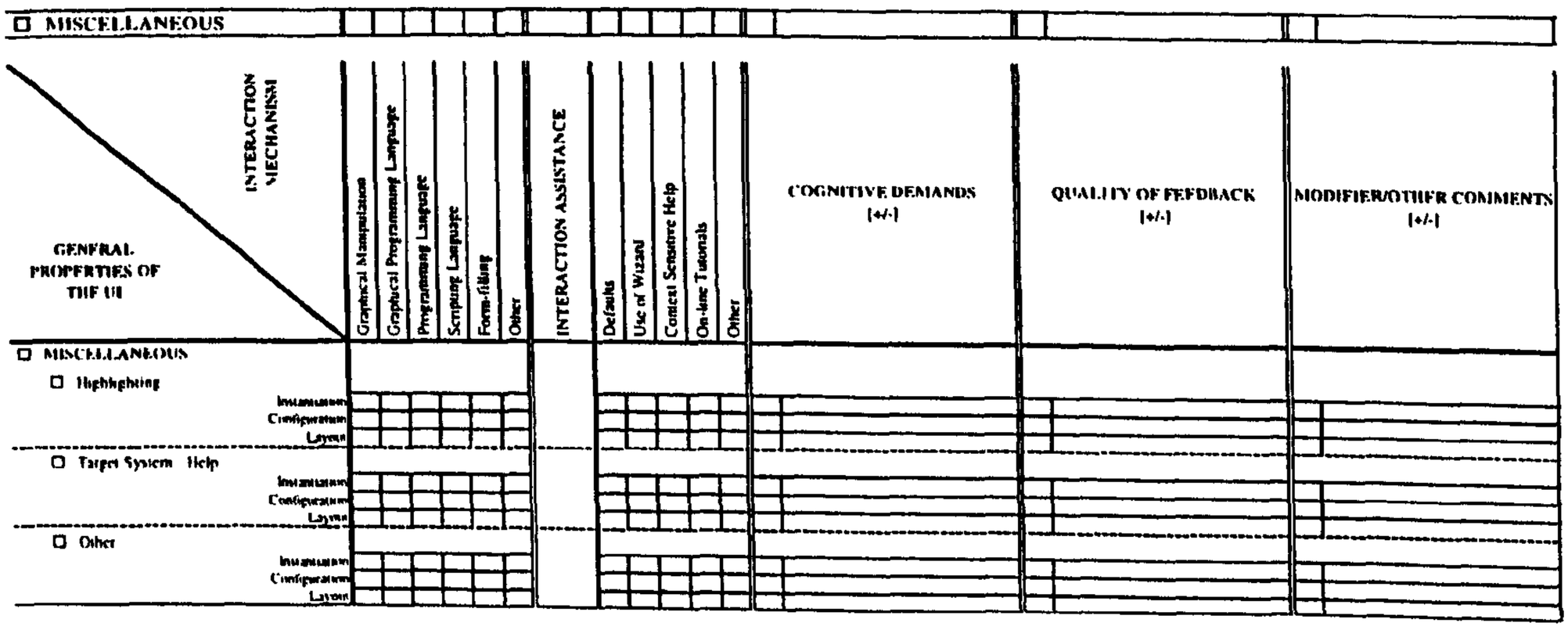

DIALOGUE
INTERACTION MECHANISM
GENERAL PROPERTIES OF THE UI
Graphical Manipulations
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
Linear Dialogue
Hierarchical Dialogue
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
DIALOGUE
End user Customisable
Instantiation
Configuration
Layout
Adaptive
Instantiation
Configuration
Layout
Multi-thread
Instantiation
Configuration
Layout
Asynchronous
Instantiation
Configuration
Layout
End user Interruptible
Instantiation
Configuration
Layout
Other
Instantiation
Configuration
Layout

ACTIVE DEVICES is concerned with the multiplicity of virtual or physical devices which can be accepting input or performing output at the same time.

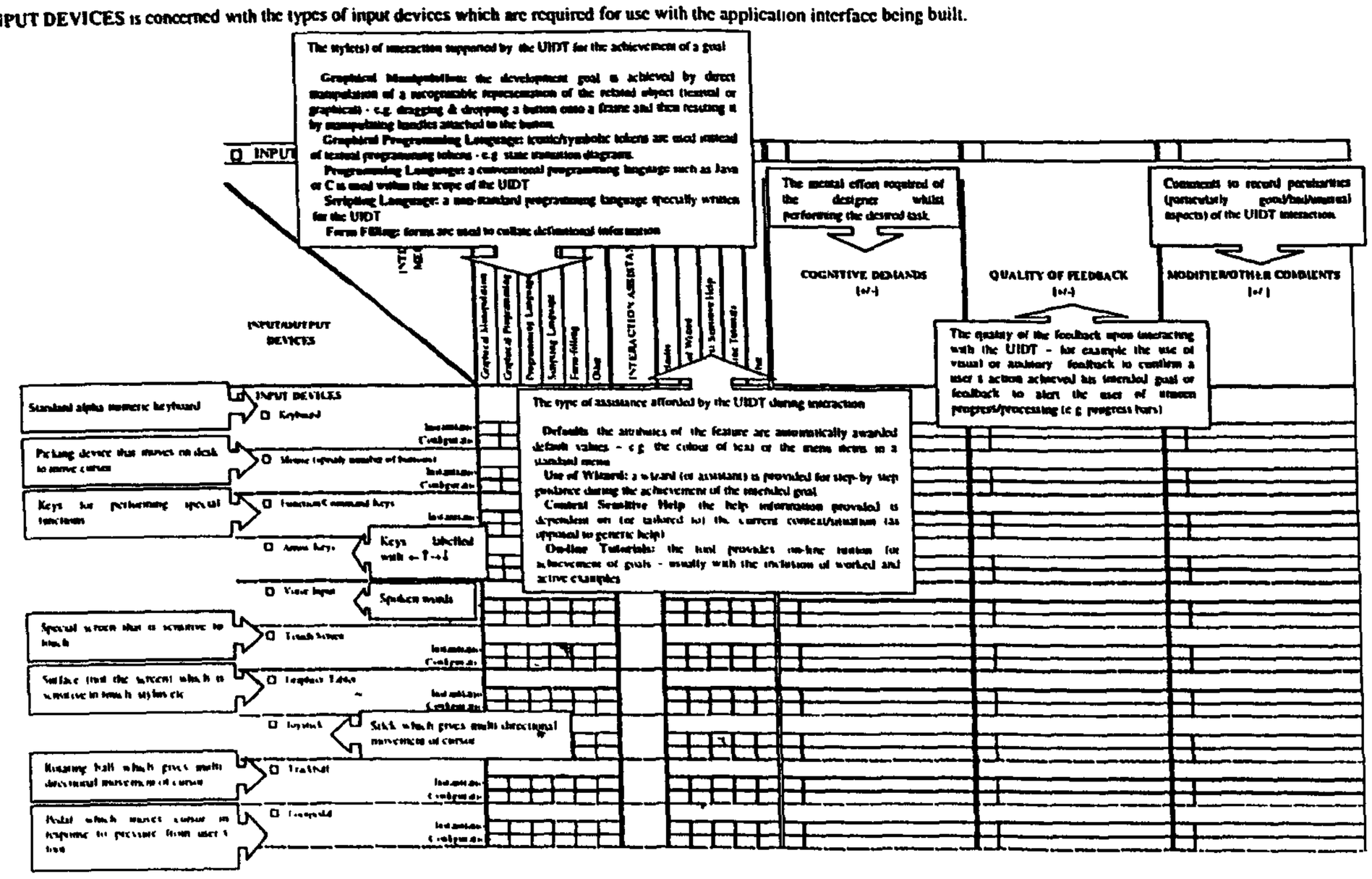


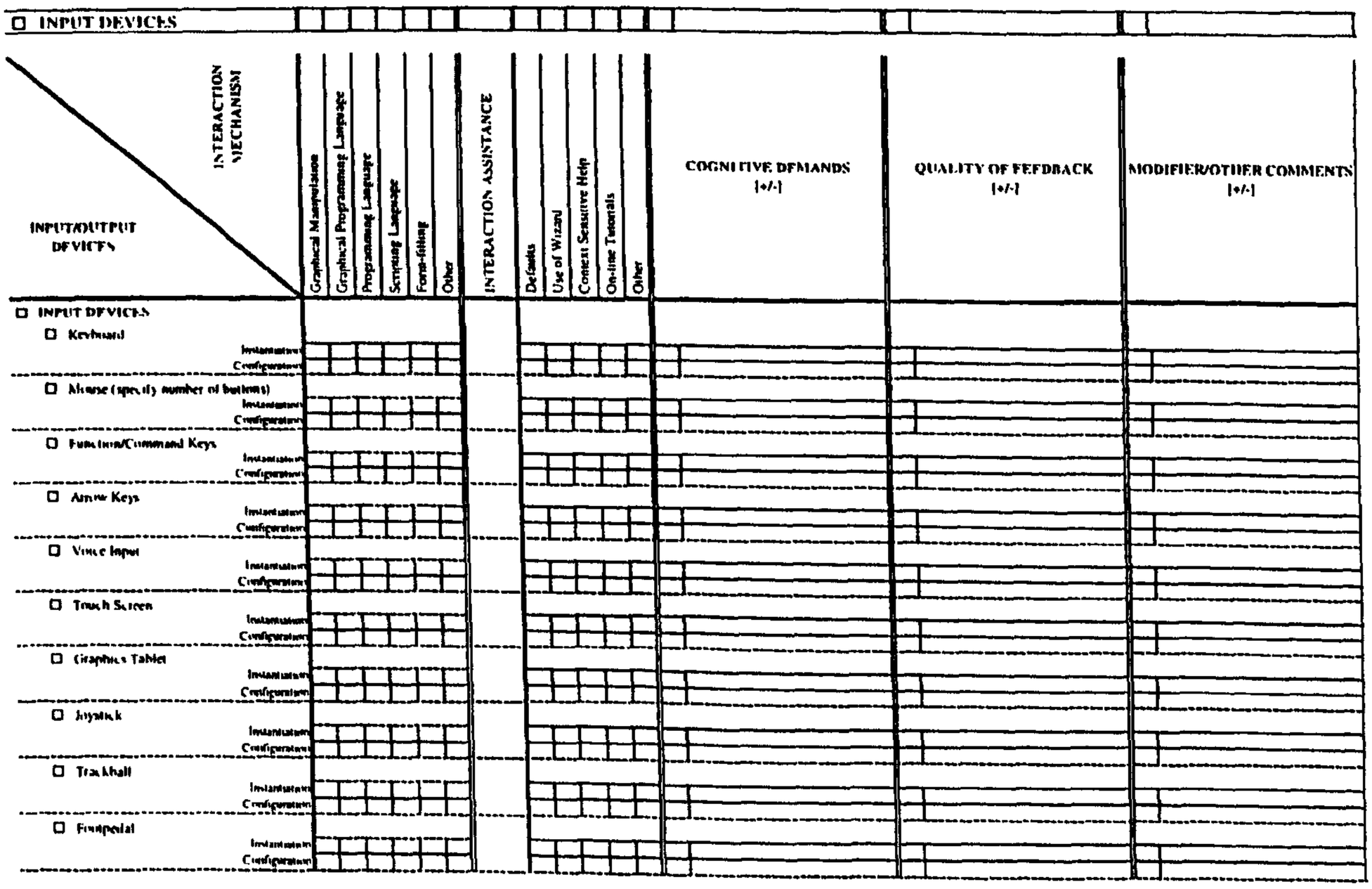
MISCELLANEOUS is concerned with those general properties of the user interface which do not easily fall into one of the other categories.

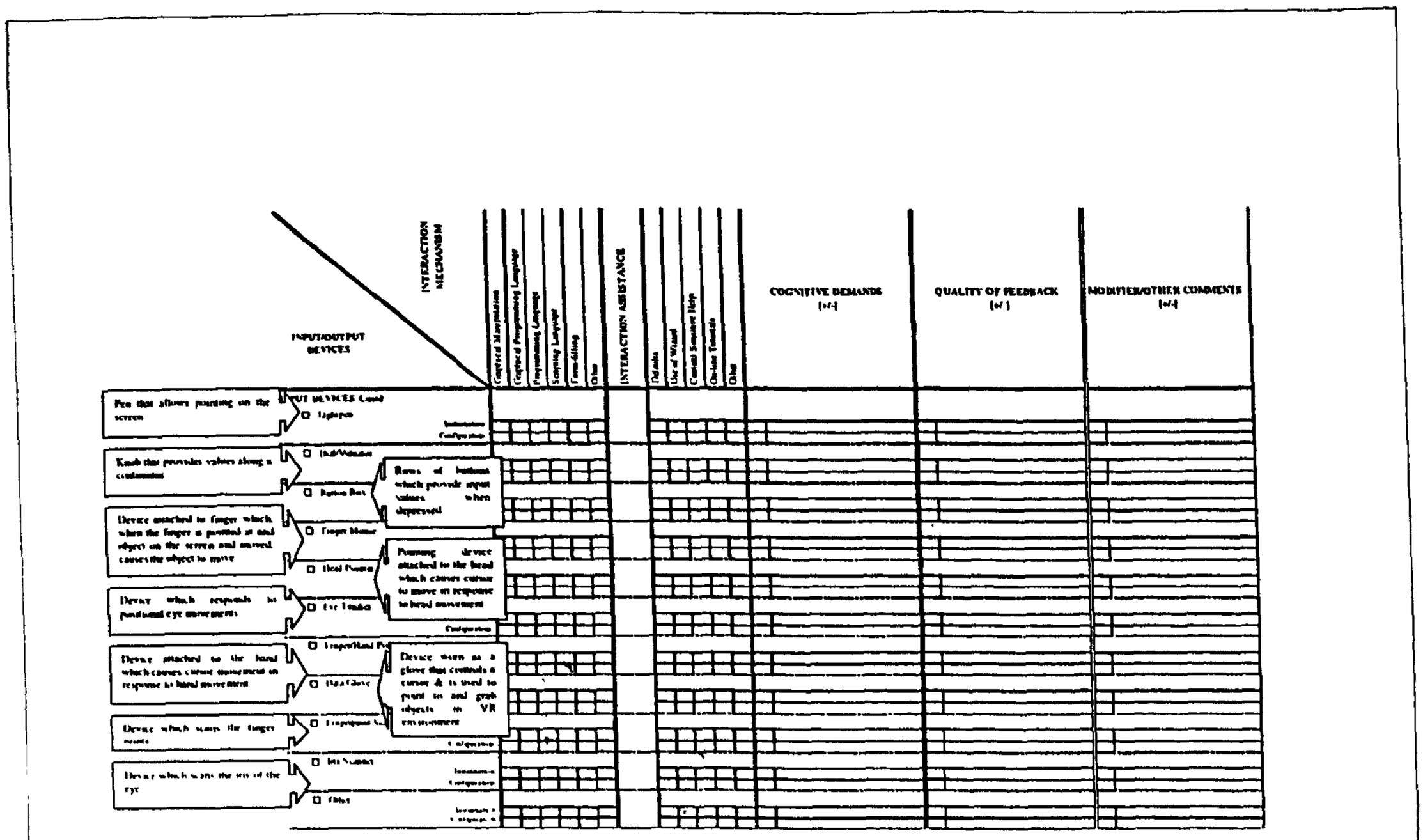




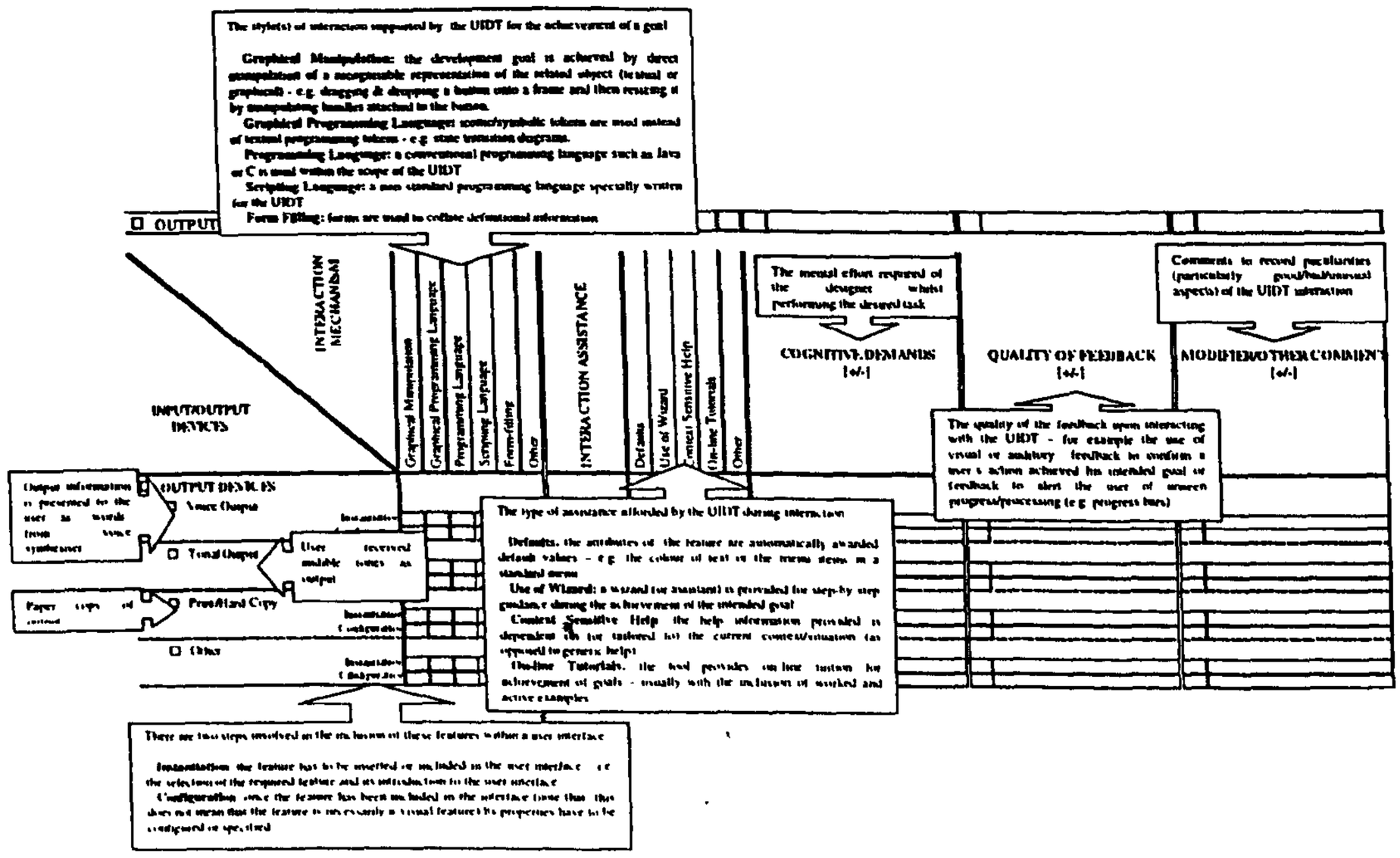
INPUT DEVICES is concerned with the types of input devices which are required for use with the application interface being built.

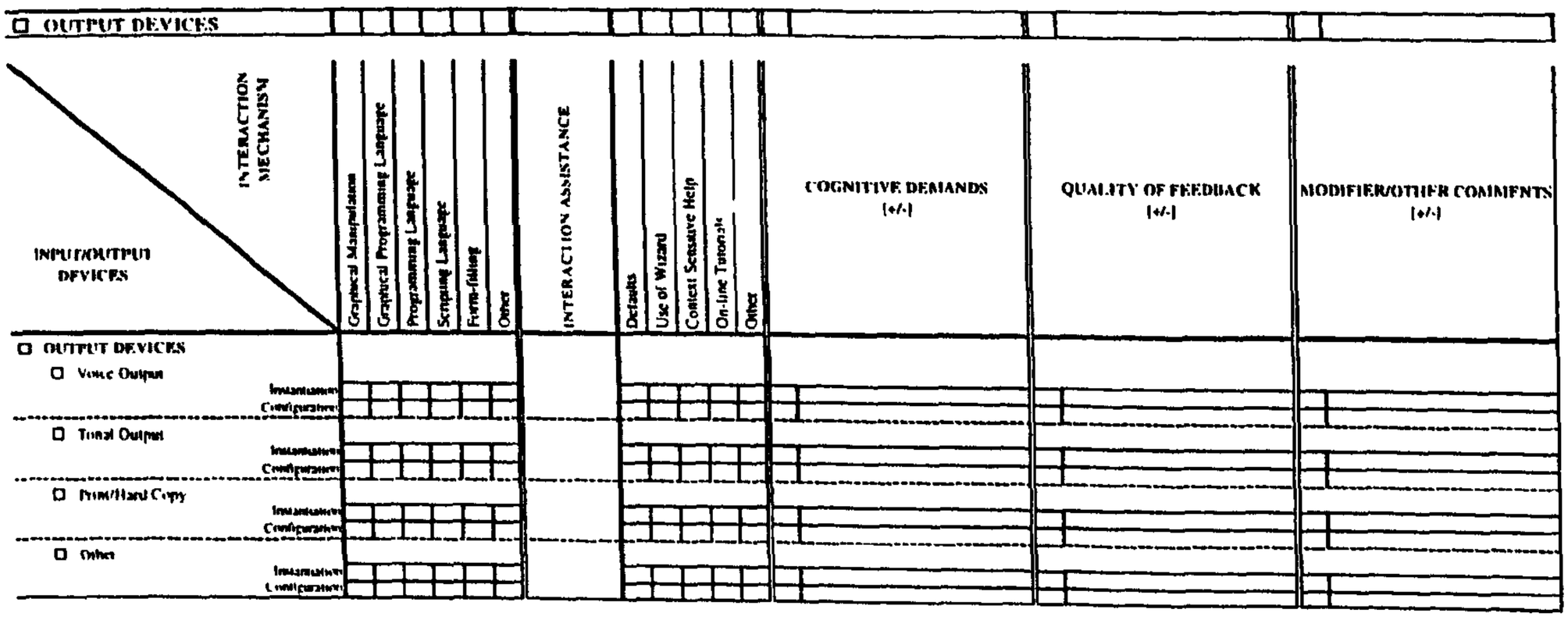




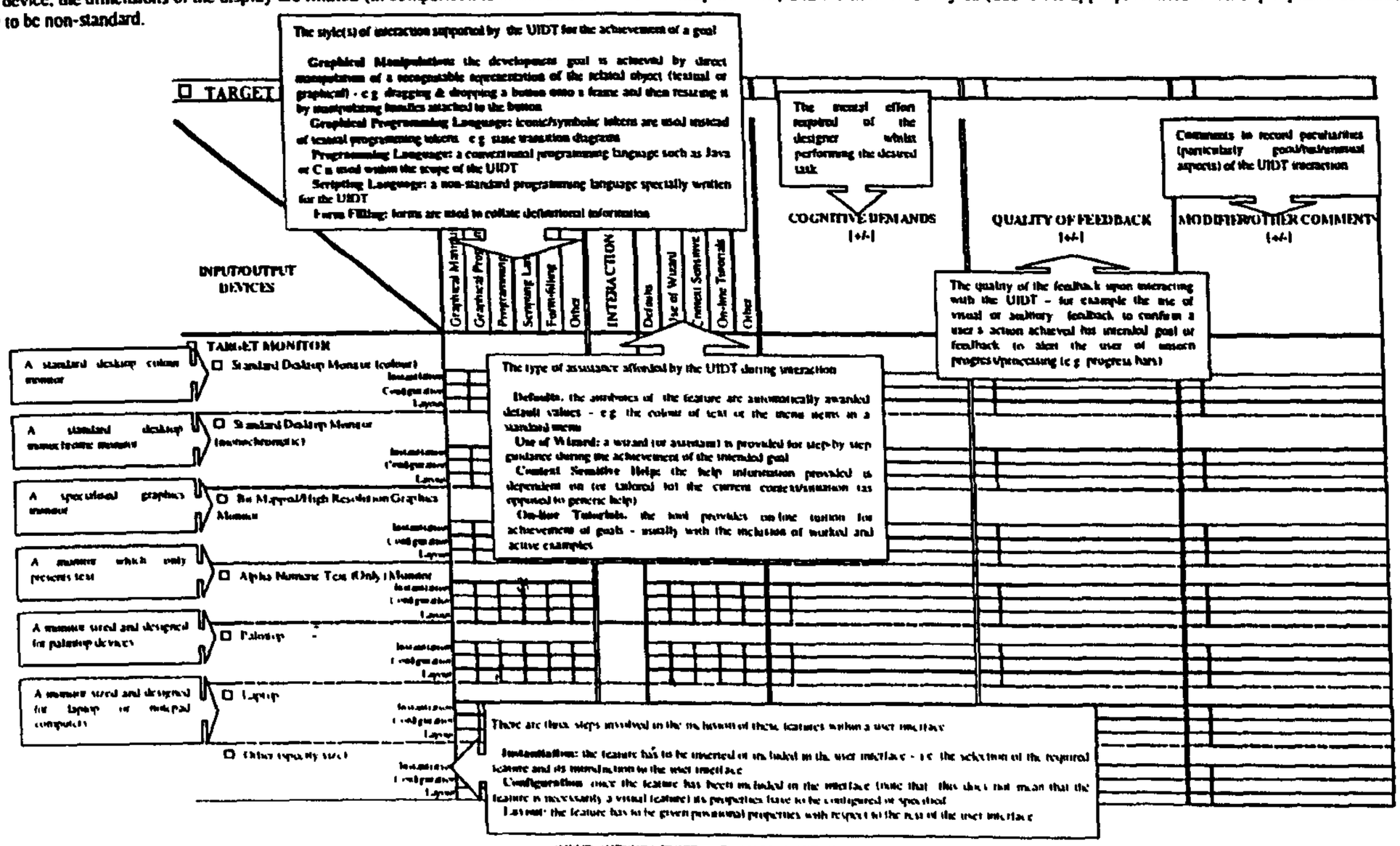

INPUT/OUTPUT DEVICES
INTERACTION MECHANISM
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
INPUT DEVICES Contd.
Lightpen
Dial/Valuator
Button Box
Finger Mouse
Head Pointer
Eye Tracker
Finger/Hand Pointer
Data Glove
Fingerprint Scanner
Iris Scanner
Other
Instantiation
Configuration
Pen that allows pointing on the screen
Knob that provides values along a continuum
Device attached to finger which, when the finger is pointed at and object on the screen and moved, causes the object to move
Device which responds to positional eye movements
Device attached to the hand which causes cursor movement in response to hand movement
Device which scans the finger prints
Device which scans the iris of the eye
Buttons of instrument which provide input values when depressed
Pointing device attached to the head which causes cursor to move in response to head movement
Device worn as a glove that controls a cursor & is used to point to and grab objects in VR environment
SUIT
INPUT/OUTPUT DEVICES - INPUT DEVICES
Page G6

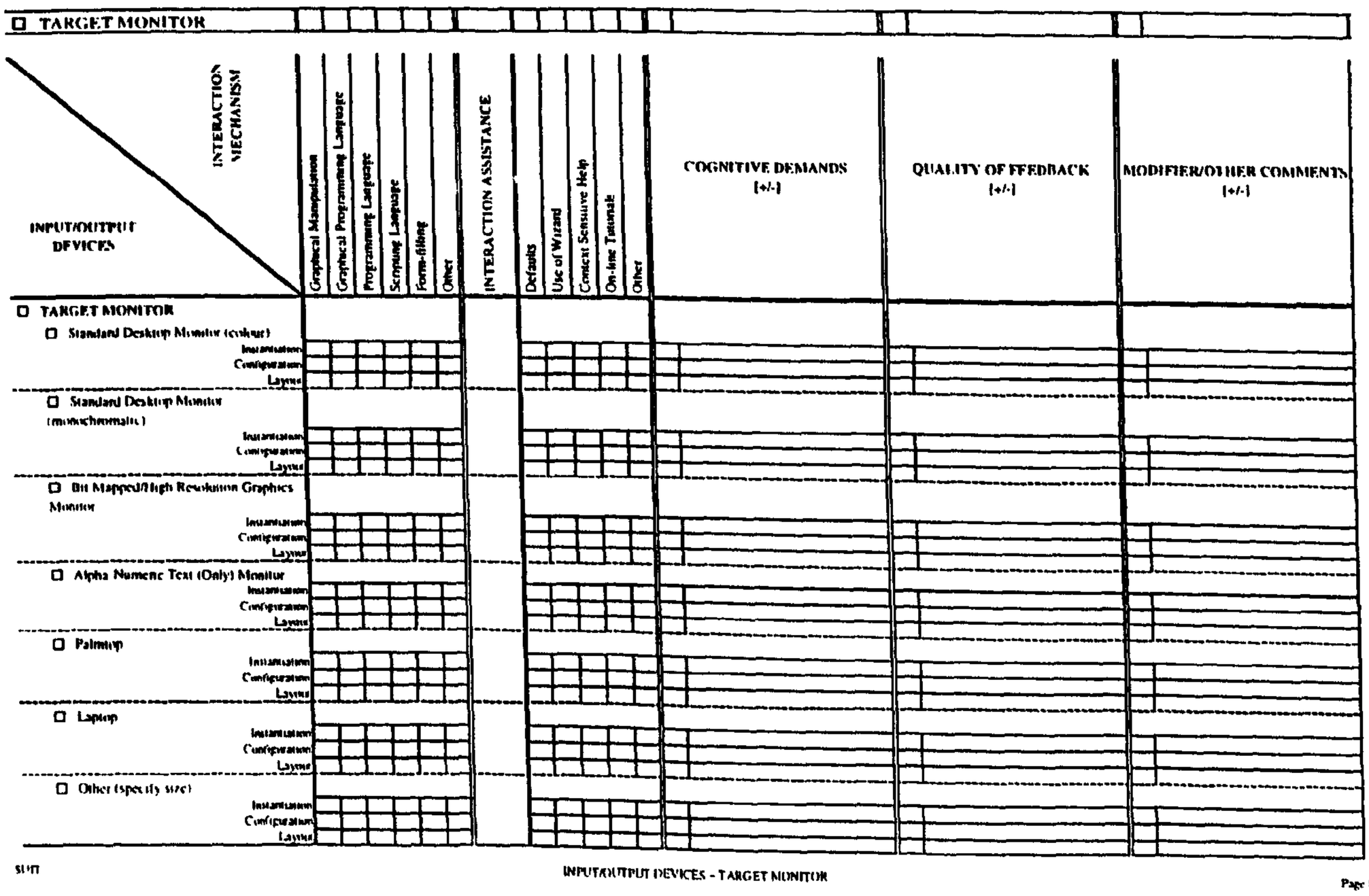
OUTPUT DEVICES is concerned with the types of output devices which are required for use with the application interface being built.



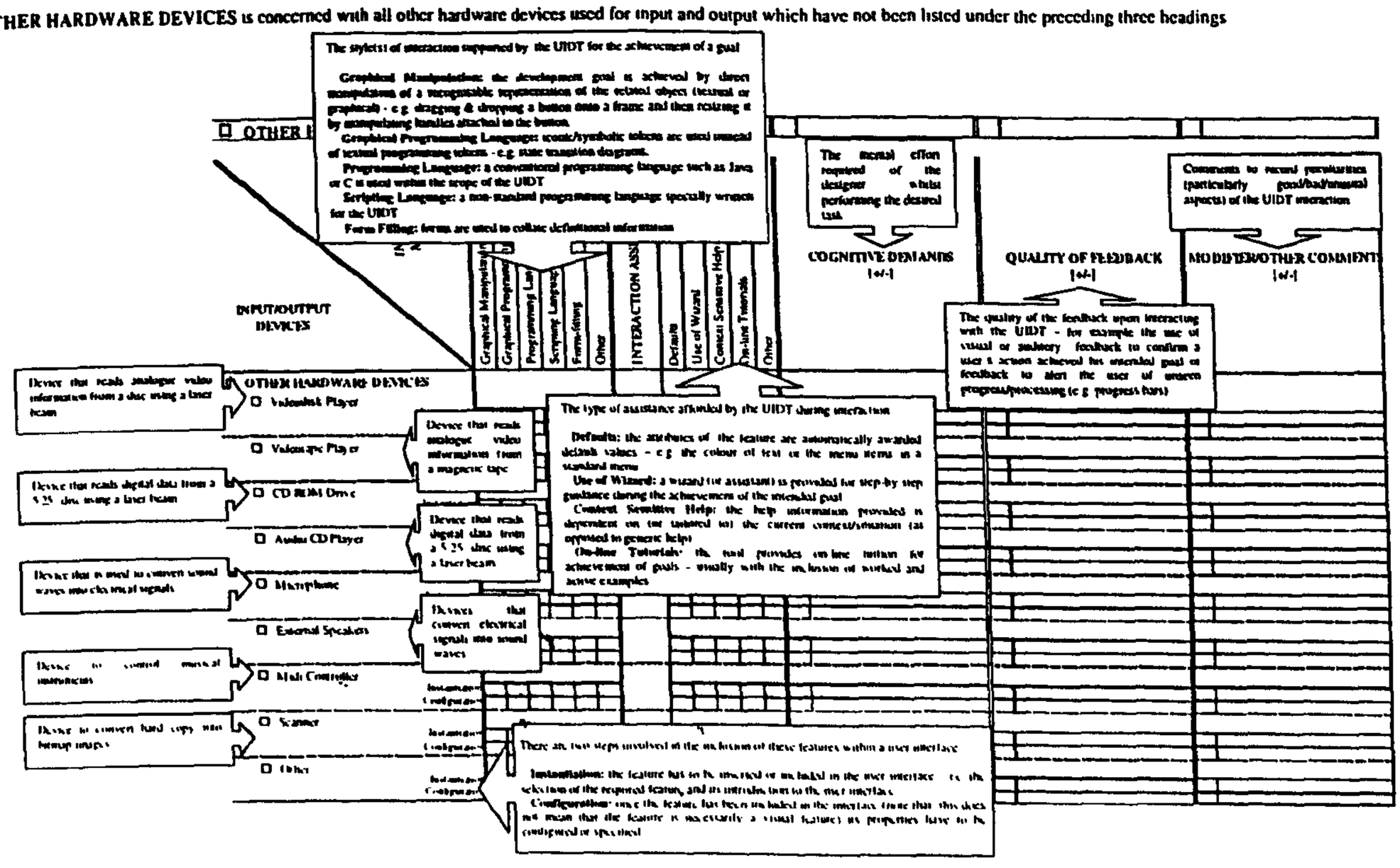


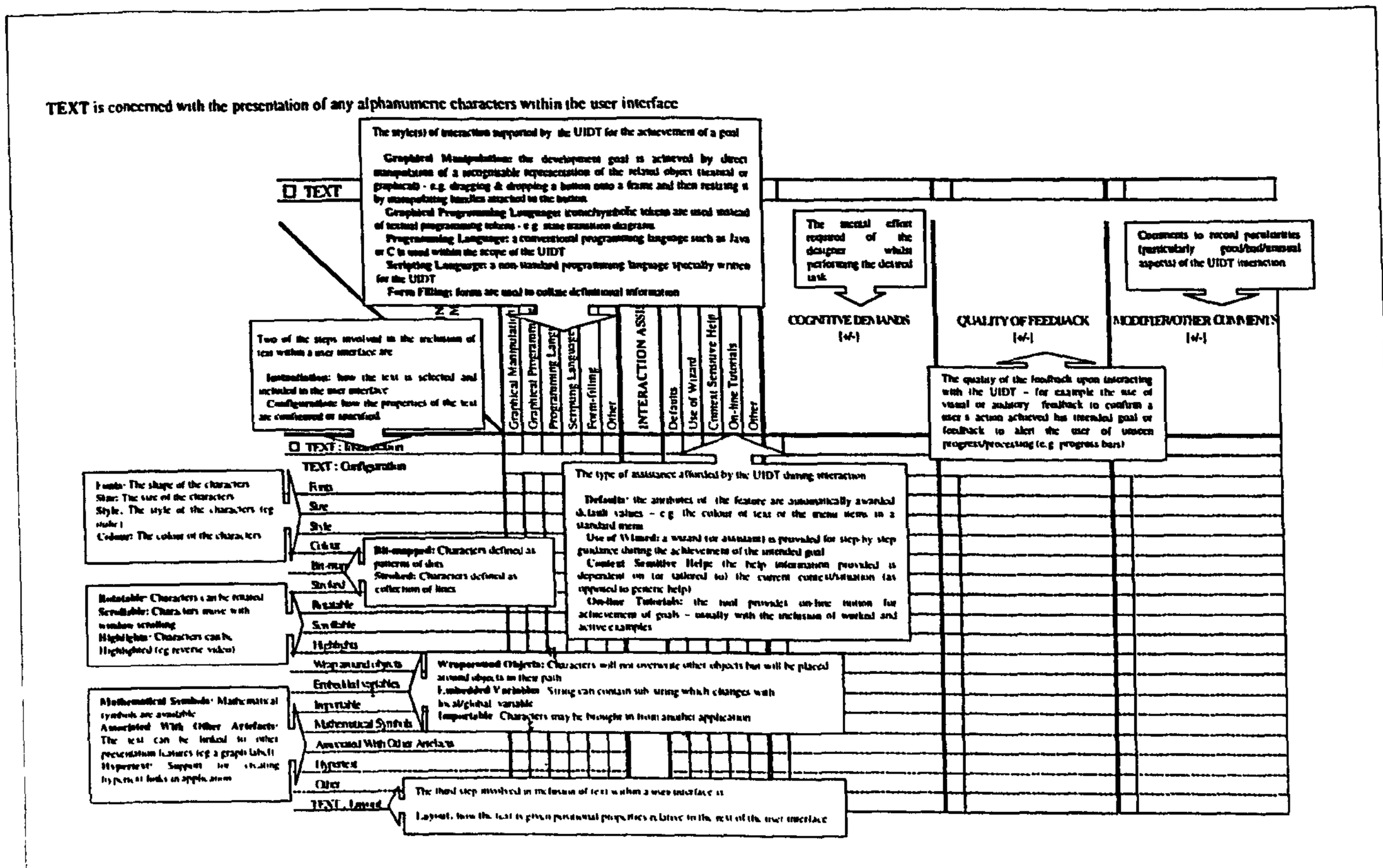
TARGET MONITORS is concerned with the type of monitor which will be used by the system running the application for which the user interface is being developed The nature of the target monitor has the potential to exert substantial influence over the appropriate features and interaction styles included within the user interface - for example, if an application is being developed for a palm-top device, the dimensions of the display are limited (in comparison to those of a standard desktop monitor) and the interaction styles (due to the appropriateness of other peripheral devices) are likely to be non-standard.

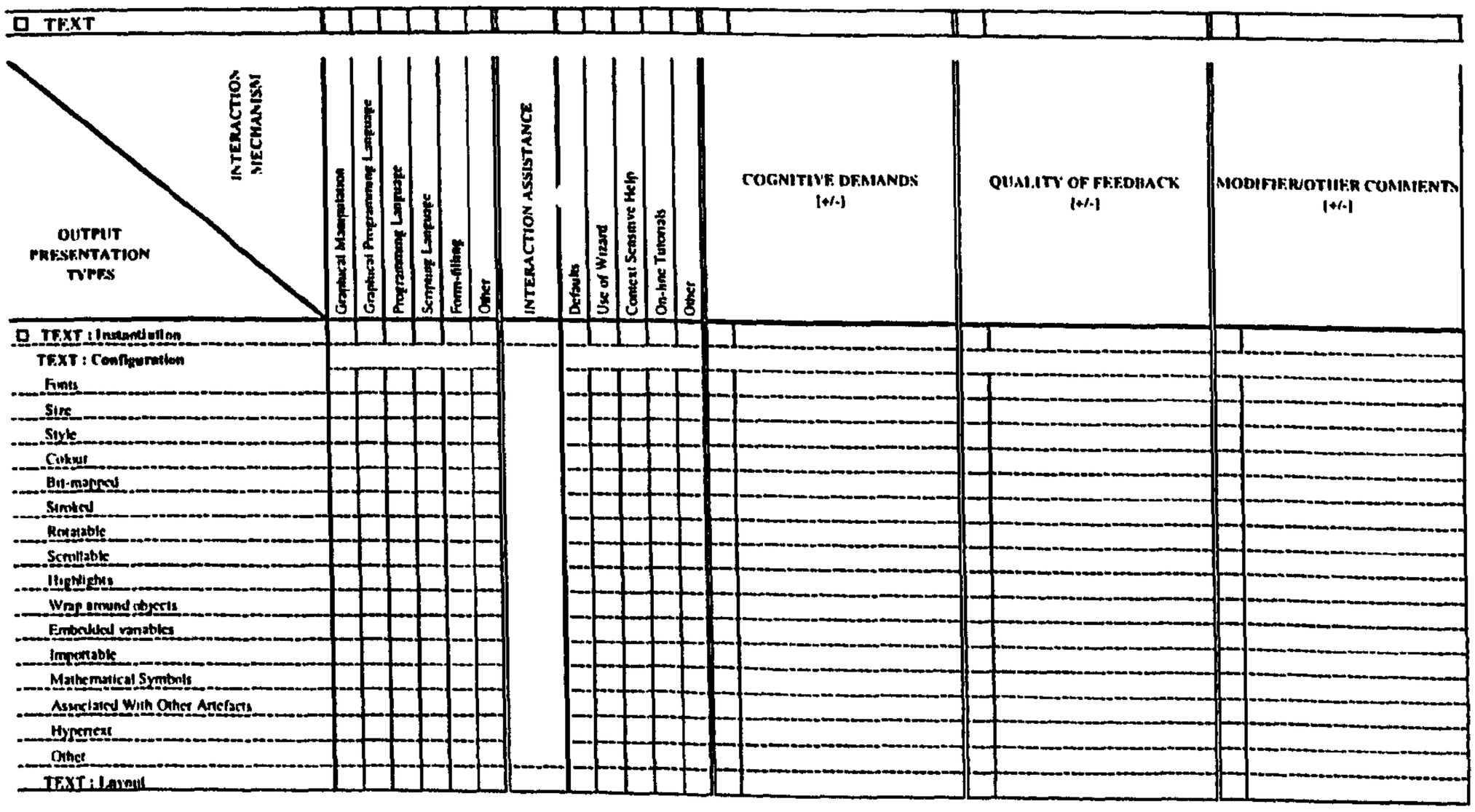


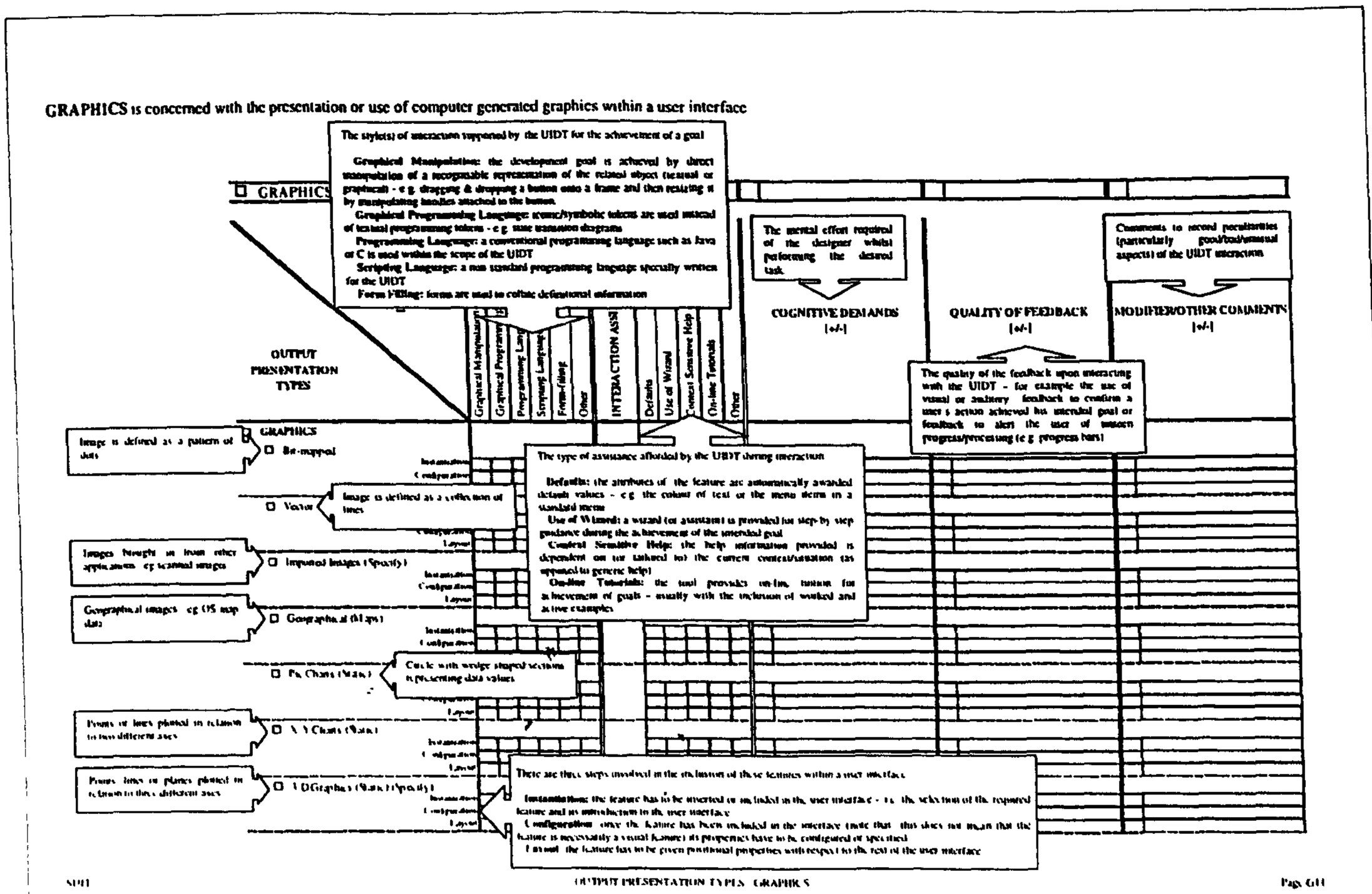


OTHER HARDWARE DEVICES is concerned with all other hardware devices used for input and output which have not been listed under the preceding three headings

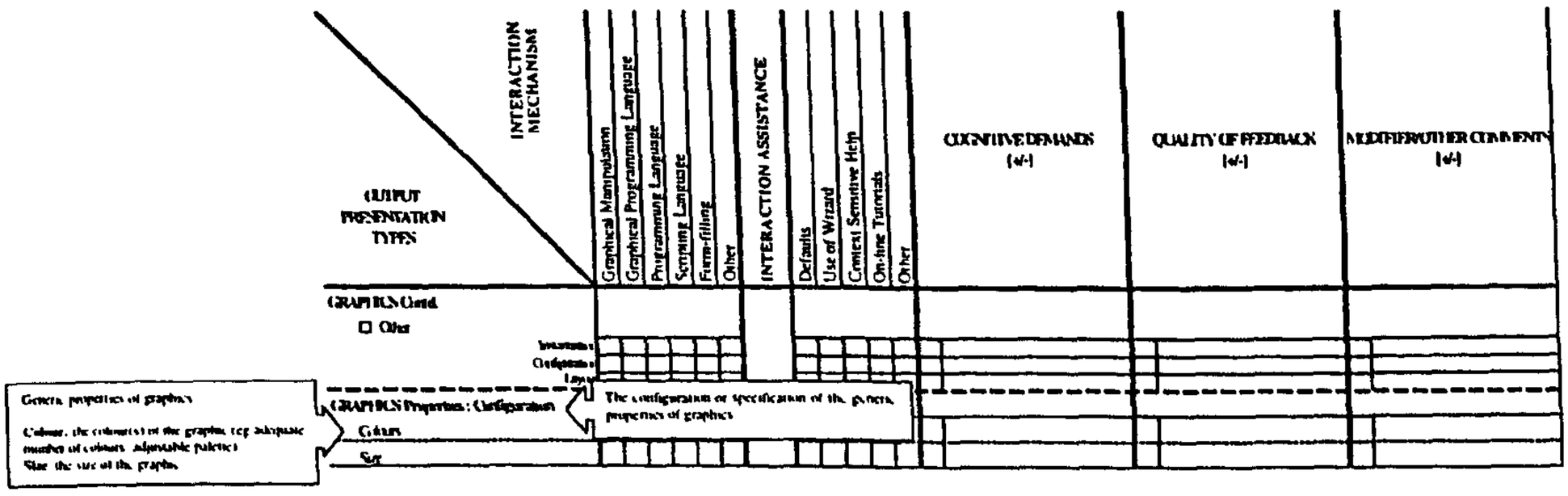


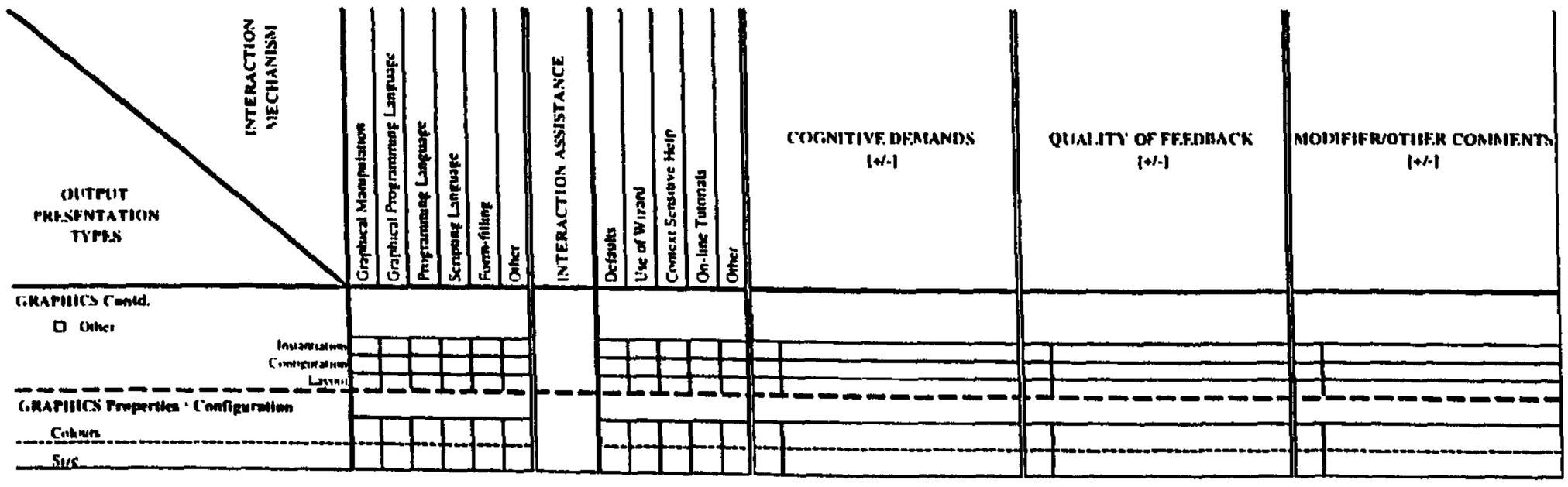

TEXT is concerned with the presentation of any alphanumeric characters within the user interface
TEXT
The style(s) of interaction supported by the UIDT for the achievement of a goal
Graphical Manipulation: the development goal is achieved by direct manipulation of a recognisable representation of the related object (textual or graphical) - e.g. dragging & dropping a button onto a frame and then resizing it by manipulating handles attached to the button
Graphical Programming Language: iconic/symbolic tokens are used instead of textual programming tokens - e.g. state transition diagrams
Programming Language: a conventional programming language such as Java or C is used within the scope of the UIDT
Scripting Language: a non-standard programming language specially written for the UIDT
Form Filling: forms are used to collate definitional information
The mental effort required of the designer whilst performing the desired task
COGNITIVE DEMANDS [+/-]
Comments to record peculiarities (particularly good/bad/unusual aspects) of the UIDT interaction
MODIFIER/OTHER COMMENTS [+/-]
Two of the steps involved in the inclusion of text within a user interface are
Instantiation: how the text is selected and included in the user interface
Configuration: how the properties of the text are confirmed or specified
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
QUALITY OF FEEDBACK [+/-]
The quality of the feedback upon interacting with the UIDT - for example the use of visual or auditory feedback to confirm a user's action achieved has intended goal or feedback to alert the user of unseen progress/processing (e.g. progress bars)
TEXT : Instantiation
TEXT : Configuration
Fonts: The shape of the characters
Size: The size of the characters
Style: The style of the characters (eg italic)
Colour: The colour of the characters
Fonts
Size
Style
Colour
Bit-mapped: Characters defined as patterns of dots
Stroked: Characters defined as collection of lines
Bit-mapped
Stroked
Rotatable: Characters can be rotated
Scrollable: Characters move with window scrolling
Highlights: Characters can be highlighted (eg reverse video)
Rotatable
Scrollable
Highlights
The type of assistance afforded by the UIDT during interaction
Defaults: the attributes of the feature are automatically awarded default values - e.g. the colour of text or the menu items in a standard menu
Use of Wizard: a wizard (or assistant) is provided for step-by step guidance during the achievement of the intended goal
Context Sensitive Help: the help information provided is dependent on (or tailored to) the current context/situation (as opposed to generic help)
On-line Tutorials: the tool provides on-line tuition for achievement of goals - usually with the inclusion of worked and active examples
Wrap around objects: Characters will not overwrite other objects but will be placed around objects in their path
Embedded Variables: String can contain sub string which changes with local/global variable
Importable: Characters may be brought in from another application
Wrap around objects
Embedded variables
Importable
Mathematical Symbols: Mathematical symbols are available
Associated With Other Artefacts: The text can be linked to other presentation features (eg a graphic label)
Hypertext: Support for creating hypertext links in application
Mathematical Symbols
Associated With Other Artefacts
Hypertext
Other
TEXT : Layout
The final step involved in inclusion of text within a user interface is
Layout: how the text is given positional properties relative to the rest of the user interface
SUIT
OUTPUT PRESENTATION TYPES TEXT
Page G10

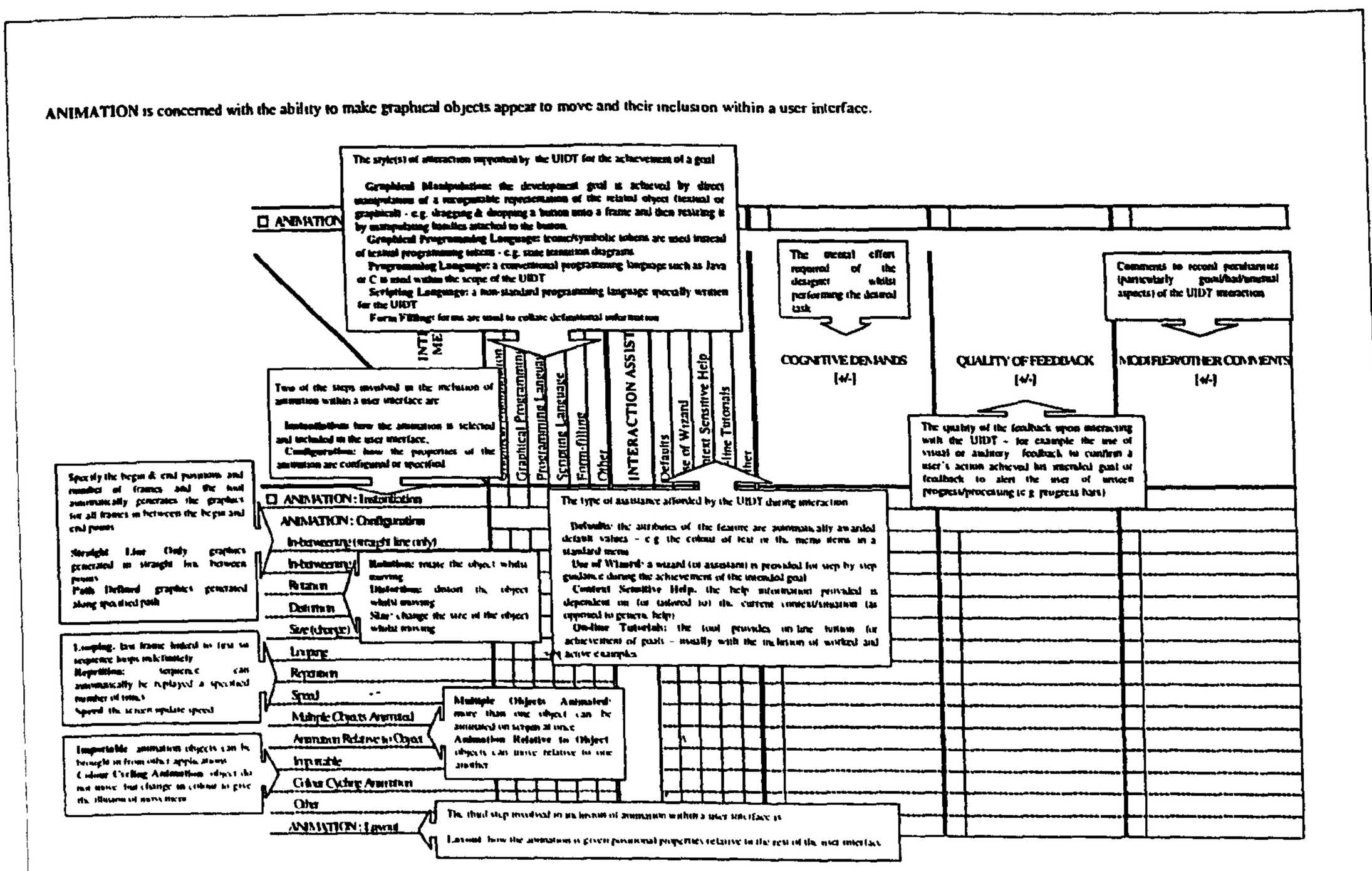

TEXT
OUTPUT PRESENTATION TYPES
INTERACTION MECHANISM
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
TEXT : Instantiation
TEXT : Configuration
Fonts
Size
Style
Colour
Bit-mapped
Stroked
Rotatable
Scrollable
Highlights
Wrap around objects
Embedded variables
Importable
Mathematical Symbols
Associated With Other Artefacts
Hypertext
Other
TEXT : Layout

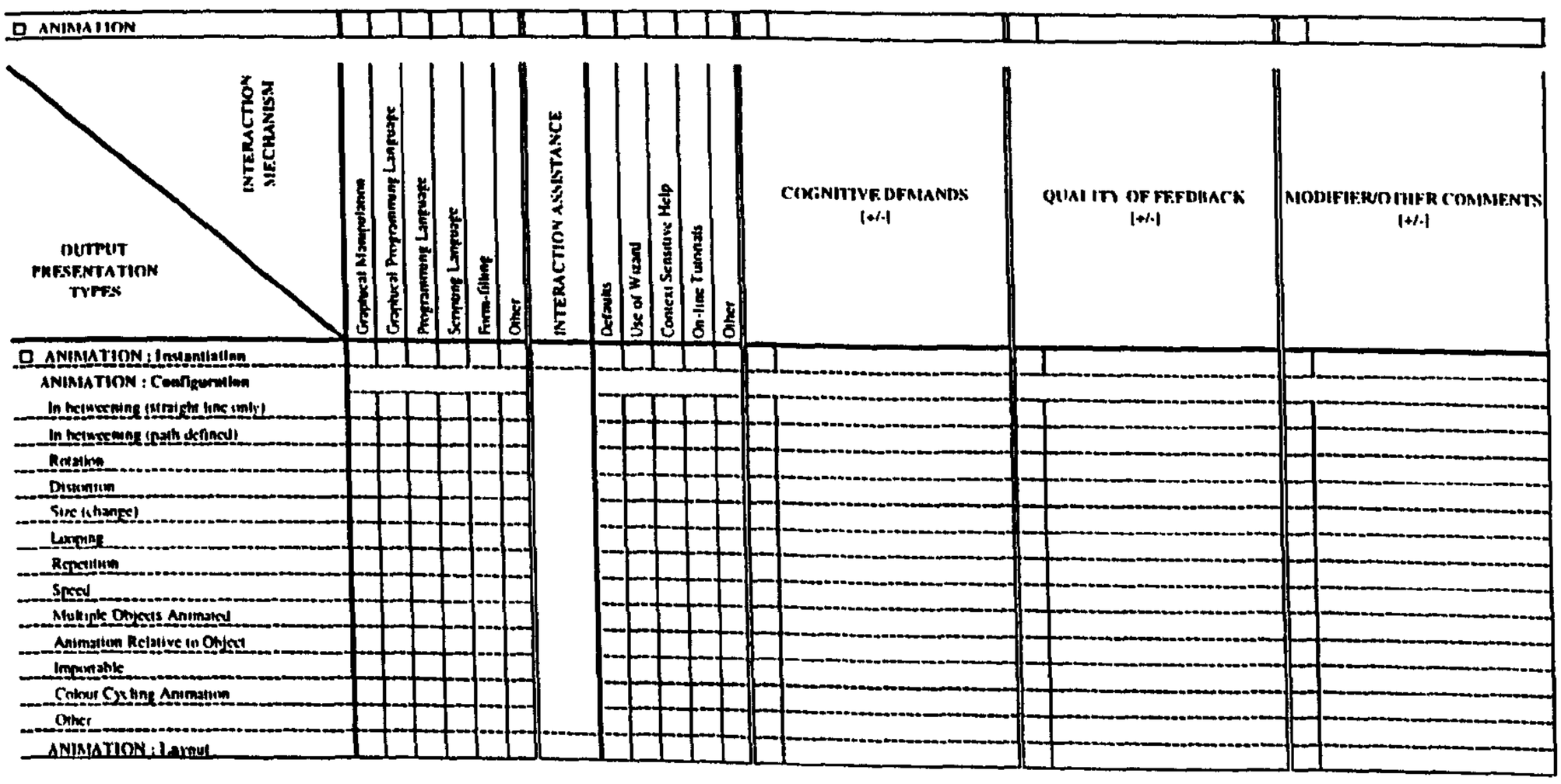


GRAPHICS
INTERACTION MECHANISM
OUTPUT PRESENTATION TYPES
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
GRAPHICS
Bit-mapped
Vector
Imported Images (Specify)
Geographical (Maps)
Pie Charts (Static)
X Y Charts (Static)
3D Graphics (Static) (Specify)
Instantiation
Configuration
Layout


INTERACTION MECHANISM
OUTPUT PRESENTATION TYPES
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS (+/-)
QUALITY OF FEEDBACK (+/-)
MODIFIER/OTHER COMMENTS (+/-)
GRAPHICS Contd.
Other
Instantiation
Configuration
Layout
GRAPHICS Properties : Configuration
Colours
Size
General properties of graphics
Colour: the colour(s) of the graphic (eg adequate number of colours, adjustable palette)
Size: the size of the graphic
The configuration or specification of the general properties of graphics

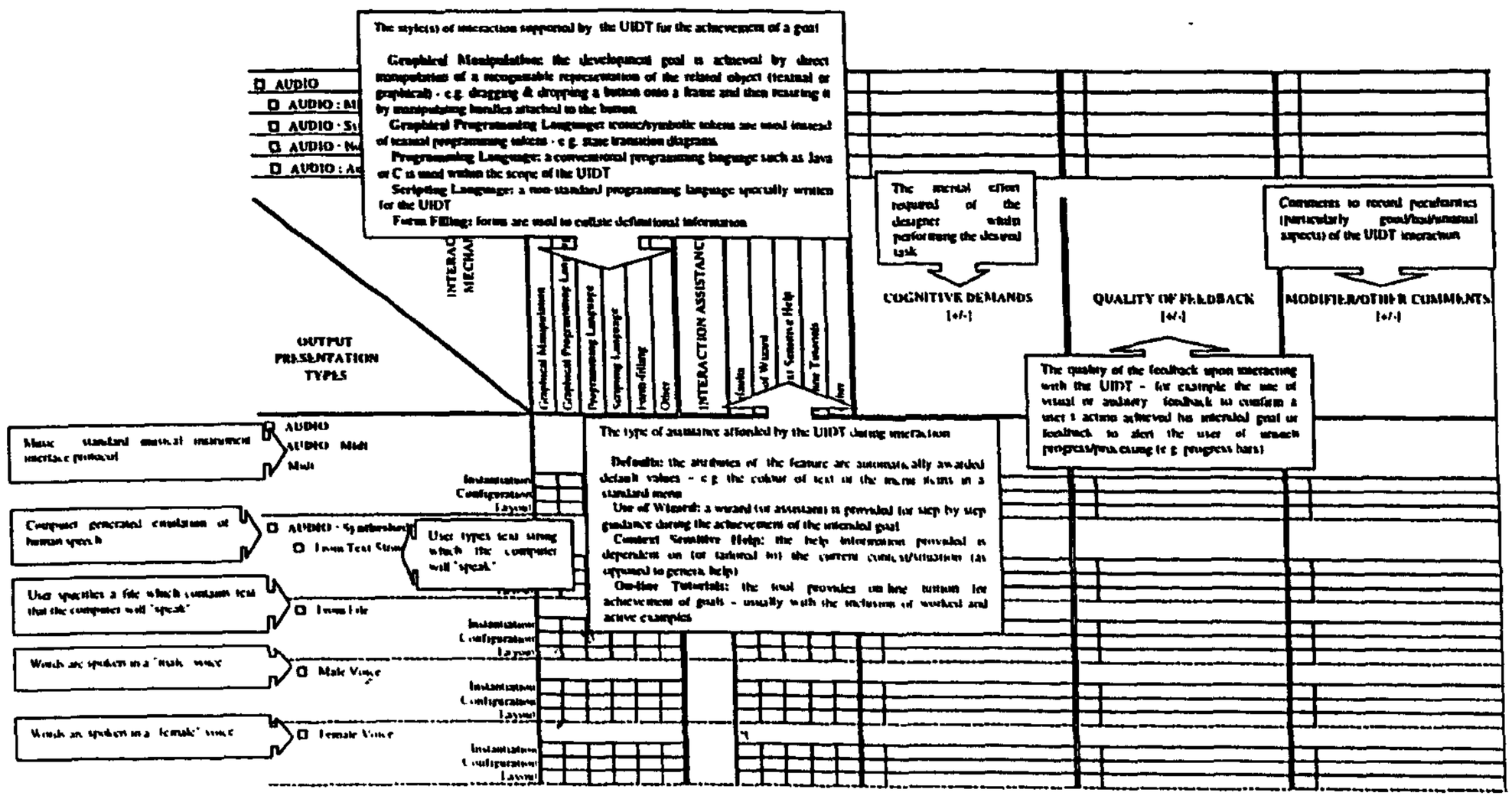

INTERACTION MECHANISM
OUTPUT PRESENTATION TYPES
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS (+/-)
QUALITY OF FEEDBACK (+/-)
MODIFIER/OTHER COMMENTS (+/-)
GRAPHICS Contd.
Other
Instantiation
Configuration
Layout
GRAPHICS Properties : Configuration
Colours
Size

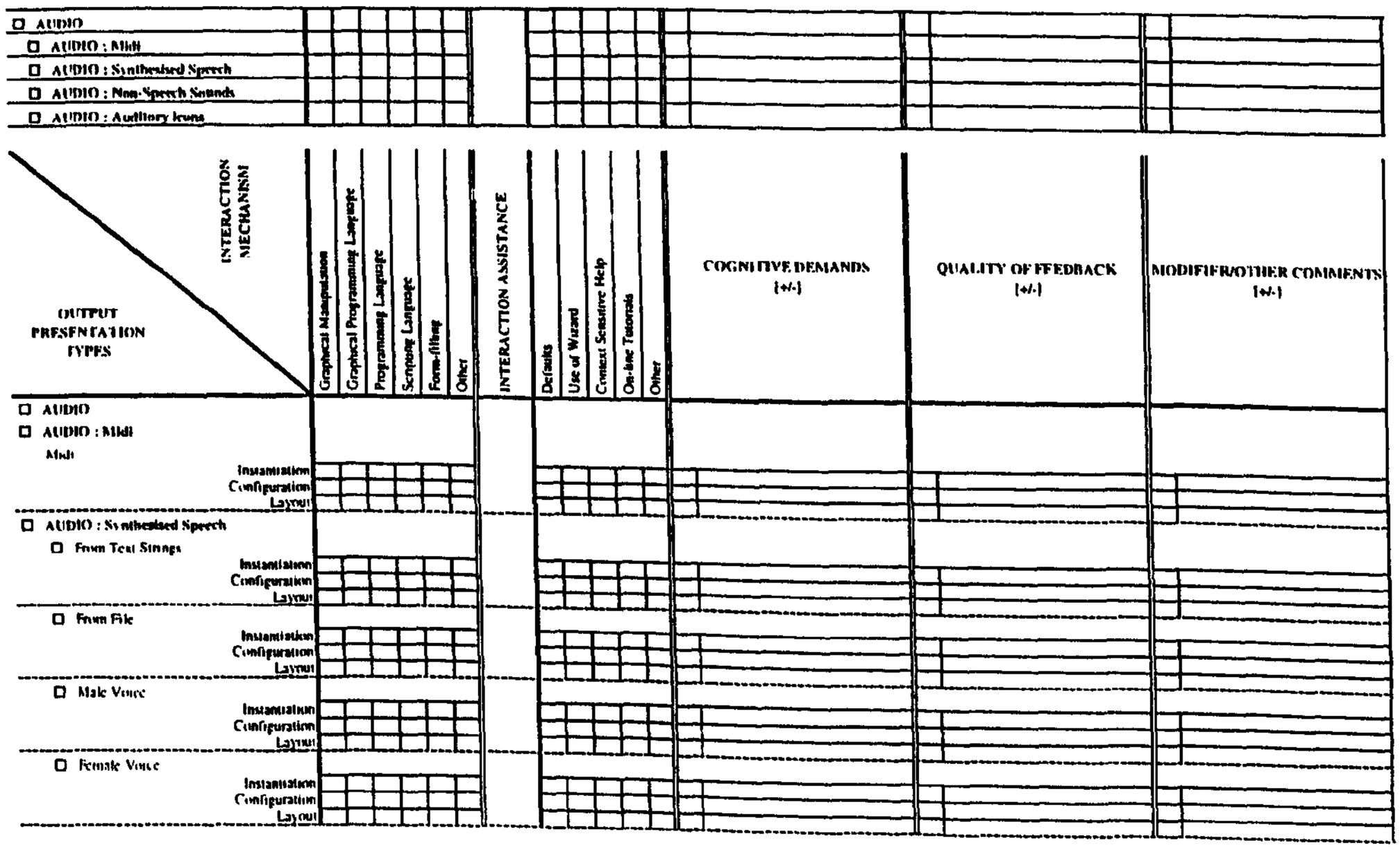


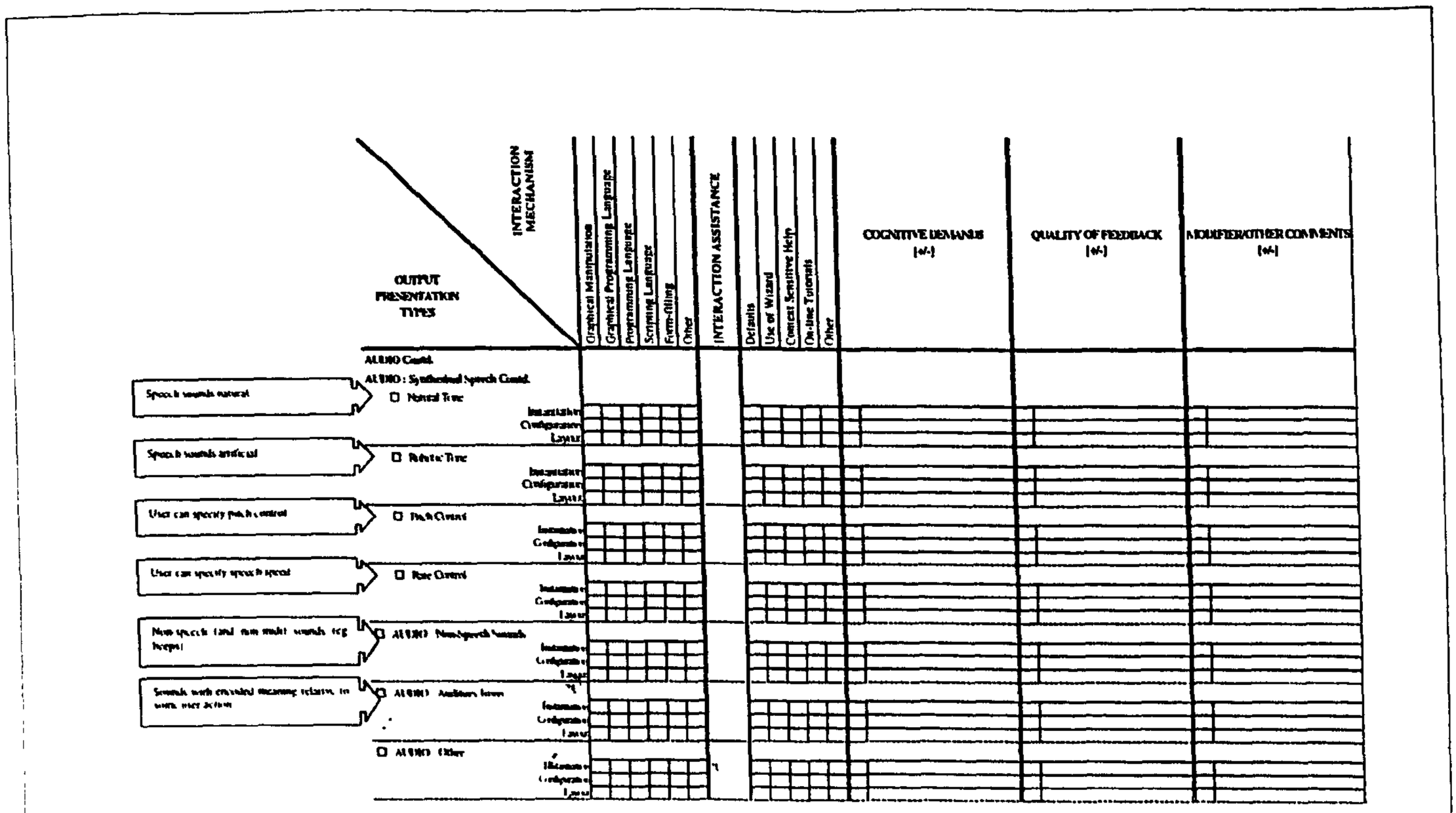


ANIMATION	INTERACTION MECHANISM: Graphical Manipulation	Graphical Programming Language	Programming Language	Scripting Language	Form-filling	Other	INTERACTION ASSISTANCE	Defaults	Use of Wizard	Context Sensitive Help	On-line Tutorials	Other	COGNITIVE DEMANDS [+/-]	QUALITY OF FEEDBACK [+/-]	MODIFIER/OTHER COMMENTS [+/-]
OUTPUT PRESENTATION TYPES															
ANIMATION : Instantiation															
ANIMATION : Configuration															
In betweening (straight line only)															
In betweening (path defined)															
Rotation															
Distortion															
Size (change)															
Looping															
Repetition															
Speed															
Multiple Objects Animated															
Animation Relative to Object															
Importable															
Colour Cycling Animation															
Other															
ANIMATION : Layout															

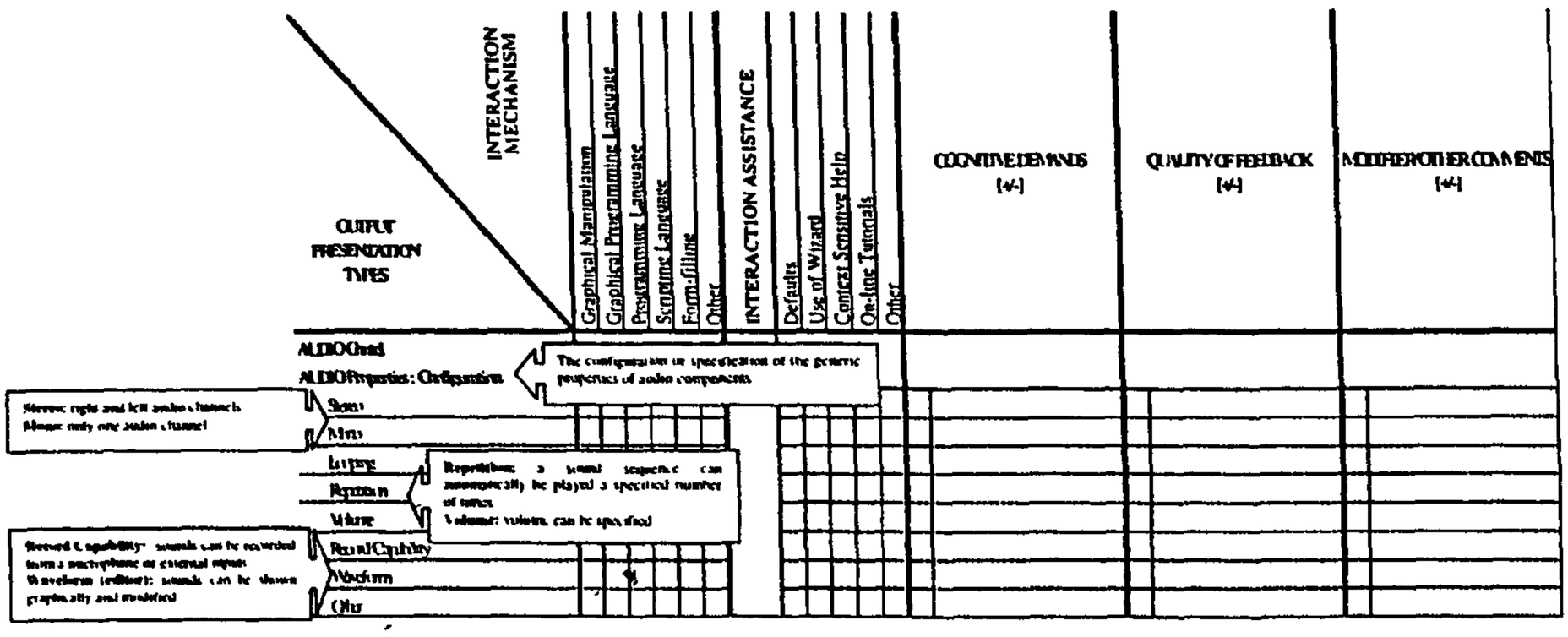
AUDIO is concerned with any kind of sounds and their inclusion within a user interface

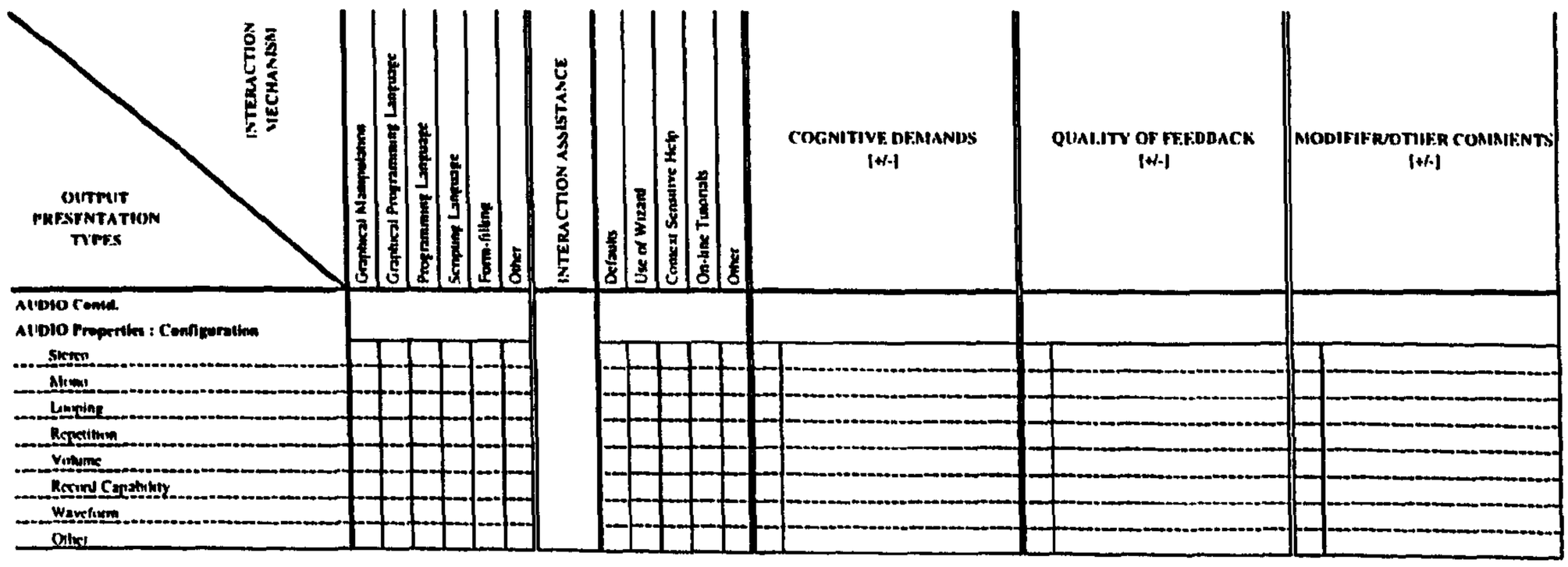




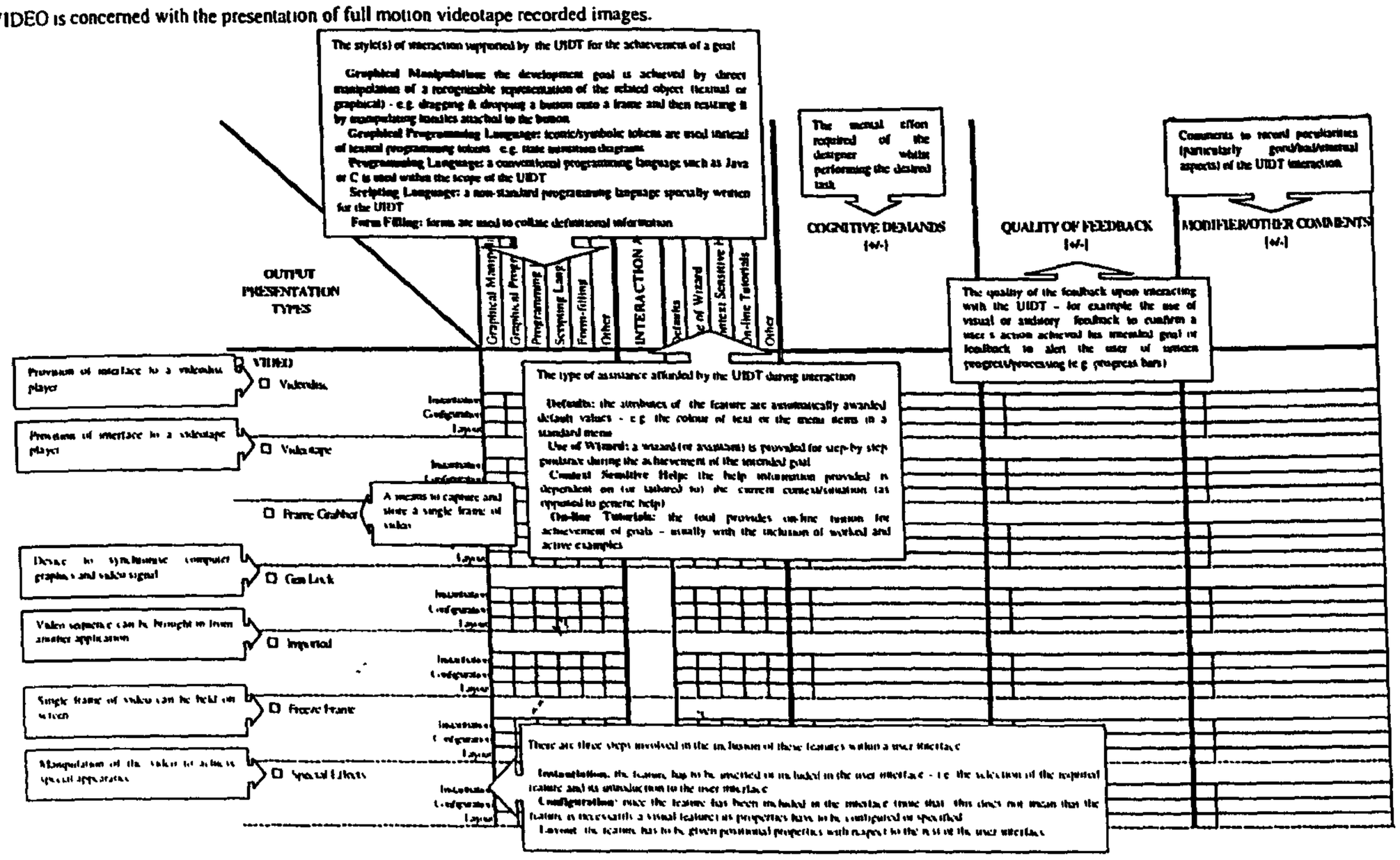


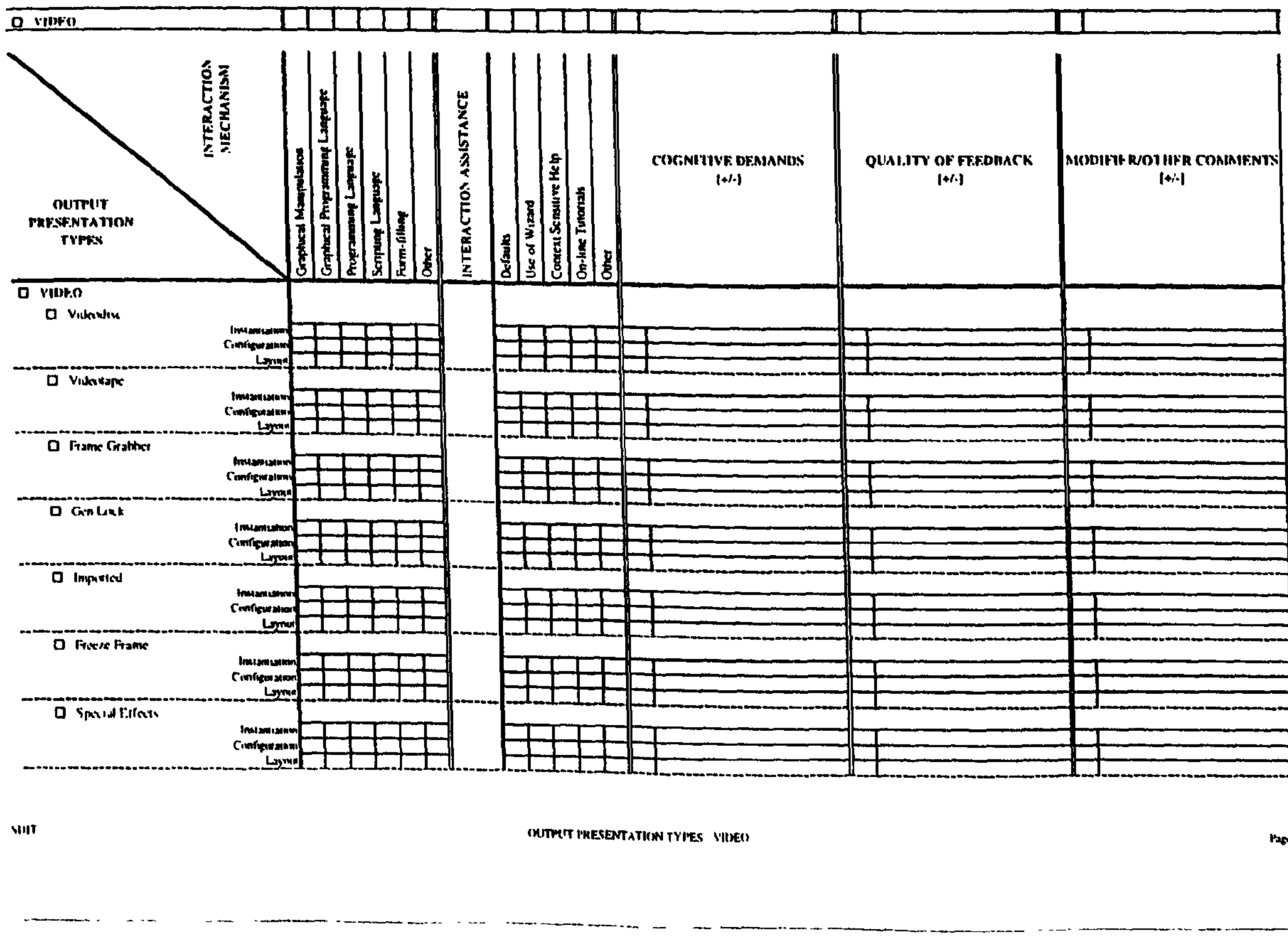
INTERACTION MECHANISM
OUTPUT PRESENTATION TYPES
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
AUDIO Contd.
AUDIO : Synthesized Speech Contd.
Natural Tone
Instantiation
Configuration
Layout
Robotic Tone
Pitch Control
Rate Control
AUDIO : Non-Speech Sounds
AUDIO : Auditory Icons
AUDIO : Other

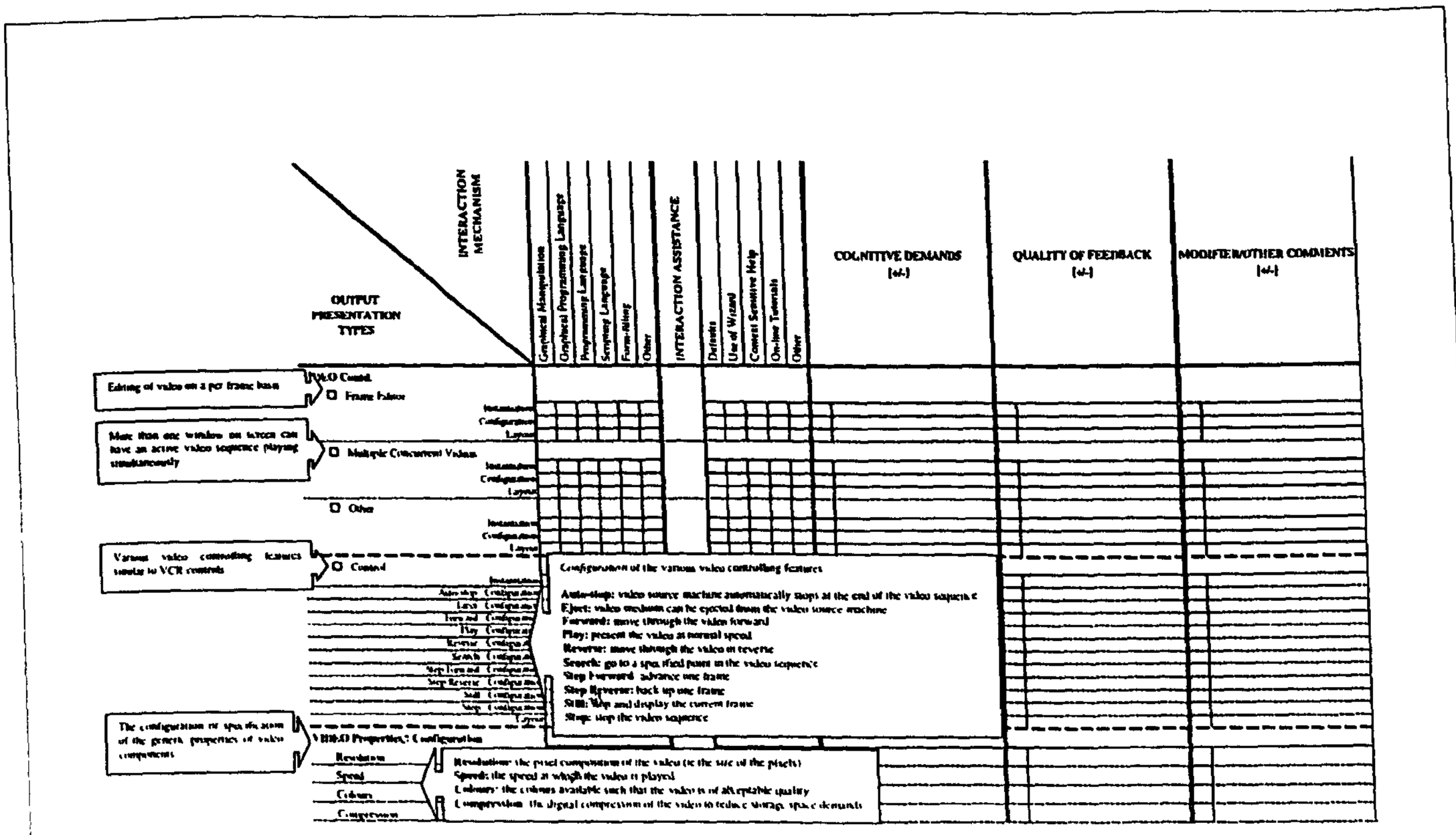

INTERACTION MECHANISM
OUTPUT PRESENTATION TYPES
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
AUDIO Genl.
AUDIO Properties : Configuration
The configuration or specification of the generic properties of audio components
Stereo: right and left audio channels
Mono: only one audio channel
Looping
Repetition
Volume
Record Capability
Waveform
Other
Repetition: a sound sequence can automatically be played a specified number of times
Volume: volume can be specified

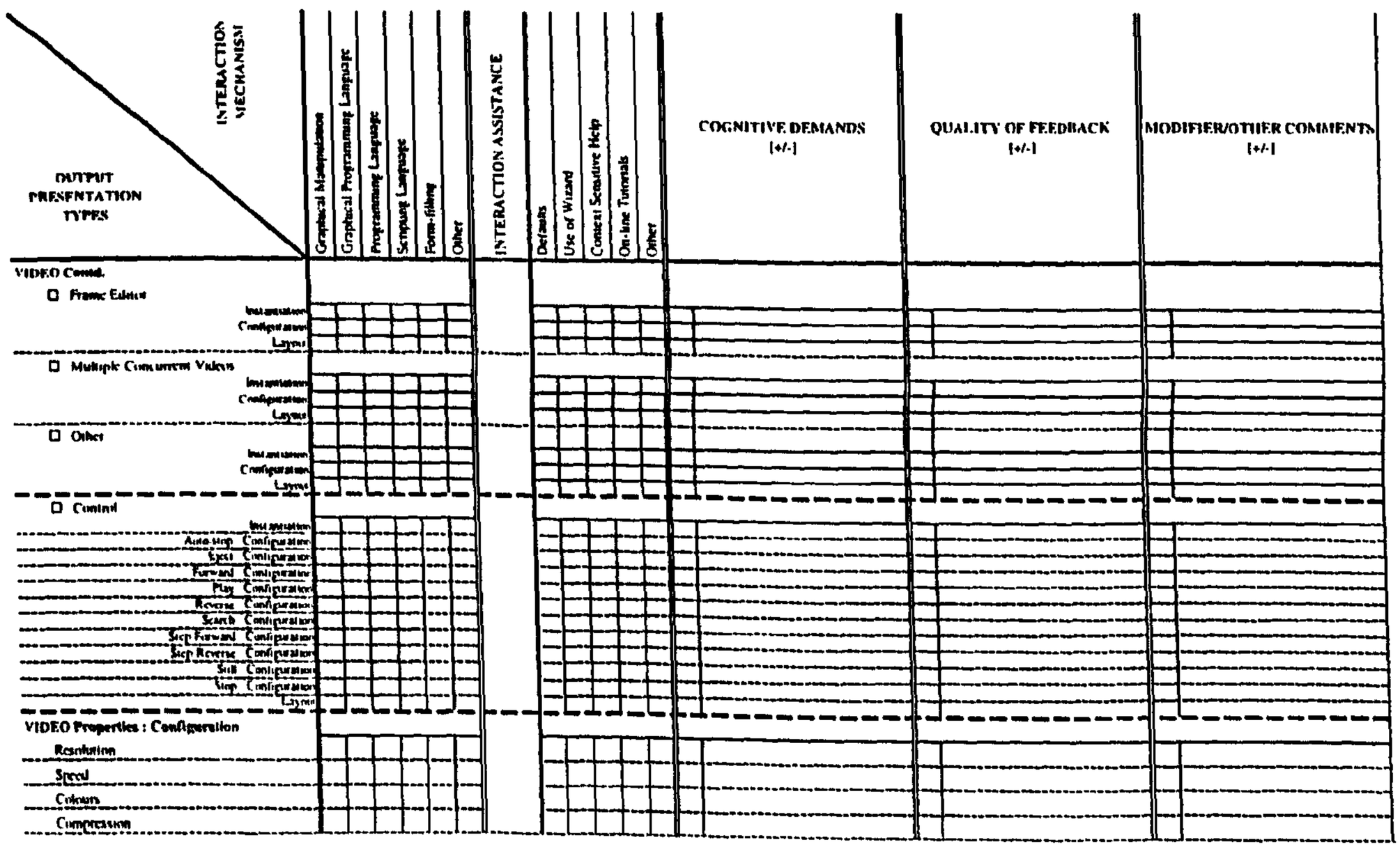

INTERACTION MECHANISM
OUTPUT PRESENTATION TYPES
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
AUDIO Genl.
AUDIO Properties : Configuration
Stereo
Mono
Looping
Repetition
Volume
Record Capability
Waveform
Other

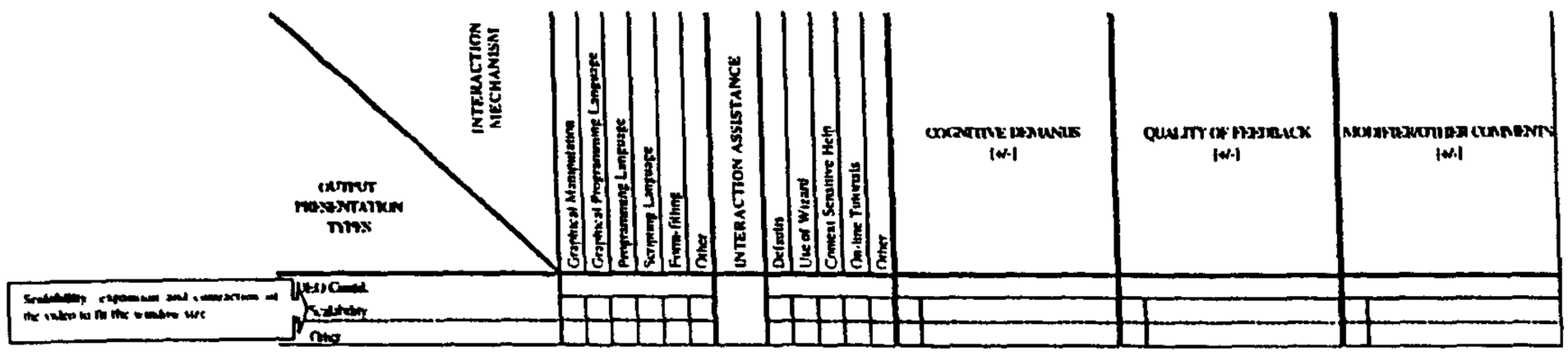
VIDEO is concerned with the presentation of full motion videotape recorded images.

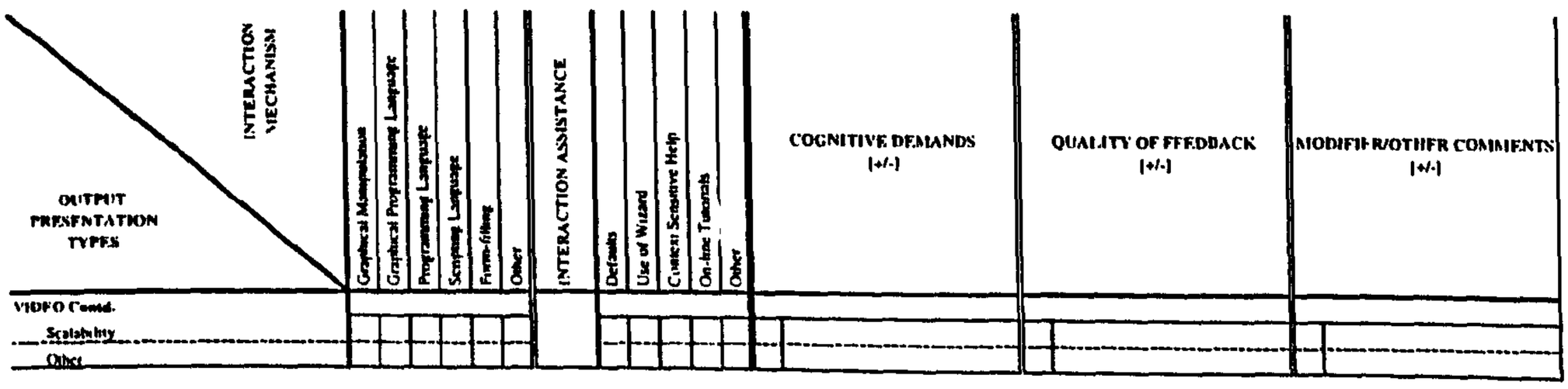


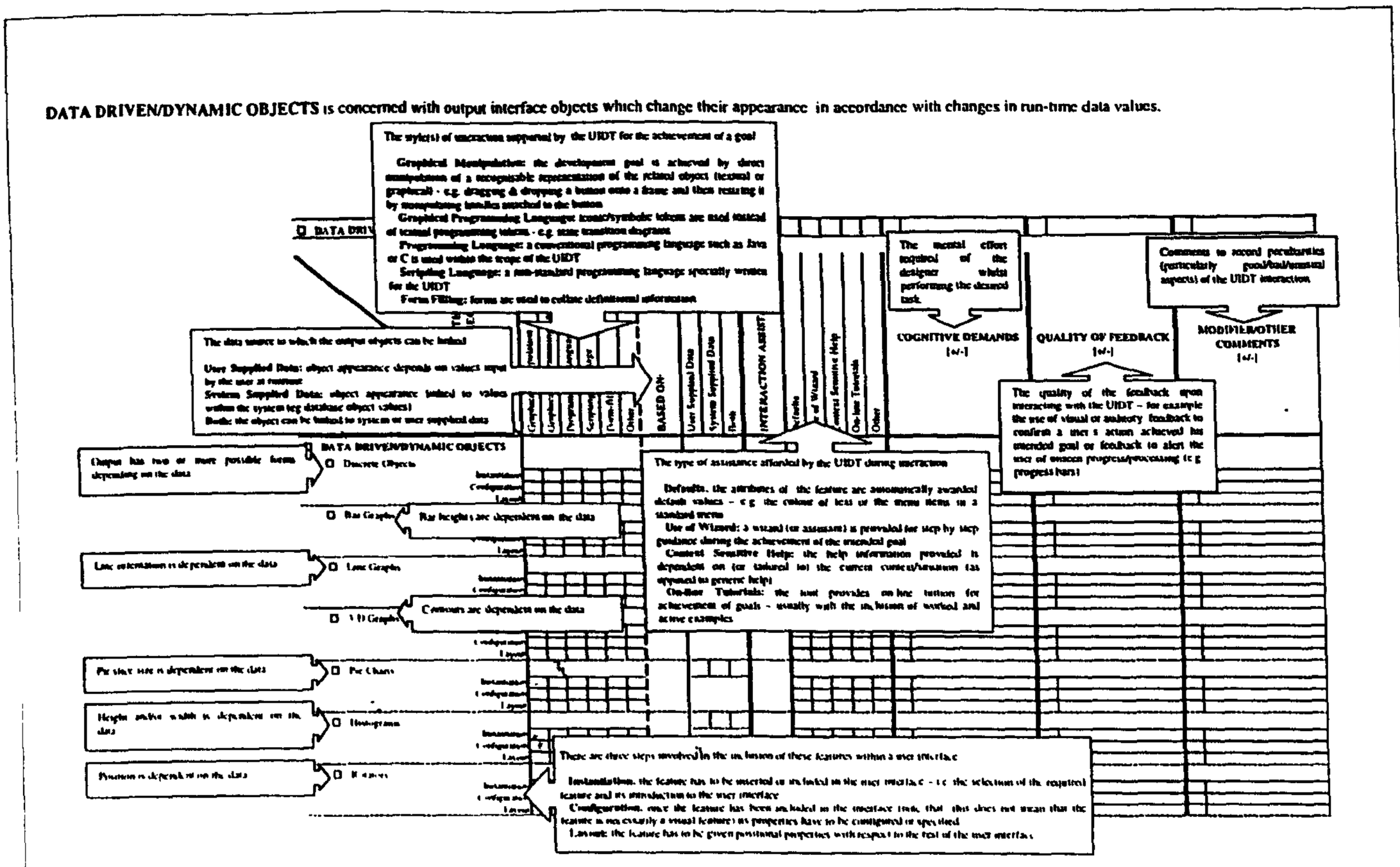


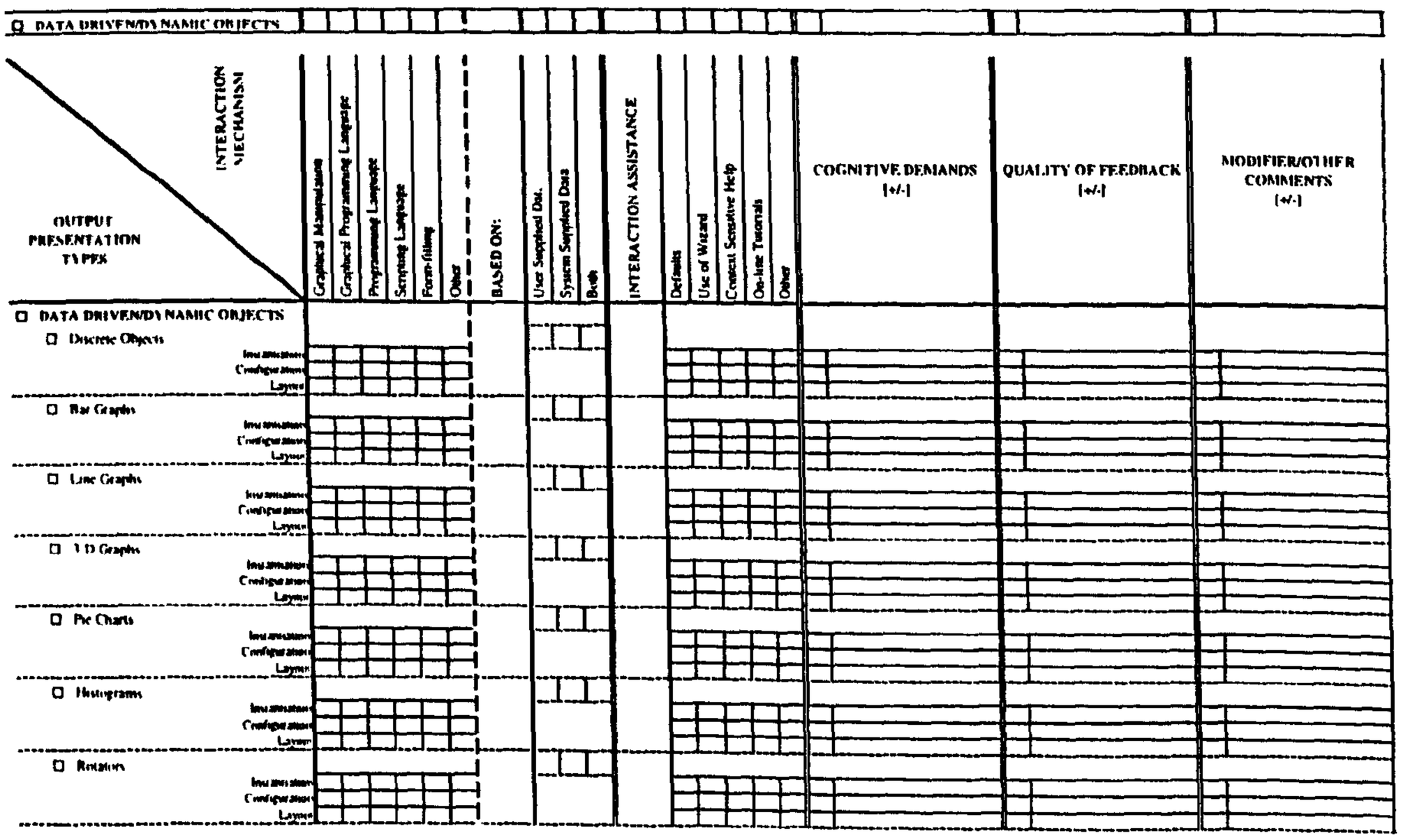

INTERACTION MECHANISM
OUTPUT PRESENTATION TYPES
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
Editing of video on a per frame basis
VIDEO Contd.
Frame Editor
Multiple Concurrent Videos
Other
Control
Various video controlling features similar to VCR controls
Configuration of the various video controlling features
Auto-stop: video source machine automatically stops at the end of the video sequence
Eject: video medium can be ejected from the video source machine
Forward: move through the video forward
Play: present the video at normal speed
Reverse: move through the video in reverse
Search: go to a specified point in the video sequence
Step Forward: advance one frame
Step Reverse: back up one frame
Still: Map and display the current frame
Stop: stop the video sequence
The configuration or specification of the generic properties of video components
VIDEO Properties : Configuration
Resolution
Speed
Colours
Compression
Resolution: the pixel composition of the video (ie the size of the pixels)
Speed: the speed at which the video is played
Colours: the colours available such that the video is of acceptable quality
Compression: the digital compression of the video to reduce storage space demands

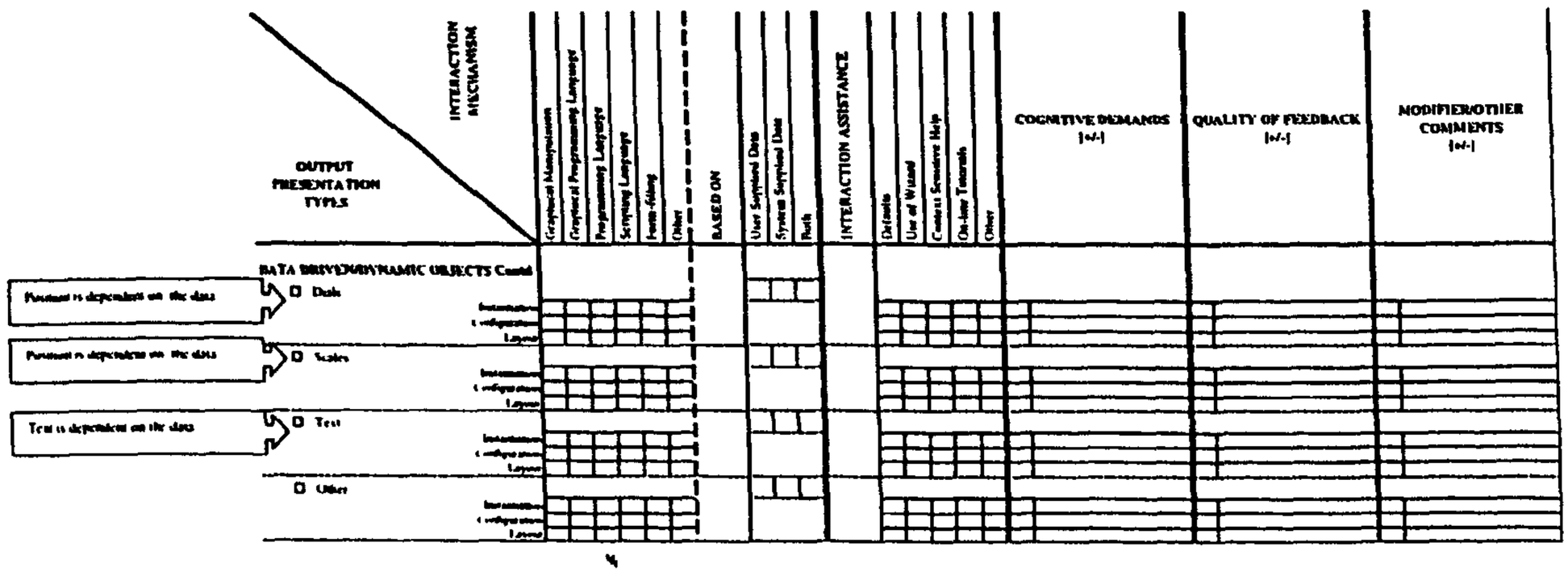

INTERACTION MECHANISM
OUTPUT PRESENTATION TYPES
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
VIDEO Contd.
Frame Editor
Multiple Concurrent Videos
Other
Control
Auto-stop Configuration
Eject Configuration
Forward Configuration
Play Configuration
Reverse Configuration
Search Configuration
Step Forward Configuration
Step Reverse Configuration
Still Configuration
Stop Configuration
Layout
VIDEO Properties : Configuration
Resolution
Speed
Colours
Compression

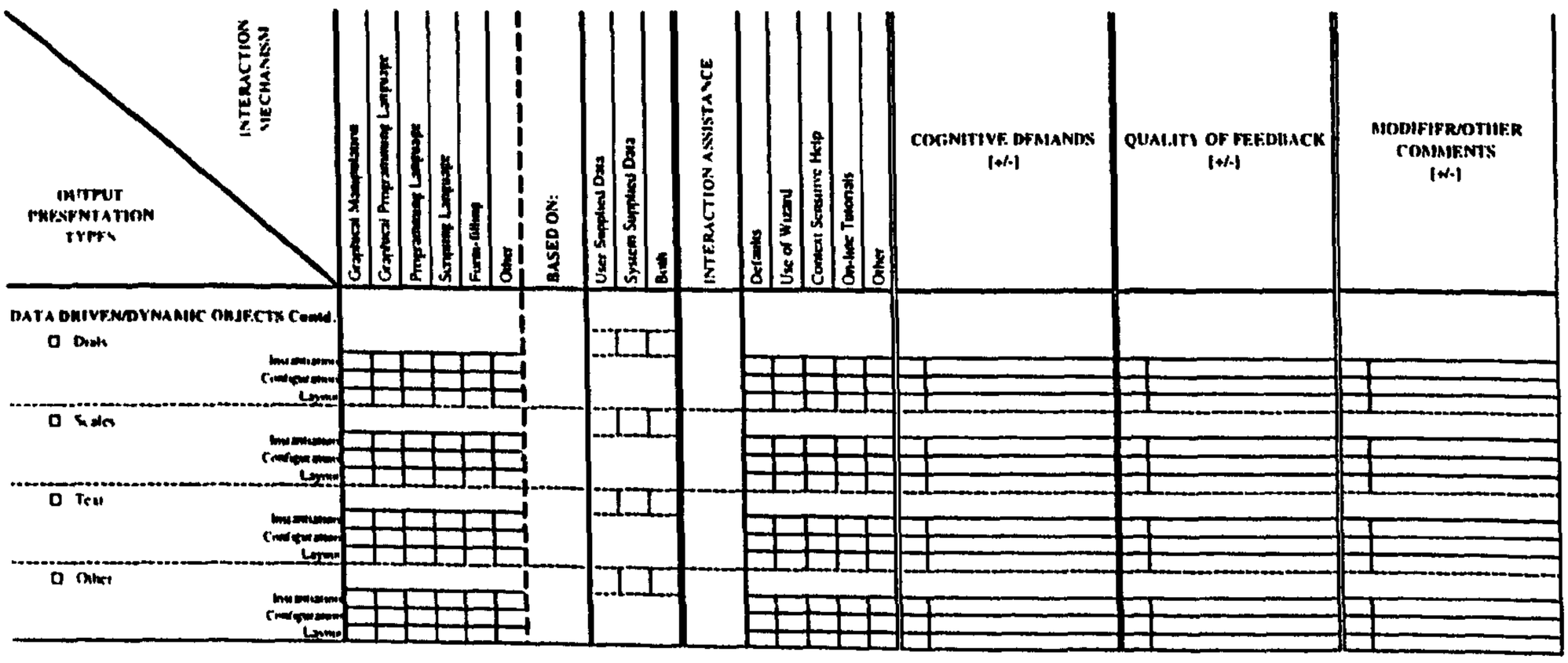

INTERACTION MECHANISM
OUTPUT PRESENTATION TYPES
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
Scalability: expansion and contraction of the video to fit the window size
VIDEO Cond.
Scalability
Other


INTERACTION MECHANISM
OUTPUT PRESENTATION TYPES
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
VIDEO Cond.
Scalability
Other

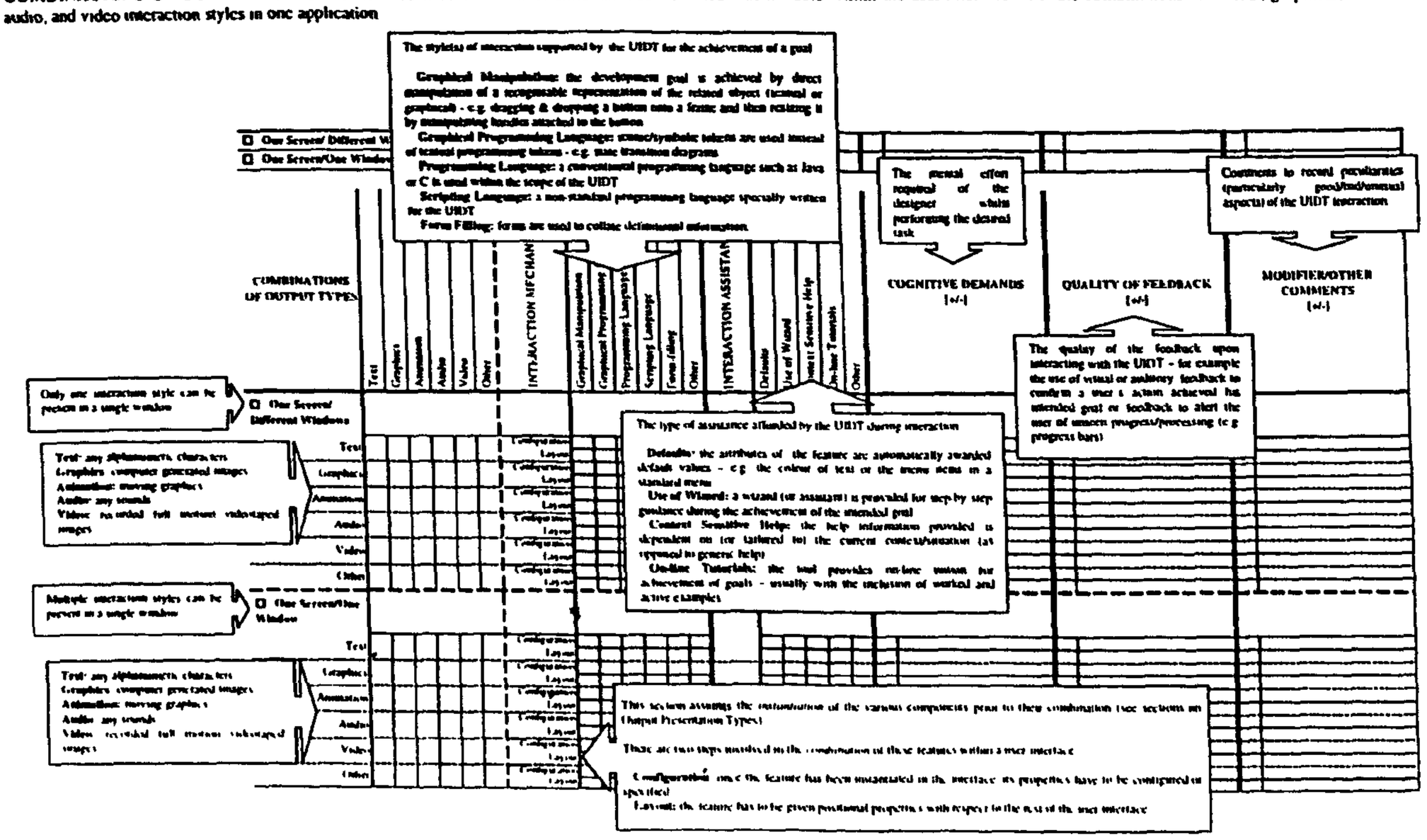

DATA DRIVEN/DYNAMIC OBJECTS is concerned with output interface objects which change their appearance in accordance with changes in run-time data values.
The style(s) of interaction supported by the UIDT for the achievement of a goal
Graphical Manipulation: the development goal is achieved by direct manipulation of a recognisable representation of the related object (textual or graphical) - e.g. dragging & dropping a button onto a frame and then resizing it by manipulating handles attached to the button
Graphical Programming Language: iconic/symbolic tokens are used instead of textual programming tokens - e.g. state transition diagrams
Programming Language: a conventional programming language such as Java or C is used within the scope of the UIDT
Scripting Language: a non-standard programming language specially written for the UIDT
Form Filling: forms are used to collate definitional information
The data source to which the output objects can be linked
User Supplied Data: object appearance depends on values input by the user at runtime
System Supplied Data: object appearance linked to values within the system (eg database object values)
Both: the object can be linked to system or user supplied data
The mental effort required of the designer whilst performing the desired task
COGNITIVE DEMANDS [+/-]
Comments to record peculiarities (particularly good/bad/unusual aspects) of the UIDT interaction
MODIFIER/OTHER COMMENTS [+/-]
QUALITY OF FEEDBACK [+/-]
The quality of the feedback upon interacting with the UIDT – for example the use of visual or auditory feedback to confirm a user's action achieved his intended goal or feedback to alert the user of system progress/processing (e.g. progress bars)
BASED ON:
INTERACTION ASSIST
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
User Supplied Data
System Supplied Data
Both
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
DATA DRIVEN/DYNAMIC OBJECTS
Discrete Objects
Bar Graphs
Line Graphs
3-D Graphs
Pie Charts
Histograms
Rotators
Instantiation
Configuration
Layout
Output has two or more possible forms depending on the data
Bar heights are dependent on the data
Line orientation is dependent on the data
Contours are dependent on the data
Pie slice size is dependent on the data
Height and/or width is dependent on the data
Position is dependent on the data
The type of assistance afforded by the UIDT during interaction
Defaults: the attributes of the feature are automatically awarded default values - e.g. the colour of text or the menu items in a standard menu
Use of Wizard: a wizard (or assistant) is provided for step by step guidance during the achievement of the intended goal
Context Sensitive Help: the help information provided is dependent on (or tailored to) the current context/situation (as opposed to generic help)
On-line Tutorials: the tool provides on-line tuition for achievement of goals - usually with the inclusion of worked and active examples
There are three steps involved in the inclusion of these features within a user interface
Instantiation: the feature has to be selected as included in the user interface - i.e. the selection of the required feature and its introduction to the user interface
Configuration: once the feature has been included in the interface tools, that does not mean that the feature is necessarily a visual feature as properties have to be configured or specified
Layout: the feature has to be given positional properties with respect to the rest of the user interface
SUIT
OUTPUT PRESENTATION TYPES DATA DRIVEN/DYNAMIC OBJECTS
Page G20

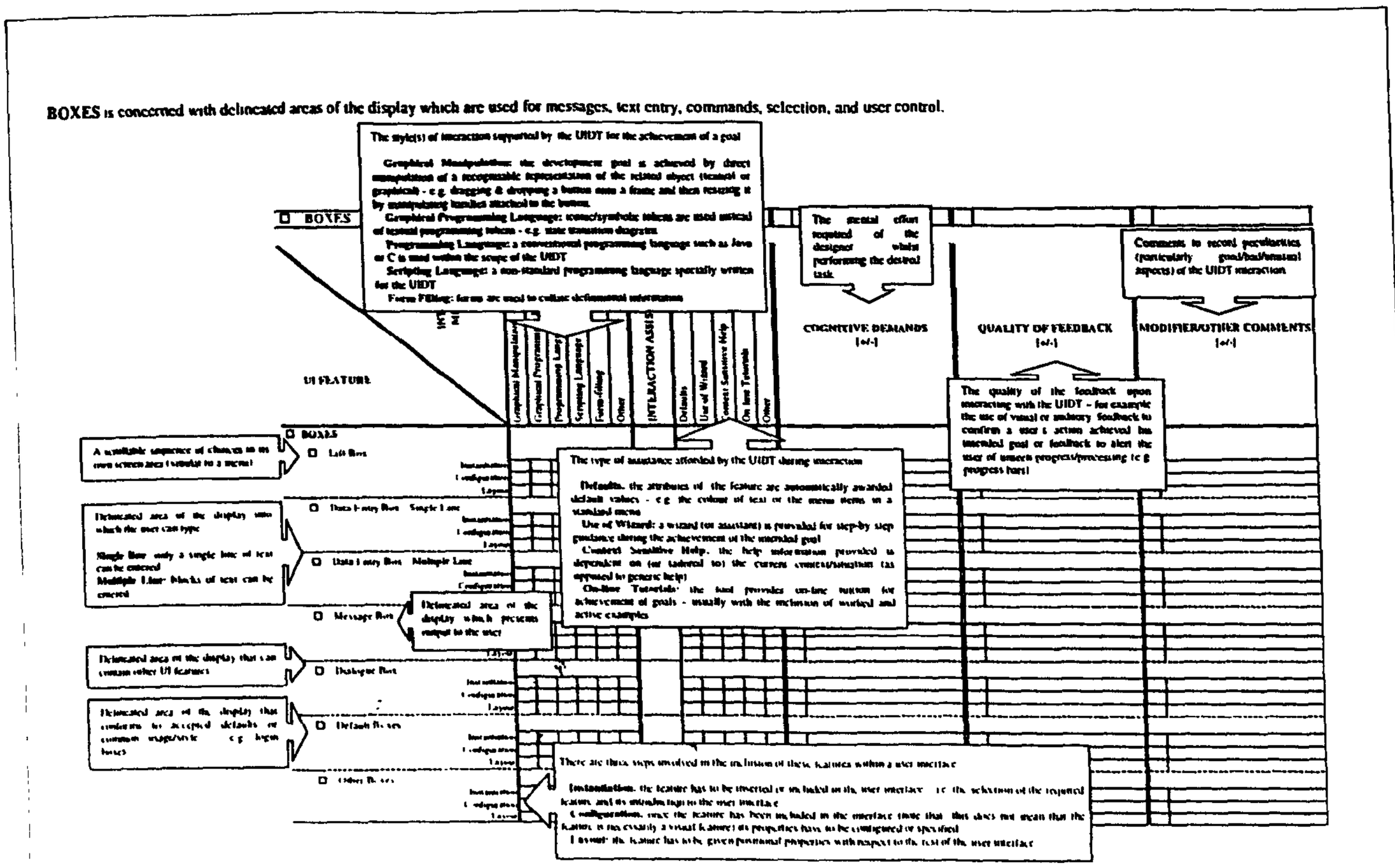

DATA DRIVEN/DYNAMIC OBJECTS
OUTPUT PRESENTATION TYPES
INTERACTION MECHANISM
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
BASED ON:
User Supplied Dat.
System Supplied Data
Both
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
DATA DRIVEN/DYNAMIC OBJECTS
Discrete Objects
Bar Graphs
Line Graphs
3-D Graphs
Pie Charts
Histograms
Rotators
Instantiation
Configuration
Layout
SUIT
OUTPUT PRESENTATION TYPES DATA DRIVEN/DYNAMIC OBJECTS
Page 20


INTERACTION MECHANISM
OUTPUT PRESENTATION TYPES
Graphical Manipulators
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
BASED ON
User Supplied Data
System Supplied Data
Push
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
DATA DRIVEN/DYNAMIC OBJECTS Contd.
Data
Scales
Text
Other
Instructions
Configuration
Layout

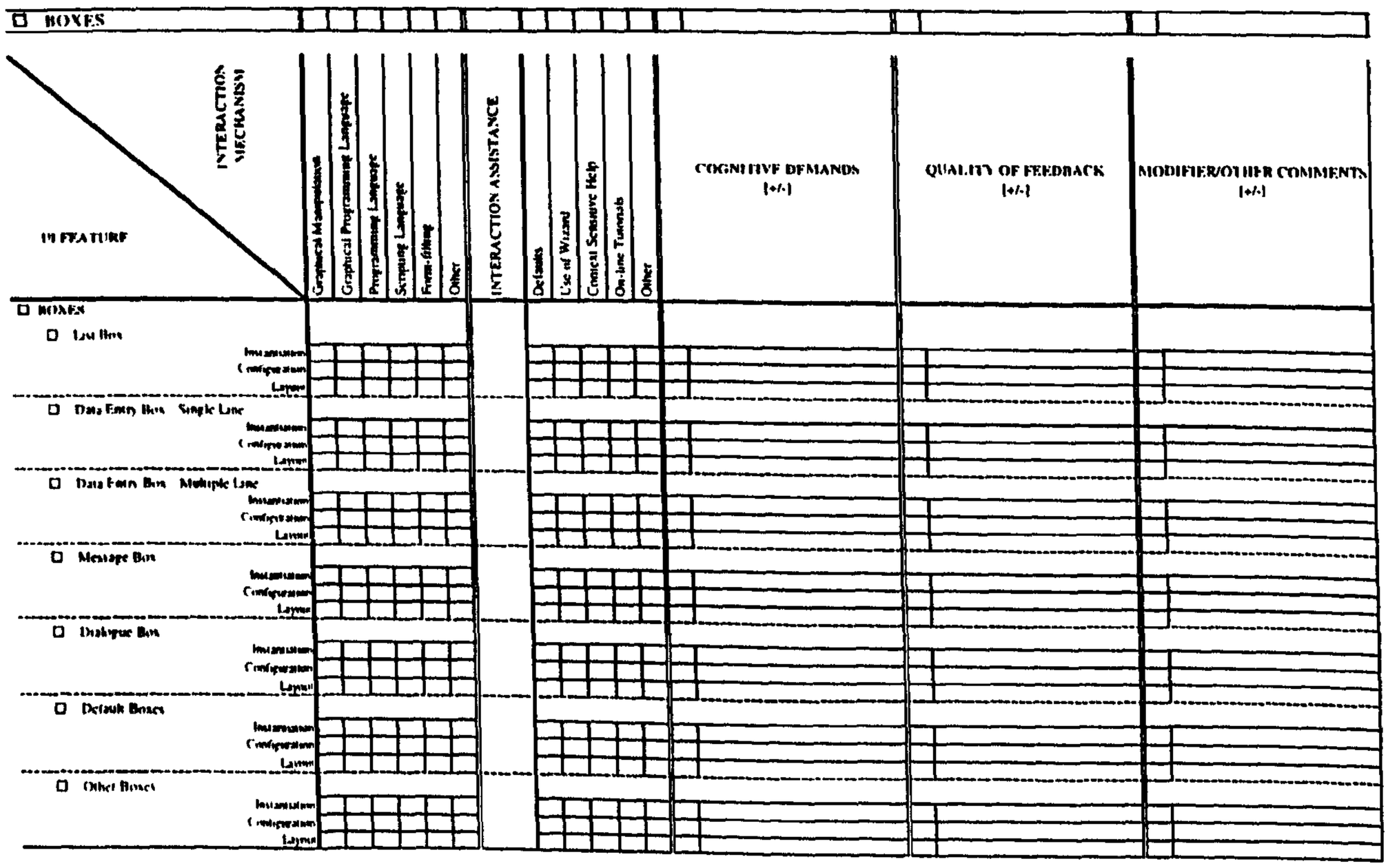

INTERACTION MECHANISM
OUTPUT PRESENTATION TYPES
Graphical Manipulators
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
BASED ON:
User Supplied Data
System Supplied Data
Both
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
DATA DRIVEN/DYNAMIC OBJECTS Contd.
Data
Scales
Text
Other
Instructions
Configuration
Layout

COMBINATIONS OF OUTPUT TYPES is concerned with the levels of simultaneous multimedia achievable within the user interface - i.e the combinations of textual, graphical, animation, audio, and video interaction styles in one application



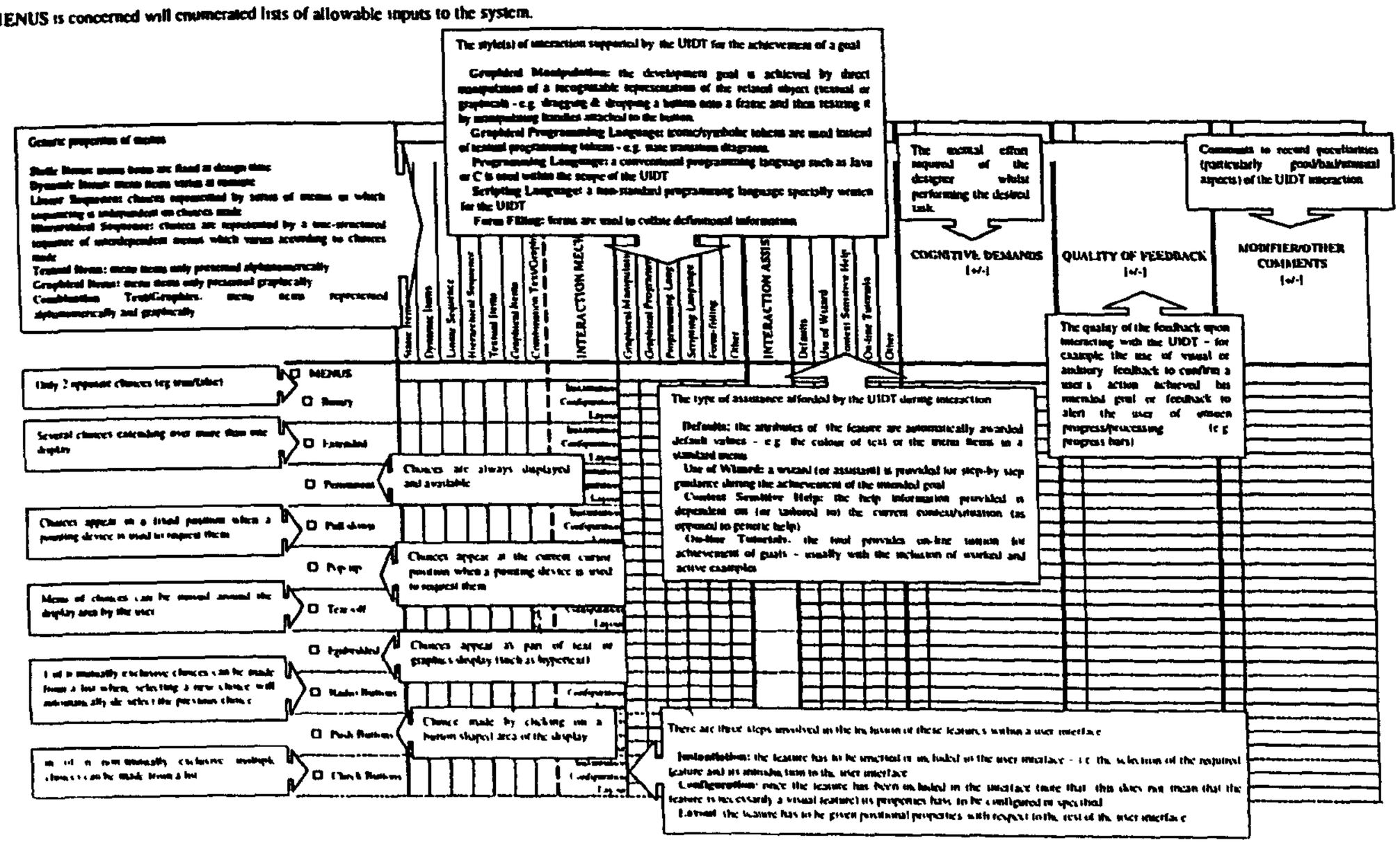


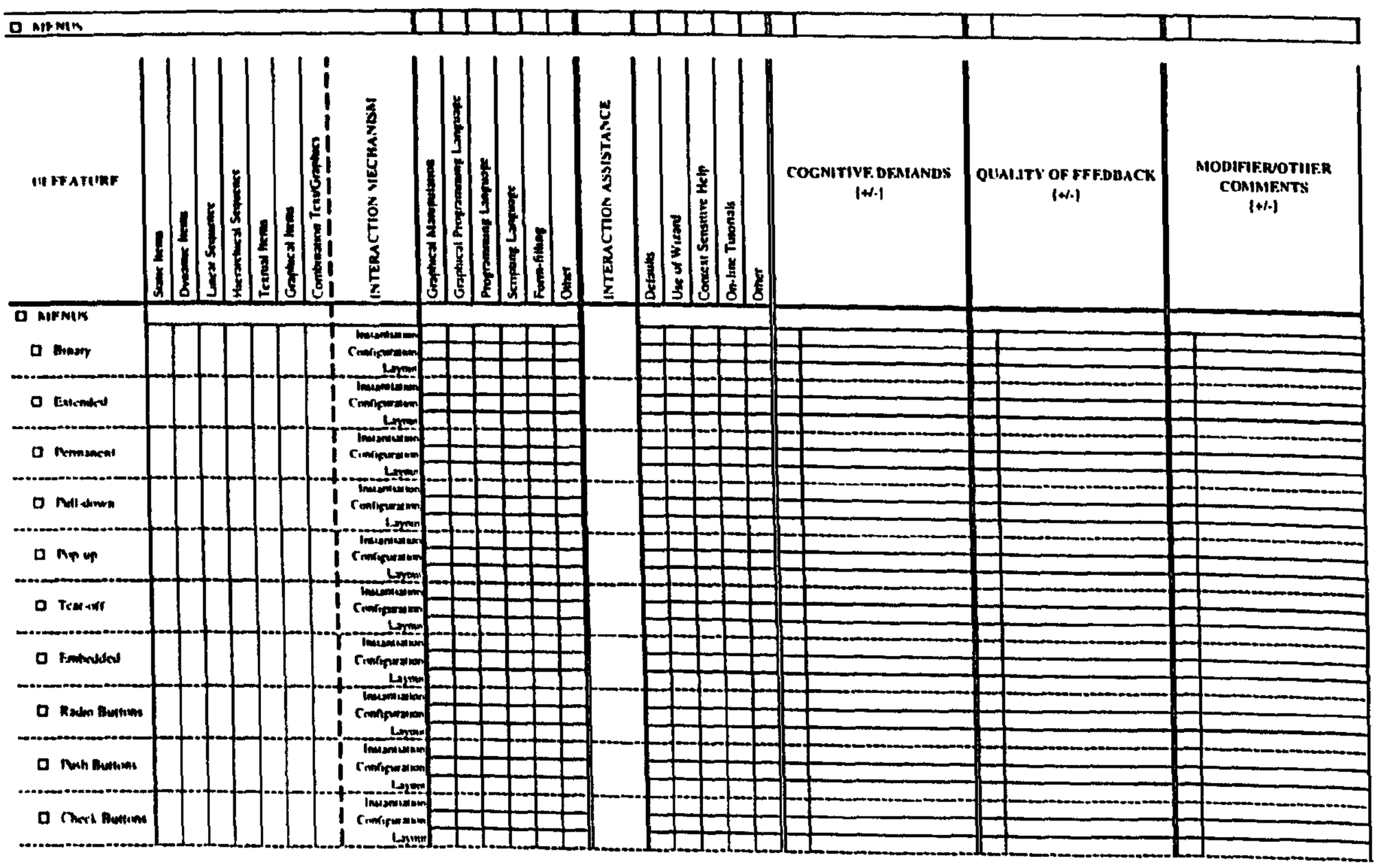
SUIT UI FEATURE: BOXES Page 63

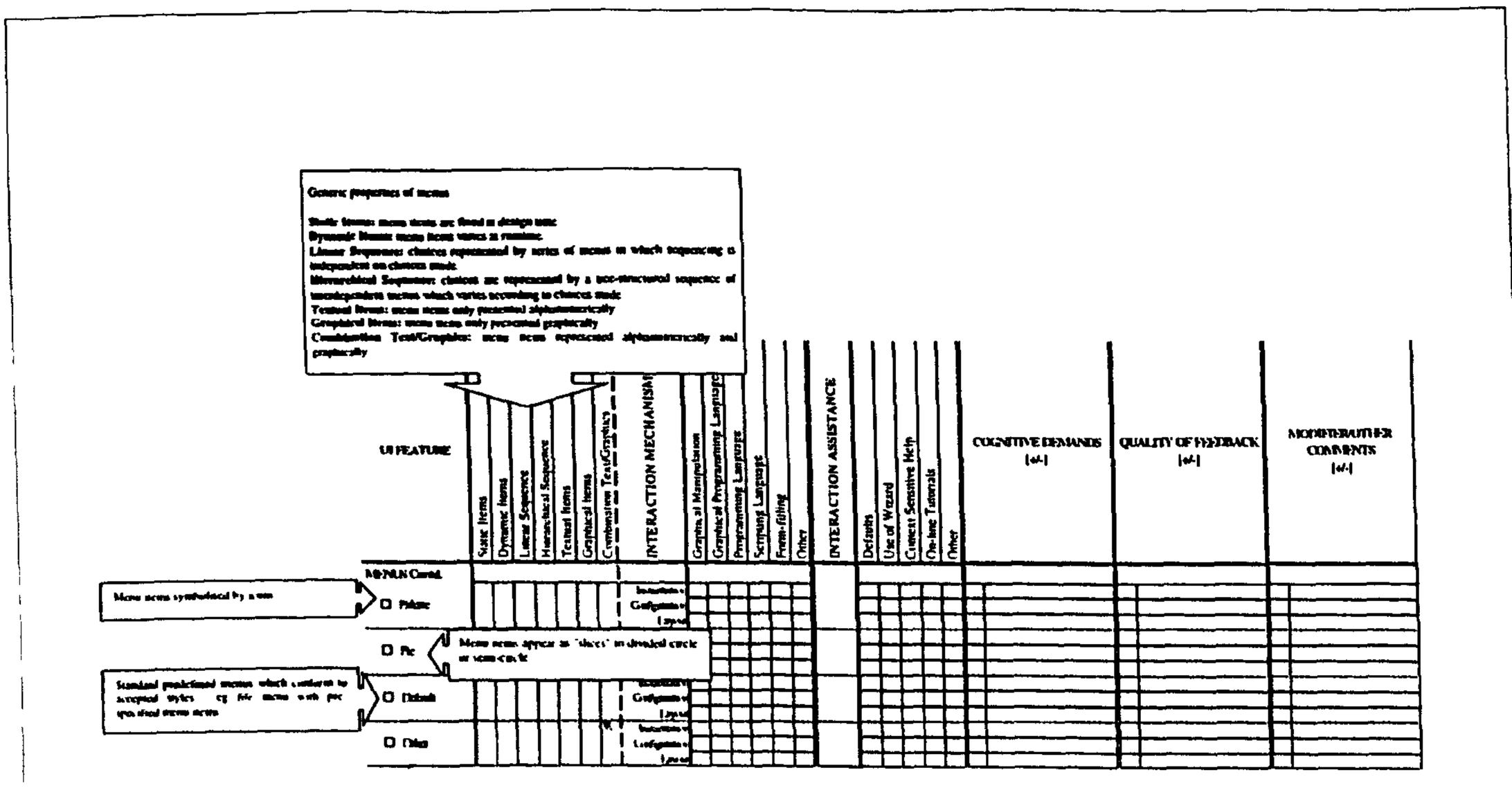


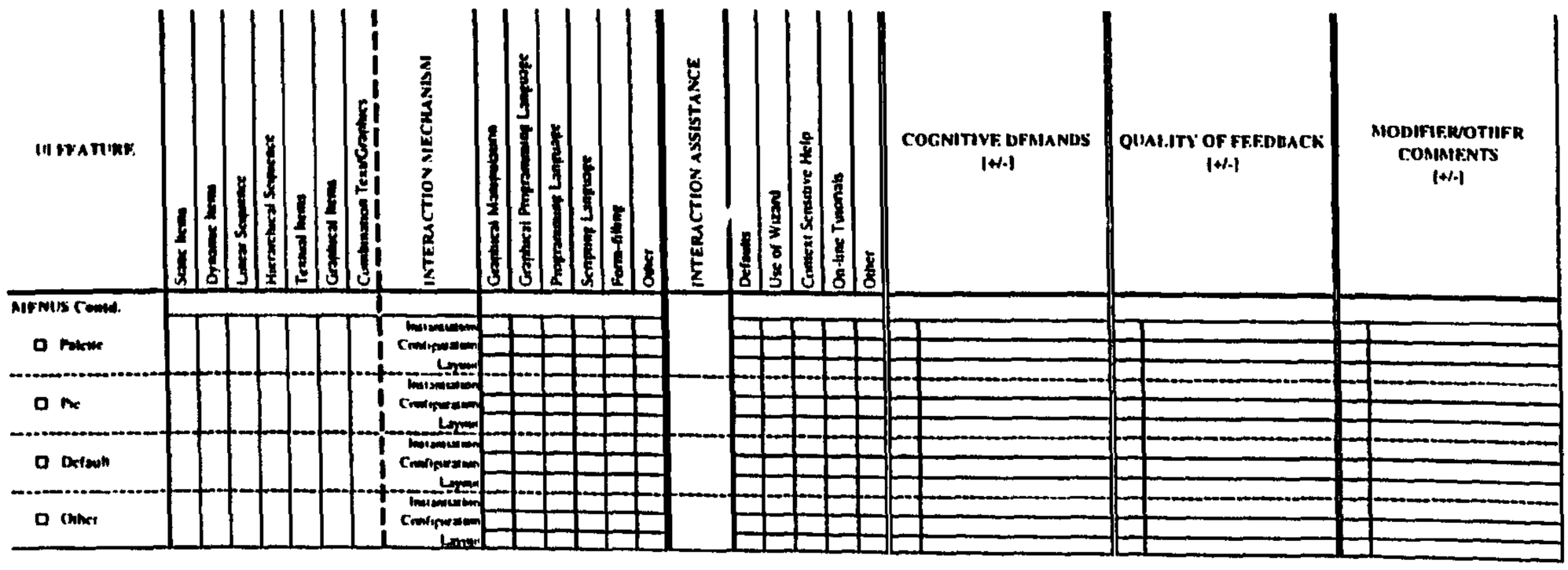
SUIT UI FEATURE: BOXES Page 64

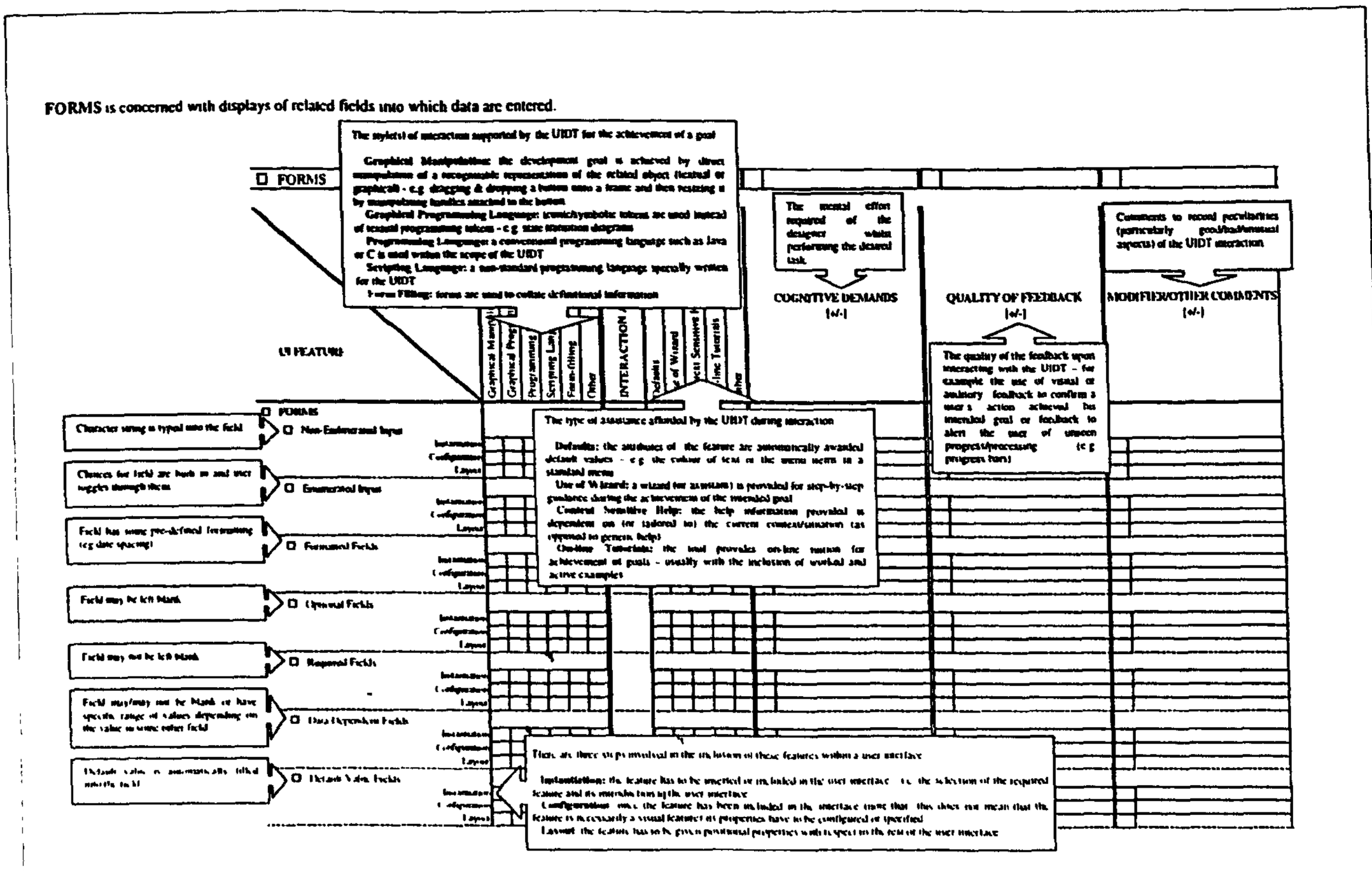
MENUS is concerned will enumerated lists of allowable inputs to the system.

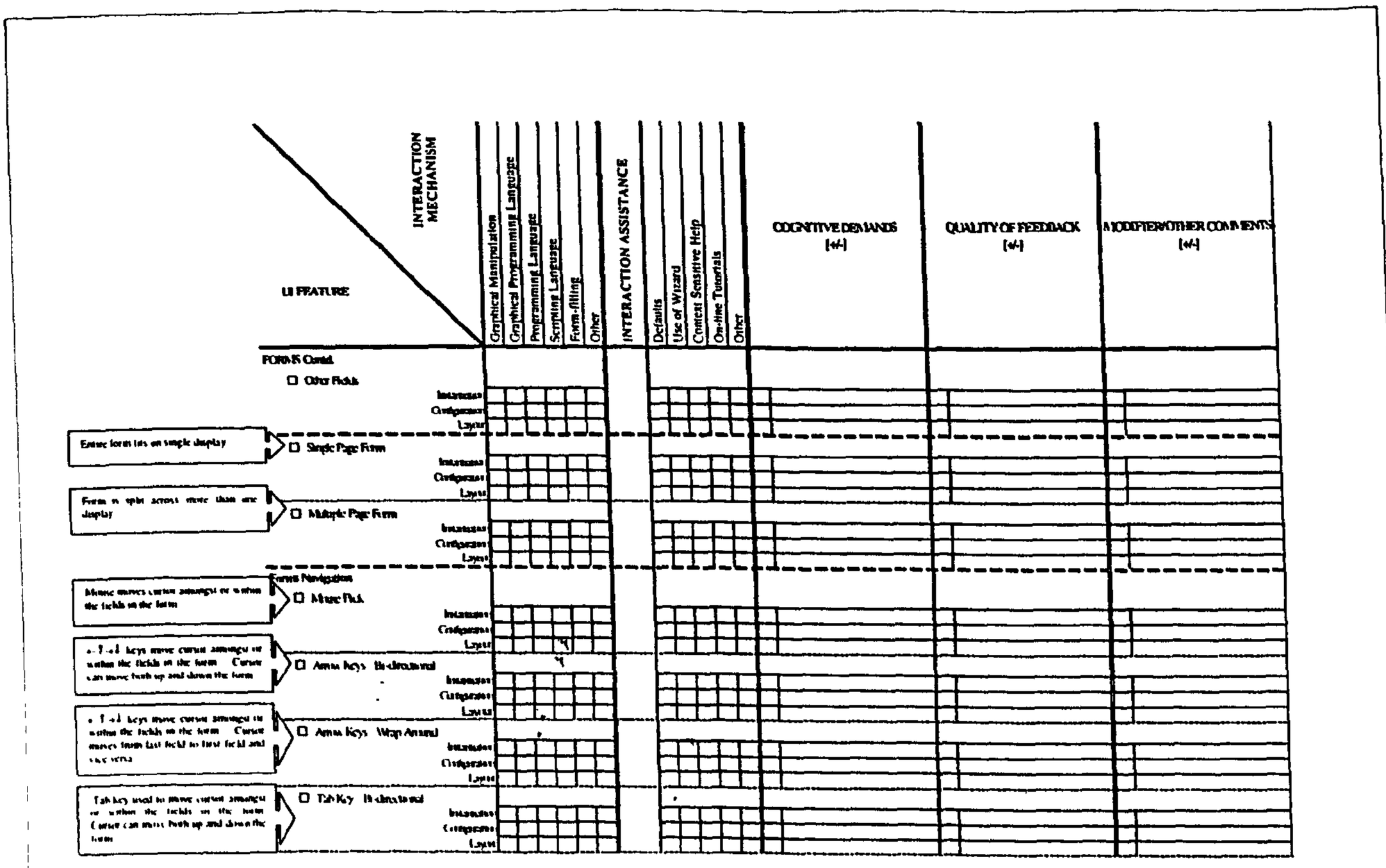


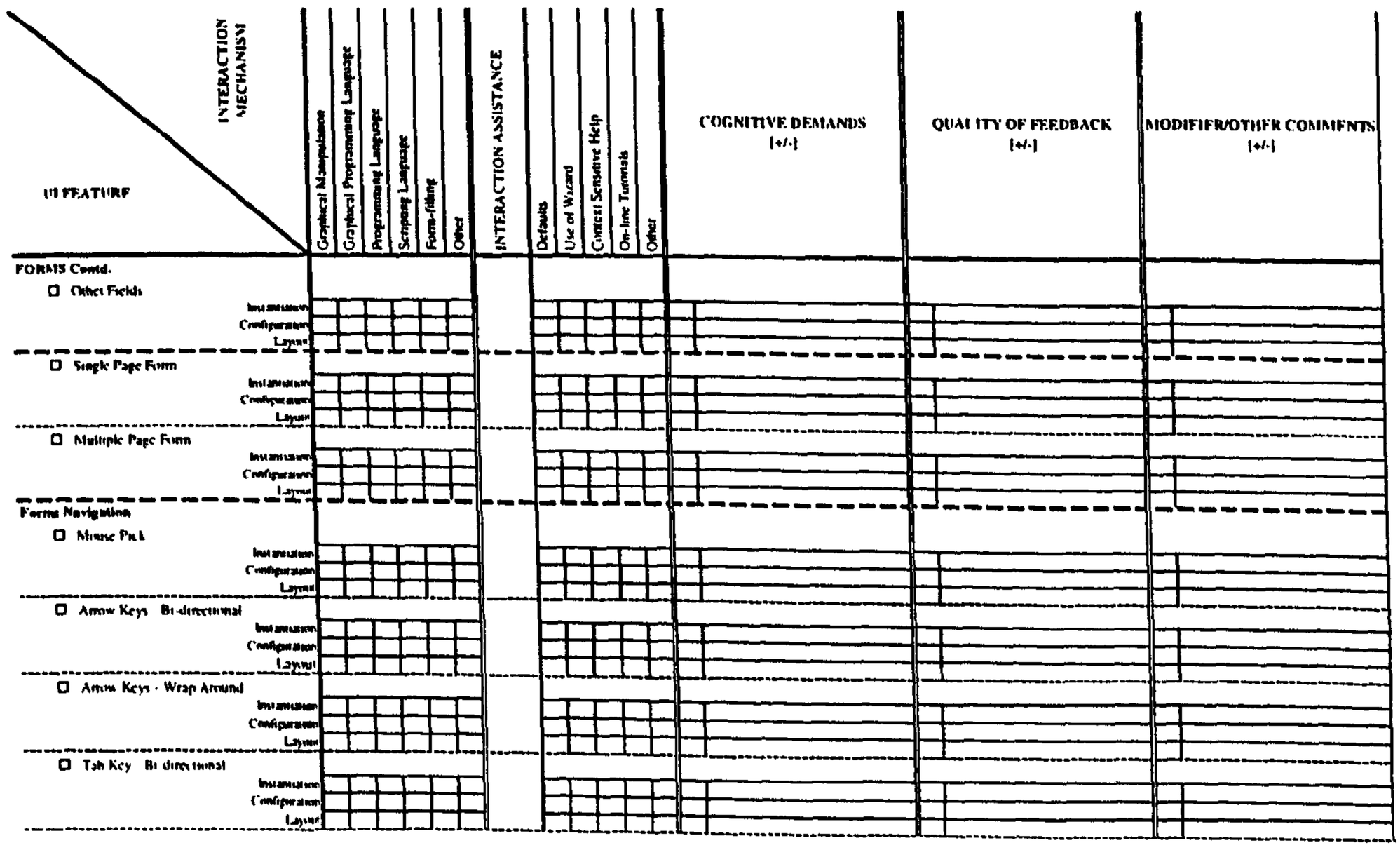


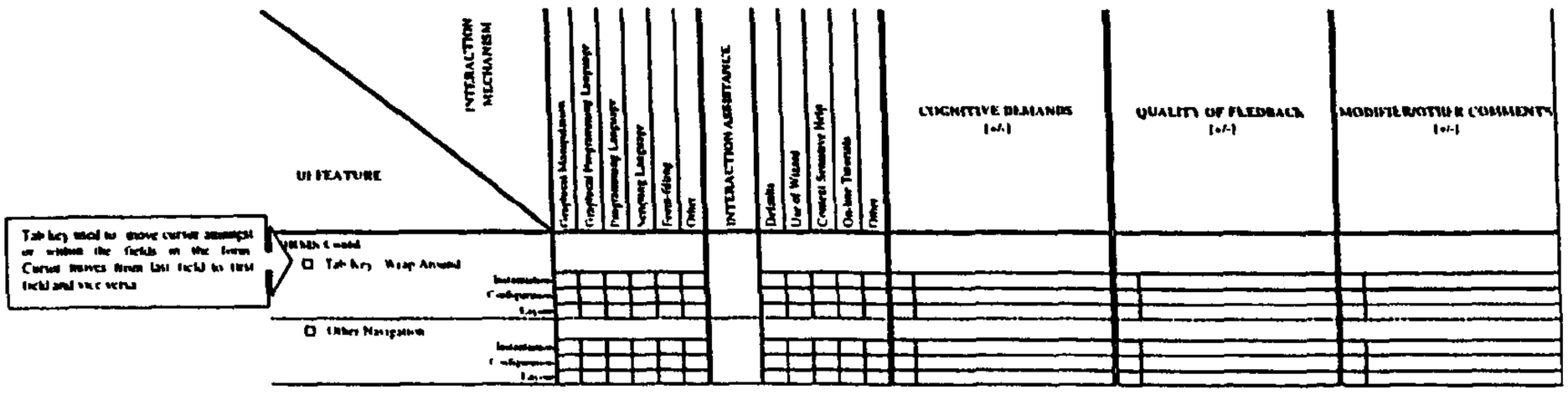

Generic properties of menus
Static Items: menu items are fixed at design time
Dynamic Items: menu items varies at runtime
Linear Sequence: choices represented by series of menus in which sequencing is independent on choices made
Hierarchical Sequence: choices are represented by a tree-structured sequence of interdependent menus which varies according to choices made
Textual Items: menu items only presented alphanumerically
Graphical Items: menu items only presented graphically
Combination Text/Graphics: menu items represented alphanumerically and graphically
UI FEATURE
INTERACTION MECHANISM
Static Items
Dynamic Items
Linear Sequence
Hierarchical Sequence
Textual Items
Graphical Items
Combination Text/Graphics
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS (+/-)
QUALITY OF FEEDBACK (+/-)
MODIFIER/OTHER COMMENTS (+/-)
MENUS Contd.
Palette
Instructions
Configuration
Layout
Pie
Default
Other
Menu items symbolised by icons
Menu items appear as "slices" in divided circle or semi-circle
Standard predefined menus which conform to accepted styles eg file menus with predefined menu items

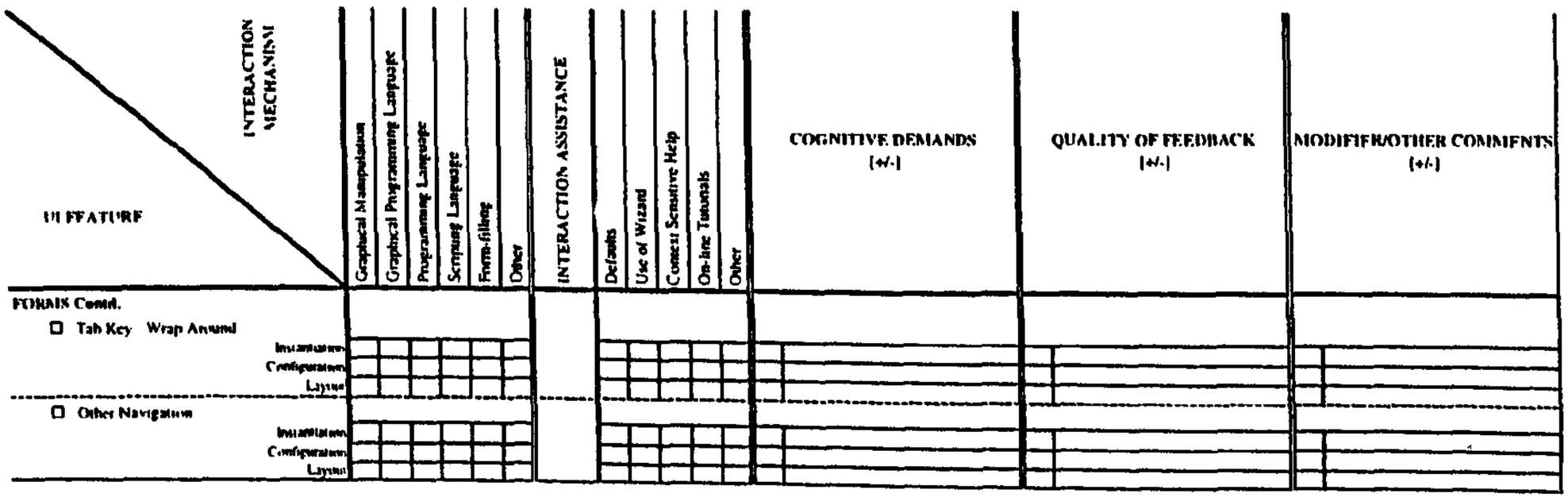

UI FEATURE
INTERACTION MECHANISM
INTERACTION ASSISTANCE
COGNITIVE DEMANDS (+/-)
QUALITY OF FEEDBACK (+/-)
MODIFIER/OTHER COMMENTS (+/-)
MENUS Contd.
Palette
Instructions
Configuration
Layout
Pie
Default
Other


FORMS is concerned with displays of related fields into which data are entered.
FORMS
UI FEATURE
INTERACTION MECHANISM
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
The mental effort required of the designer whilst performing the desired task
Comments to record peculiarities (particularly good/bad/unusual aspects) of the UIDT interaction
The quality of the feedback upon interacting with the UIDT – for example the use of visual or auditory feedback to confirm a user's action achieved his intended goal or feedback to alert the user of unseen progress/processing (e.g progress bars)
The style of interaction supported by the UIDT for the achievement of a goal
Graphical Manipulation: the development goal is achieved by direct manipulation of a recognisable representation of the related object (textual or graphical) - e.g dragging & dropping a button onto a frame and then resizing it by manipulating handles attached to the button
Graphical Programming Language: iconic/symbolic tokens are used instead of textual programming tokens - e.g state transition diagrams
Programming Language: a conventional programming language such as Java or C is used within the scope of the UIDT
Scripting Language: a non-standard programming language specially written for the UIDT
Form Filling: forms are used to collate definitional information
The type of assistance afforded by the UIDT during interaction
Defaults: the attributes of the feature are automatically awarded default values - e.g the colour of text or the menu items in a standard menus
Use of Wizard: a wizard (or assistant) is provided for step-by-step guidance during the achievement of the intended goal
Context Sensitive Help: the help information provided is dependent on (or tailored to) the current context/situation (as opposed to generic help)
On-line Tutorials: the tool provides on-line tuition for achievement of goals - usually with the inclusion of worked and active examples
There are three steps involved in the inclusion of these features within a user interface
Instantiation: the feature has to be inserted or included in the user interface i.e. the selection of the required feature and its insertion into the user interface
Configuration: once the feature has been included in the interface (note that this does not mean that the feature is necessarily a visual feature) its properties have to be configured or specified
Layout: the feature has to be given positional properties with respect to the rest of the user interface
FORMS
Character string or typed into the field
Non-Enumerated Input
Choices for field are built in and user toggles through them
Enumerated Input
Field has some pre-defined formatting (e.g date spacing)
Formatted Fields
Field may be left blank
Optional Fields
Field may not be left blank
Required Fields
Field may/may not be blank or have specific range of values depending on the value in some other field
Data Dependent Fields
Default value is automatically filled when field
Default Value Fields
Instantiation
Configuration
Layout
SUIT
UI FEATURE FORMS
Page 36

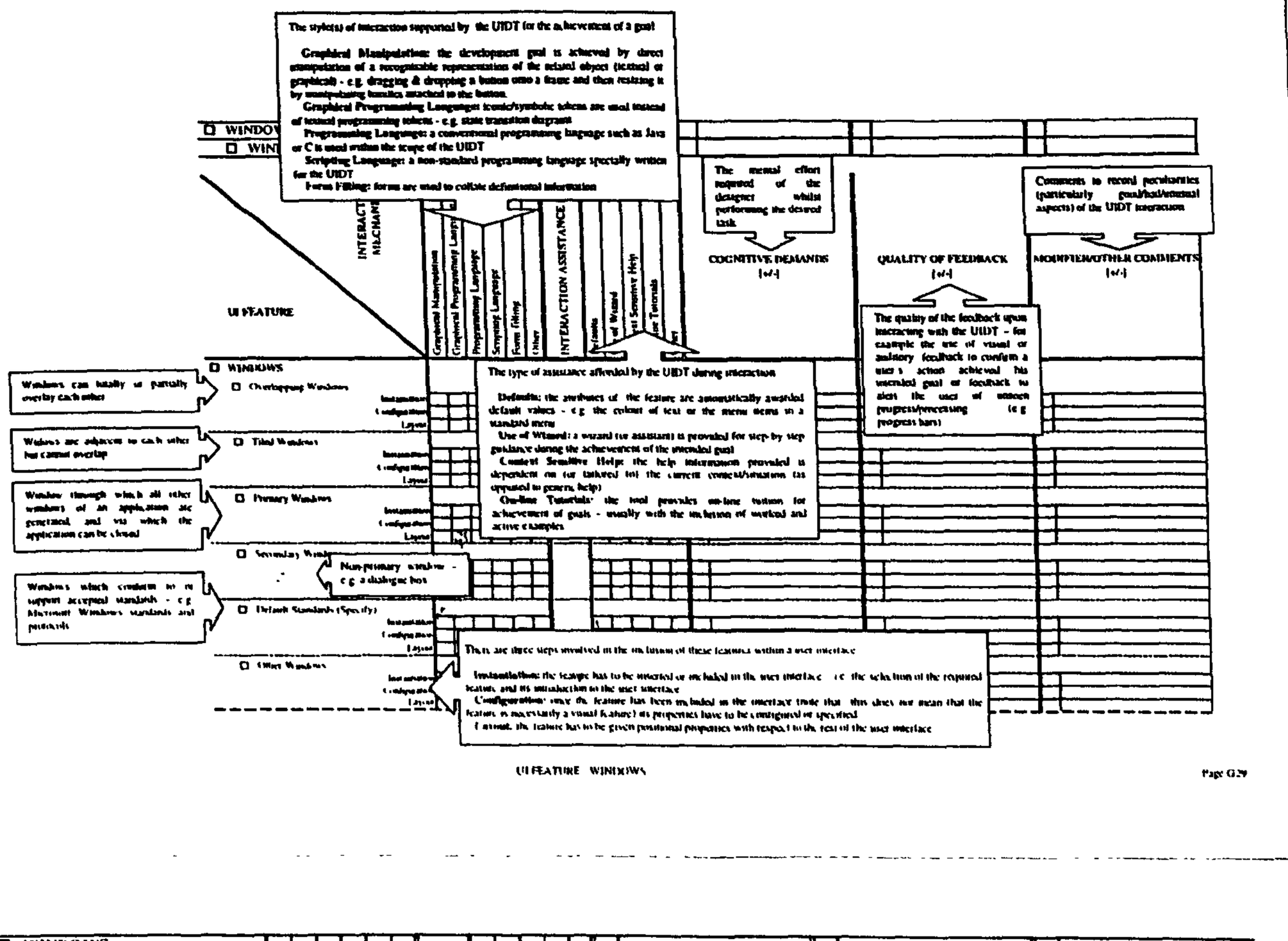

UI FEATURE
INTERACTION MECHANISM
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
FORMS Contd.
Other Fields
Instruction
Configuration
Layout
Entire form fits on single display
Single Page Form
Instruction
Configuration
Layout
Form is split across more than one display
Multiple Page Form
Instruction
Configuration
Layout
Forms Navigation
Mouse moves cursor amongst or within the fields on the form
Mouse Pick
Instruction
Configuration
Layout
Arrow keys move cursor amongst or within the fields on the form. Cursor can move both up and down the form
Arrow Keys Bi-directional
Instruction
Configuration
Layout
Arrow keys move cursor amongst or within the fields on the form. Cursor moves from last field to first field and vice versa
Arrow Keys Wrap Around
Instruction
Configuration
Layout
Tab key used to move cursor amongst or within the fields on the form. Cursor can move both up and down the form
Tab Key Bi-directional
Instruction
Configuration
Layout

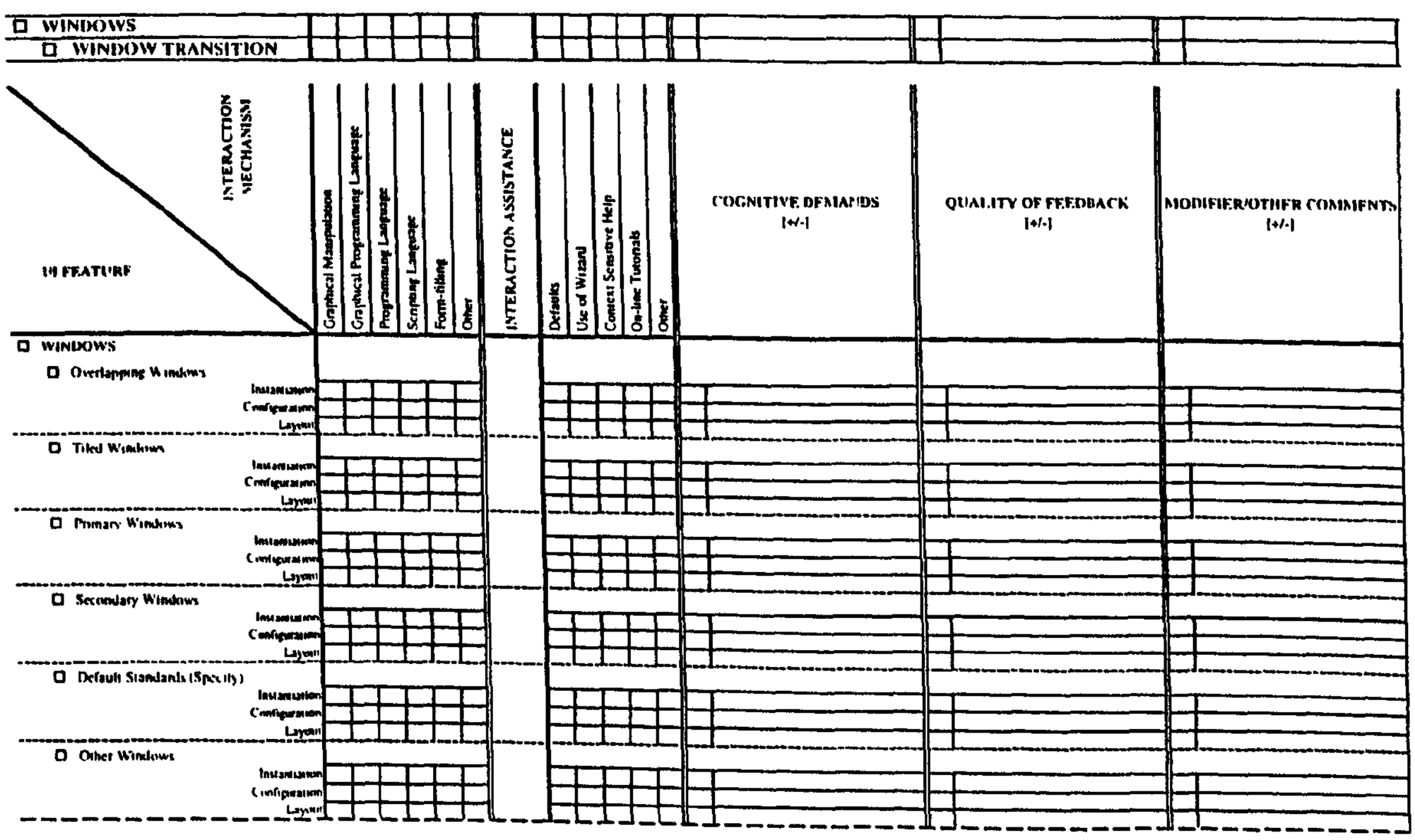

UI FEATURE
INTERACTION MECHANISM
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
FORMS Contd.
Other Fields
Instruction
Configuration
Layout
Single Page Form
Instruction
Configuration
Layout
Multiple Page Form
Instruction
Configuration
Layout
Forms Navigation
Mouse Pick
Instruction
Configuration
Layout
Arrow Keys Bi-directional
Instruction
Configuration
Layout
Arrow Keys - Wrap Around
Instruction
Configuration
Layout
Tab Key Bi-directional
Instruction
Configuration
Layout

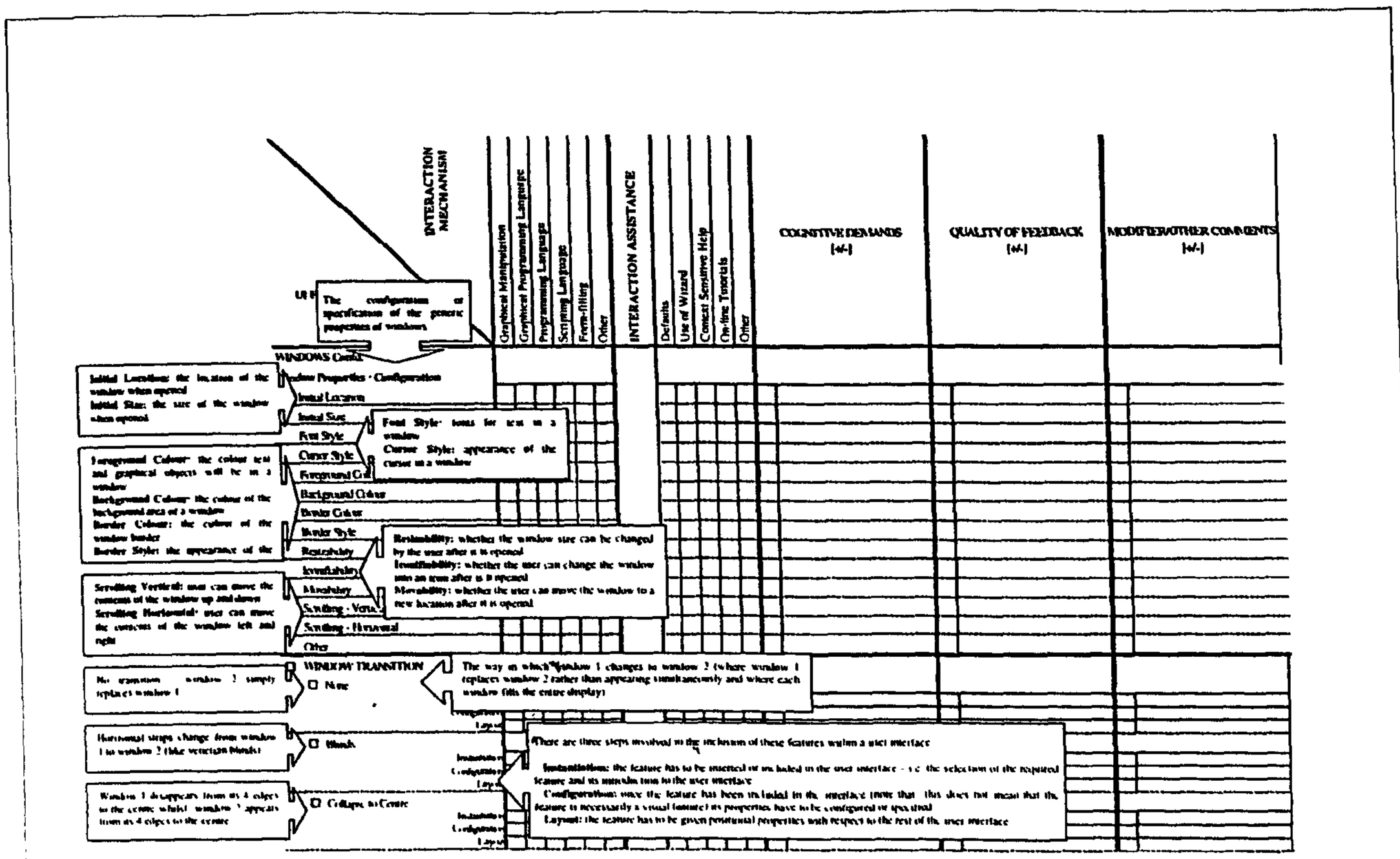

UI FEATURE
INTERACTION MECHANISM
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
Tab key used to move cursor amongst or within the fields on the form. Cursor moves from last field to first field and vice versa
FORMS Cond.
Tab Key Wrap Around
Instantiation
Configuration
Layout
Other Navigation
Instantiation
Configuration
Layout

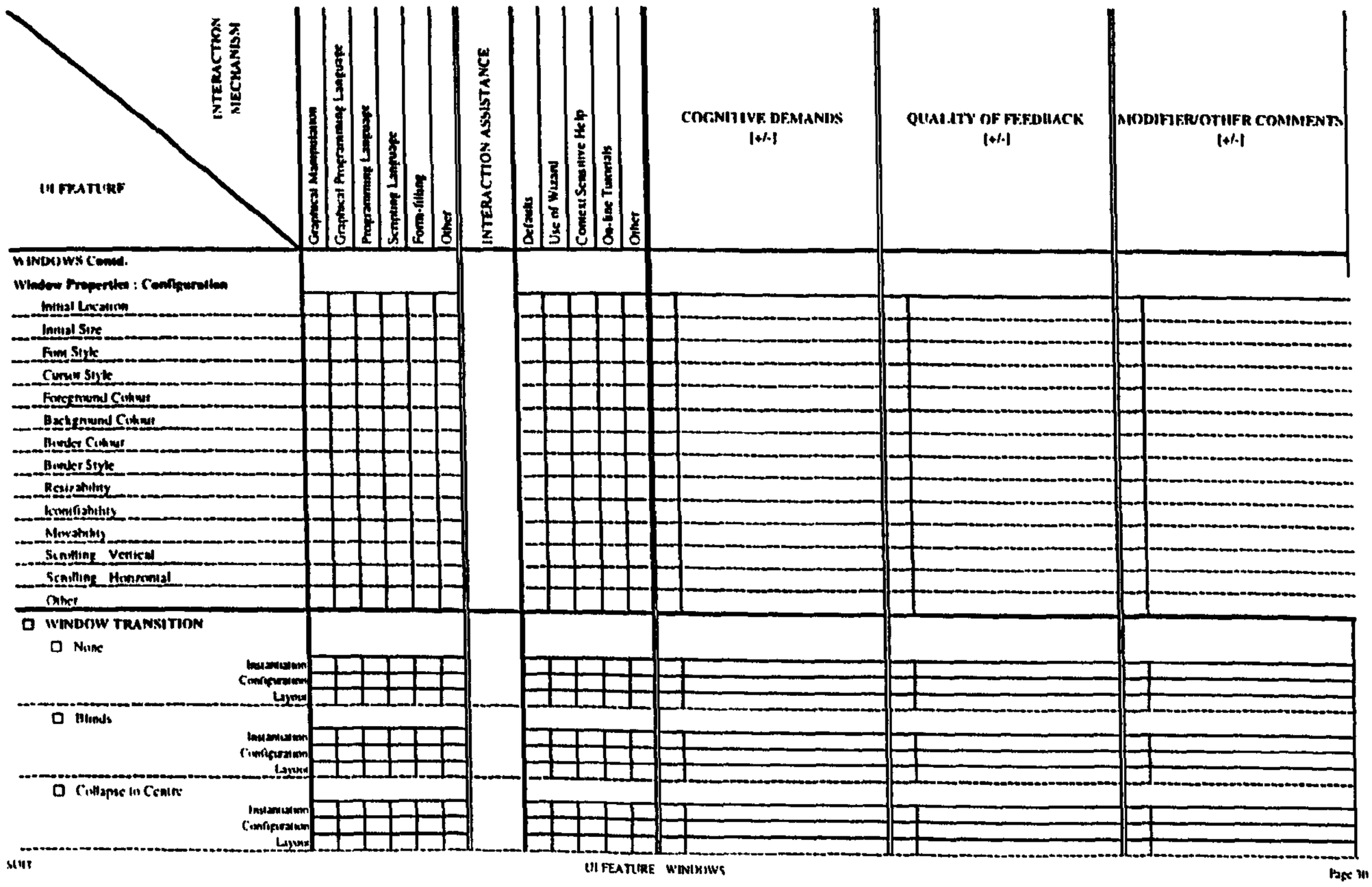

UI FEATURE
INTERACTION MECHANISM
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
FORMS Cond.
Tab Key Wrap Around
Instantiation
Configuration
Layout
Other Navigation
Instantiation
Configuration
Layout

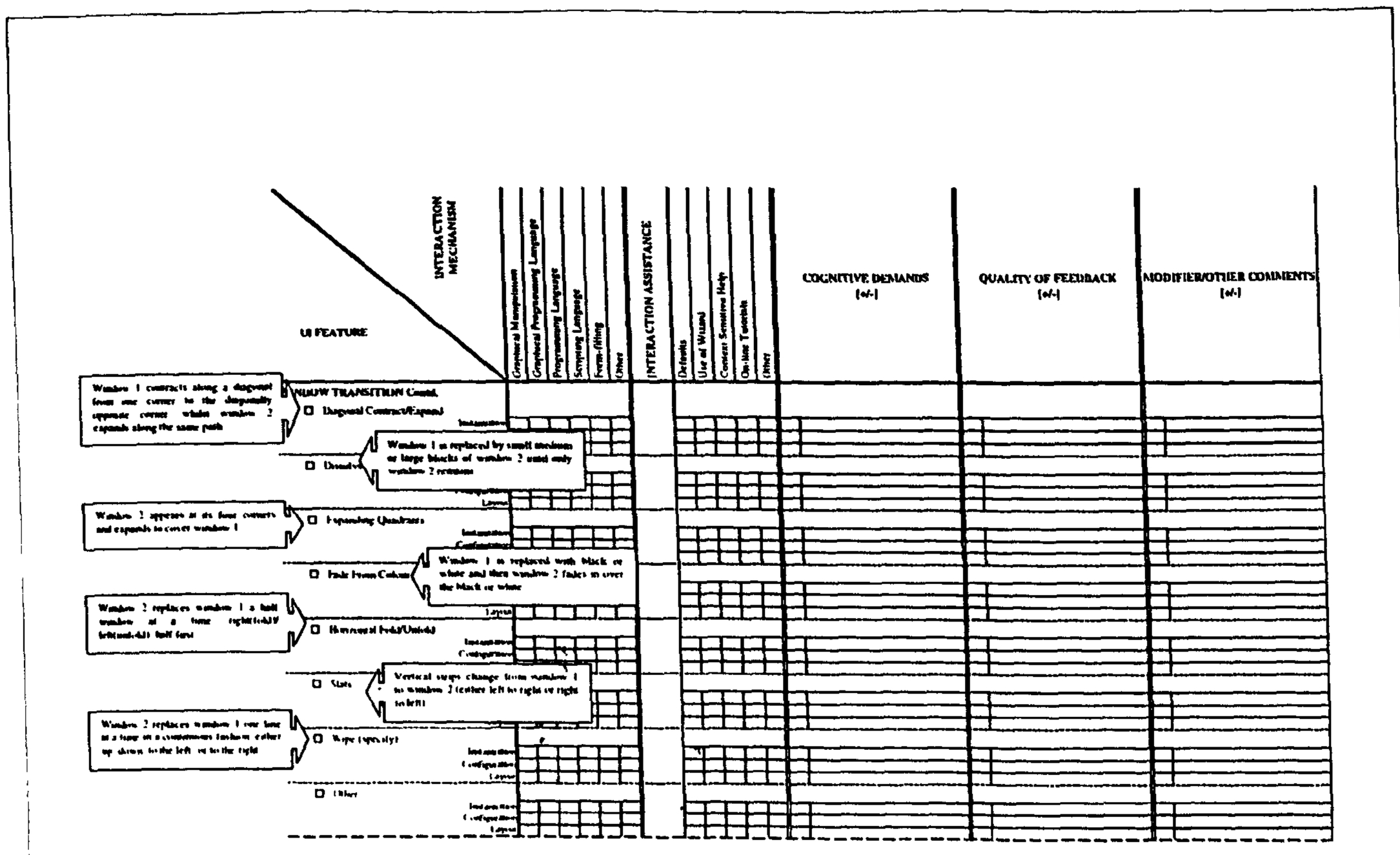
WINDOWS is concerned with the screen areas used for presentation of interface components

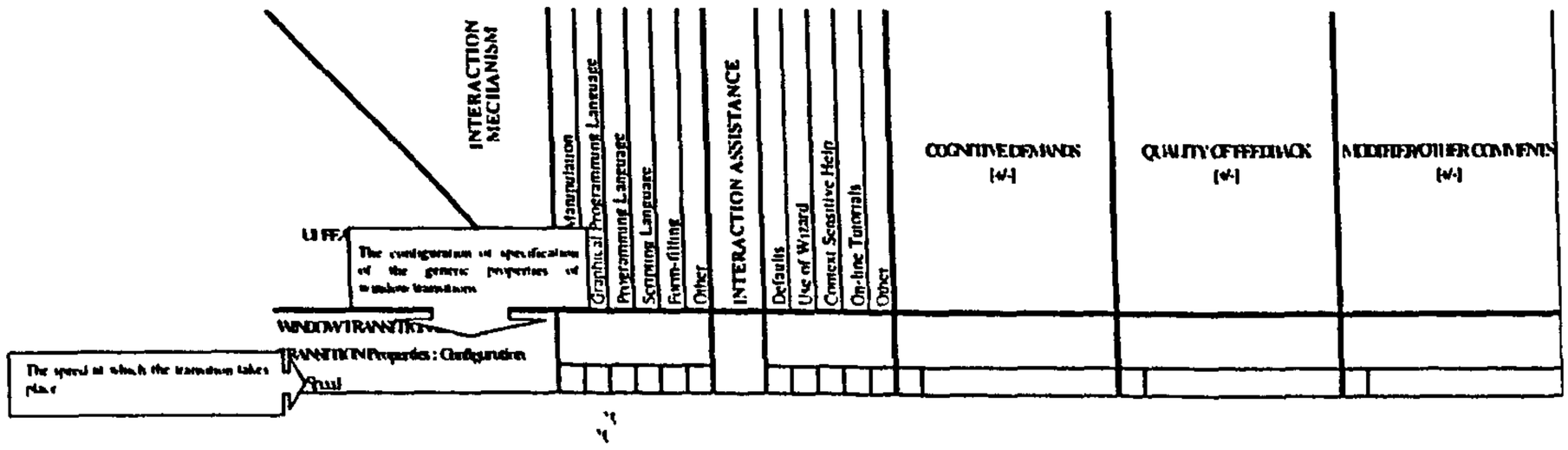


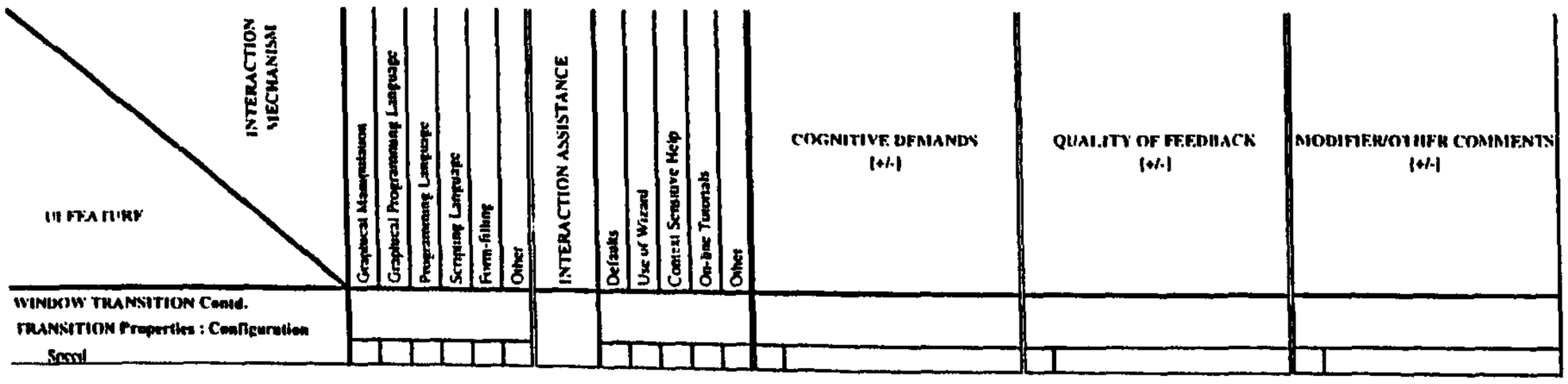


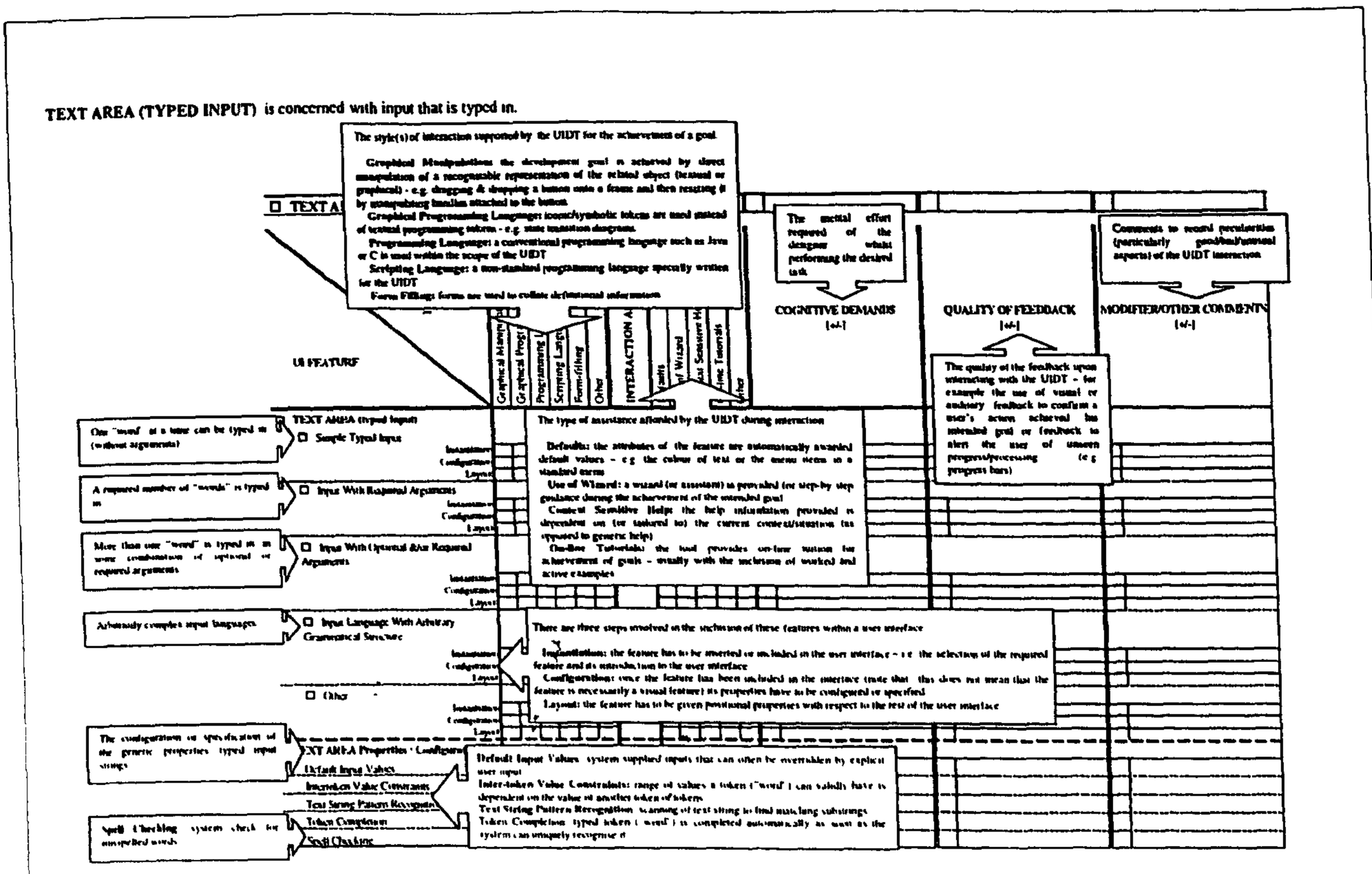

INTERACTION MECHANISM
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
The configuration or specification of the generic properties of windows
WINDOWS Cont'd
Window Properties : Configuration
Initial Location: the location of the window when opened
Initial Size: the size of the window when opened
Font Style: fonts for use in a window
Cursor Style: appearance of the cursor in a window
Foreground Colour: the colour text and graphical objects will be in a window
Background Colour: the colour of the background area of a window
Border Colour: the colour of the window border
Border Style: the appearance of the
Resizability: whether the window size can be changed by the user after it is opened
Iconifiability: whether the user can change the window into an icon after it is opened
Movability: whether the user can move the window to a new location after it is opened
Scrolling: Vertical: user can move the contents of the window up and down
Scrolling: Horizontal: user can move the contents of the window left and right
Initial Size
Font Style
Cursor Style
Foreground Colour
Background Colour
Border Colour
Border Style
Resizability
Iconifiability
Movability
Scrolling - Vertical
Scrolling - Horizontal
Other
WINDOW TRANSITION
The way in which window 1 changes to window 2 (where window 1 replaces window 2 rather than appearing simultaneously and where each window fills the entire display)
No transition: window 1 simply replaces window 1
None
Horizontal strips change from window 1 to window 2 (like venetian blinds)
Blinds
Window 1 disappears from as 4 edges to the centre whilst window 2 appears from as 4 edges to the centre
Collapse to Centre
There are three steps involved in the inclusion of these features within a user interface
Instantiation: the feature has to be inserted or included in the user interface - i.e. the selection of the required feature and its inclusion into the user interface
Configuration: once the feature has been included in the interface (note that this does not mean that the feature is necessarily a visual feature) its properties have to be configured or specified
Layout: the feature has to be given positional properties with respect to the rest of the user interface
Instantiation
Configuration
Layout
SUIT
UI FEATURE WINDOWS
Page 6/10

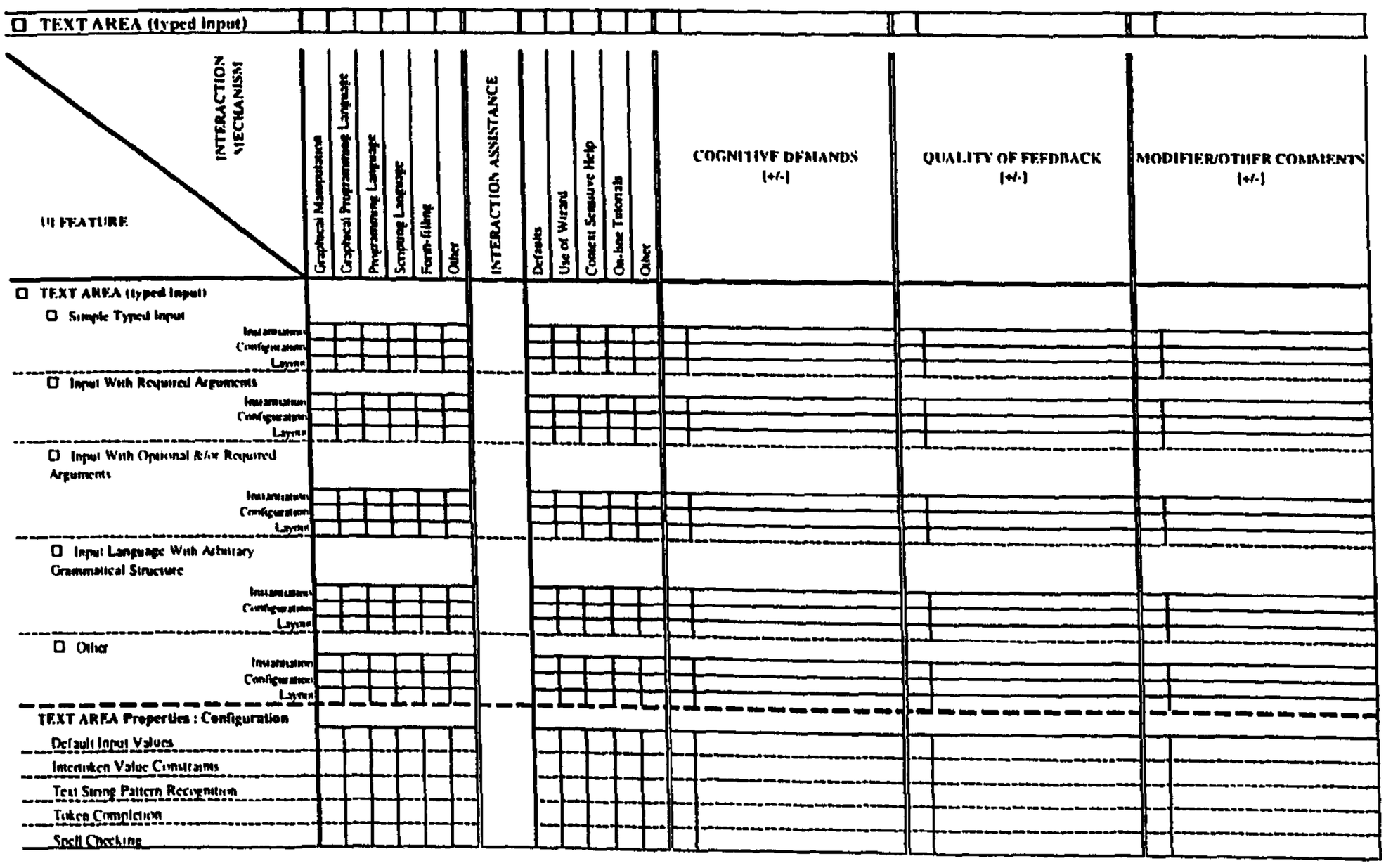

UI FEATURE
INTERACTION MECHANISM
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
WINDOWS Cont'd.
Window Properties : Configuration
Initial Location
Initial Size
Font Style
Cursor Style
Foreground Colour
Background Colour
Border Colour
Border Style
Resizability
Iconifiability
Movability
Scrolling - Vertical
Scrolling - Horizontal
Other
WINDOW TRANSITION
None
Instantiation
Configuration
Layout
Blinds
Instantiation
Configuration
Layout
Collapse to Centre
Instantiation
Configuration
Layout
SUIT
UI FEATURE WINDOWS
Page 7/10


UI FEATURE
INTERACTION MECHANISM
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
WINDOW TRANSITION Cond.
Diagonal Contract/Expand
Dissolve
Expanding Quadrants
Fade From Colour
Horizontal Fold/Unfold
Slats
Wipe (specify)
Other
Instantiation
Configuration
Layout
Window 1 contracts along a diagonal from one corner to the diagonally opposite corner while window 2 expands along the same path
Window 1 is replaced by small random or large blocks of window 2 until only window 2 remains
Window 2 appears at its four corners and expands to cover window 1
Window 1 is replaced with black or white and then window 2 fades in over the black or white
Window 2 replaces window 1 a half window at a time right(left)/left(right)-half first
Vertical strips change from window 1 to window 2 (either left to right or right to left)
Window 2 replaces window 1 one line at a time on a continuous fashion either up down to the left or to the right

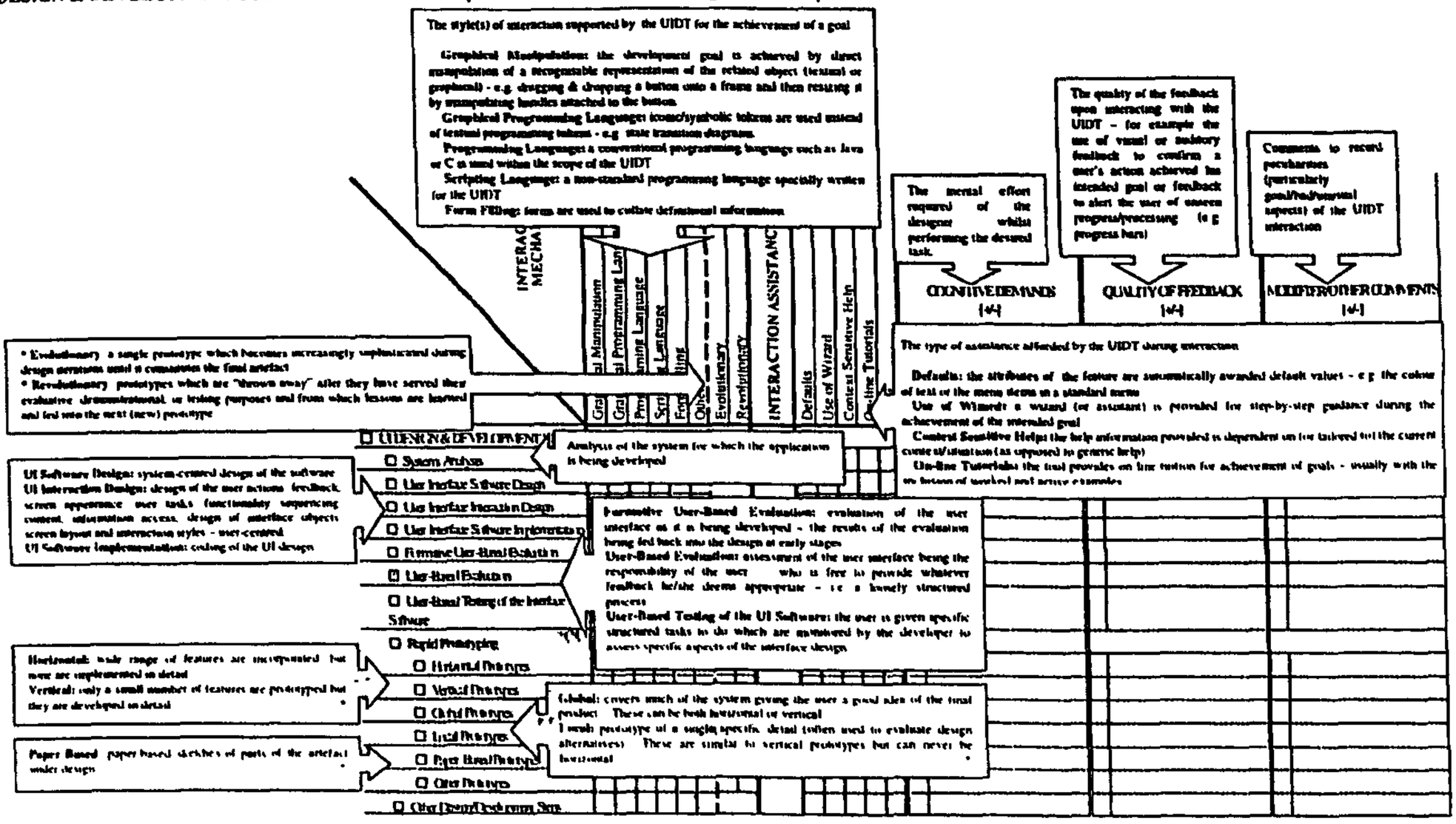

INTERACTION MECHANISM
Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
UI FEATURE
The configuration or specification of the generic properties of window transitions
WINDOW TRANSITIONS
TRANSITION Properties : Configuration
Speed
The speed at which the transition takes place
SUIT
UI FEATURE - WINDOWS
Page 0/12


INTERACTION MECHANISM
UI FEATURE
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
WINDOW TRANSITION Contd.
TRANSITION Properties : Configuration
Speed

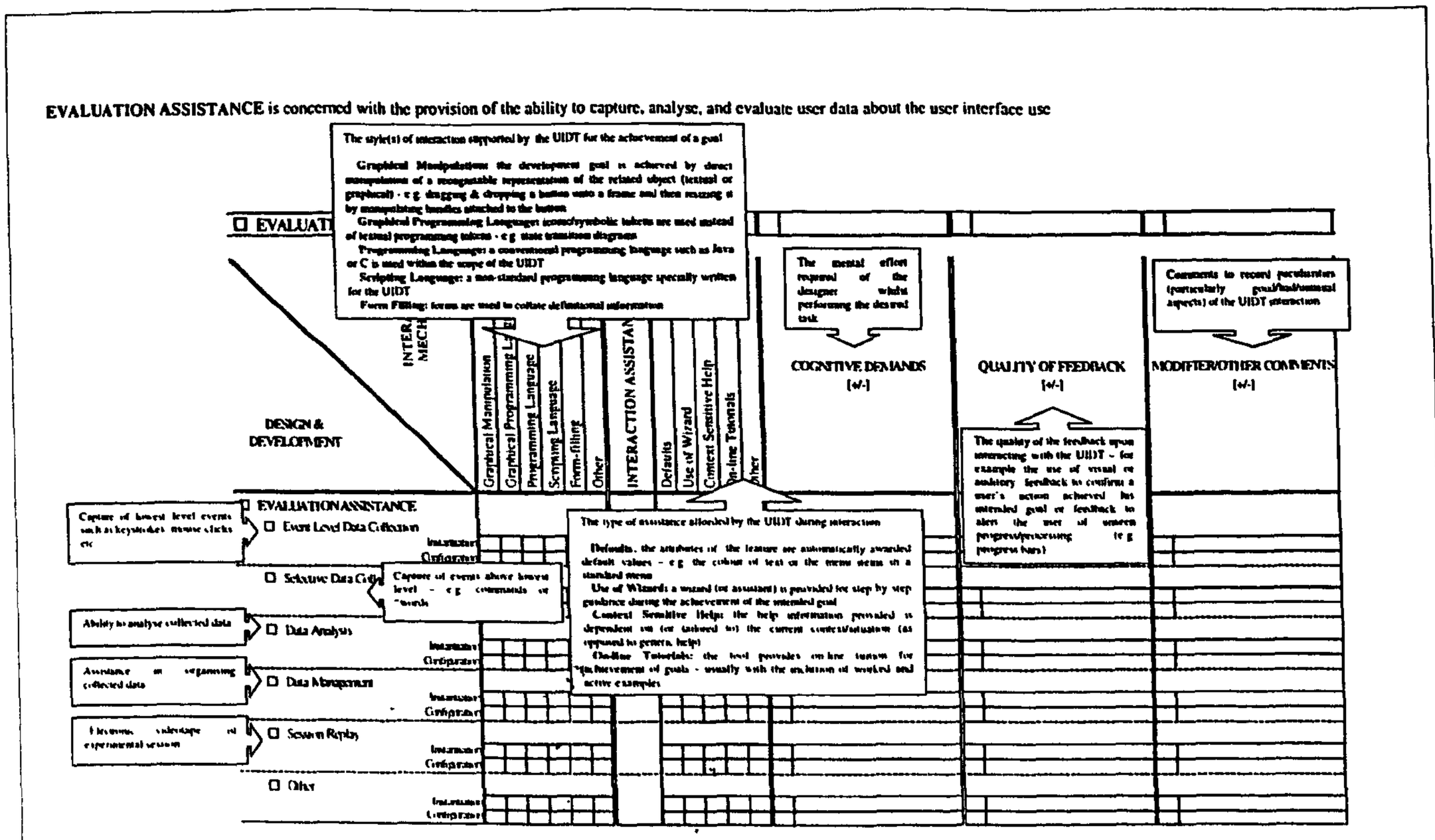

TEXT AREA (TYPED INPUT) is concerned with input that is typed in.
The style(s) of interaction supported by the UIDT for the achievement of a goal.
Graphical Manipulations: the development goal is achieved by direct manipulation of a recognisable representation of the related object (textual or graphical) - e.g. dragging & dropping a button onto a frame and then resizing it by manipulating handles attached to the button.
Graphical Programming Language: iconic/symbolic tokens are used instead of textual programming tokens - e.g. state transition diagrams.
Programming Language: a conventional programming language such as Java or C is used within the scope of the UIDT
Scripting Language: a non-standard programming language specially written for the UIDT
Form-filling: forms are used to collate alpha/numerical information
The mental effort required of the designer whilst performing the desired task
COGNITIVE DEMANDS [+/-]
Comments to record peculiarities (particularly good/bad/unusual aspects) of the UIDT interaction
MODIFIER/OTHER COMMENTS [+/-]
QUALITY OF FEEDBACK [+/-]
The quality of the feedback upon interacting with the UIDT - for example the use of visual or auditory feedback to confirm a user's action achieved his intended goal or feedback to alert the user of unseen progress/processing (e.g. progress bars)
UI FEATURE
Graphical Manip
Graphical Progr
Programming L
Scripting Lang
Form-filling
Other
INTERACTION A
Defaults
Use of Wizard
Context Sensitive H
On-line Tutorials
Other
TEXT AREA (input Input)
One "word" at a time can be typed in (without arguments)
Simple Typed Input
A required number of "words" is typed in
Input With Required Arguments
More than one "word" is typed in in some combination of optional or required arguments
Input With Optional &/or Required Arguments
Arbitrarily complex input languages
Input Language With Arbitrary Grammatical Structure
Other
Instantiation
Configuration
Layout
The type of assistance afforded by the UIDT during interaction
Defaults: the attributes of the feature are automatically awarded default values - e.g. the colour of text or the menu items on a standard menu
Use of Wizard: a wizard (or assistant) is provided for step-by-step guidance during the achievement of the intended goal
Context Sensitive Help: the help information provided is dependent on (or tailored to) the current context/situation (as opposed to generic help)
On-line Tutorials: the tool provides on-line tuition for achievement of goals - usually with the inclusion of worked and active examples
There are three steps involved in the inclusion of these features within a user interface
Instantiation: the feature has to be inserted or included in the user interface - i.e. the selection of the required feature and its introduction to the user interface
Configuration: once the feature has been included in the interface (note that this does not mean that the feature is necessarily a visual feature) its properties have to be configured or specified
Layout: the feature has to be given positional properties with respect to the rest of the user interface
The configuration or specification of the generic properties (typed input strings)
TEXT AREA Properties : Configuration
Default Input Values
Inter-token Value Constraints
Text String Pattern Recognition
Token Completion
Spell Checking
Spell Checking: system checks for misspelled words
Default Input Values: system supplied inputs that can often be overridden by explicit user input
Inter-token Value Constraints: range of values a token ("word") can validly have is dependent on the value of another token or tokens
Text String Pattern Recognition: scanning of text string to find matching substrings
Token Completion: typed token ("word") is completed automatically as soon as the system can uniquely recognise it
SUIT
UI FEATURE - TEXT AREA
Page 010

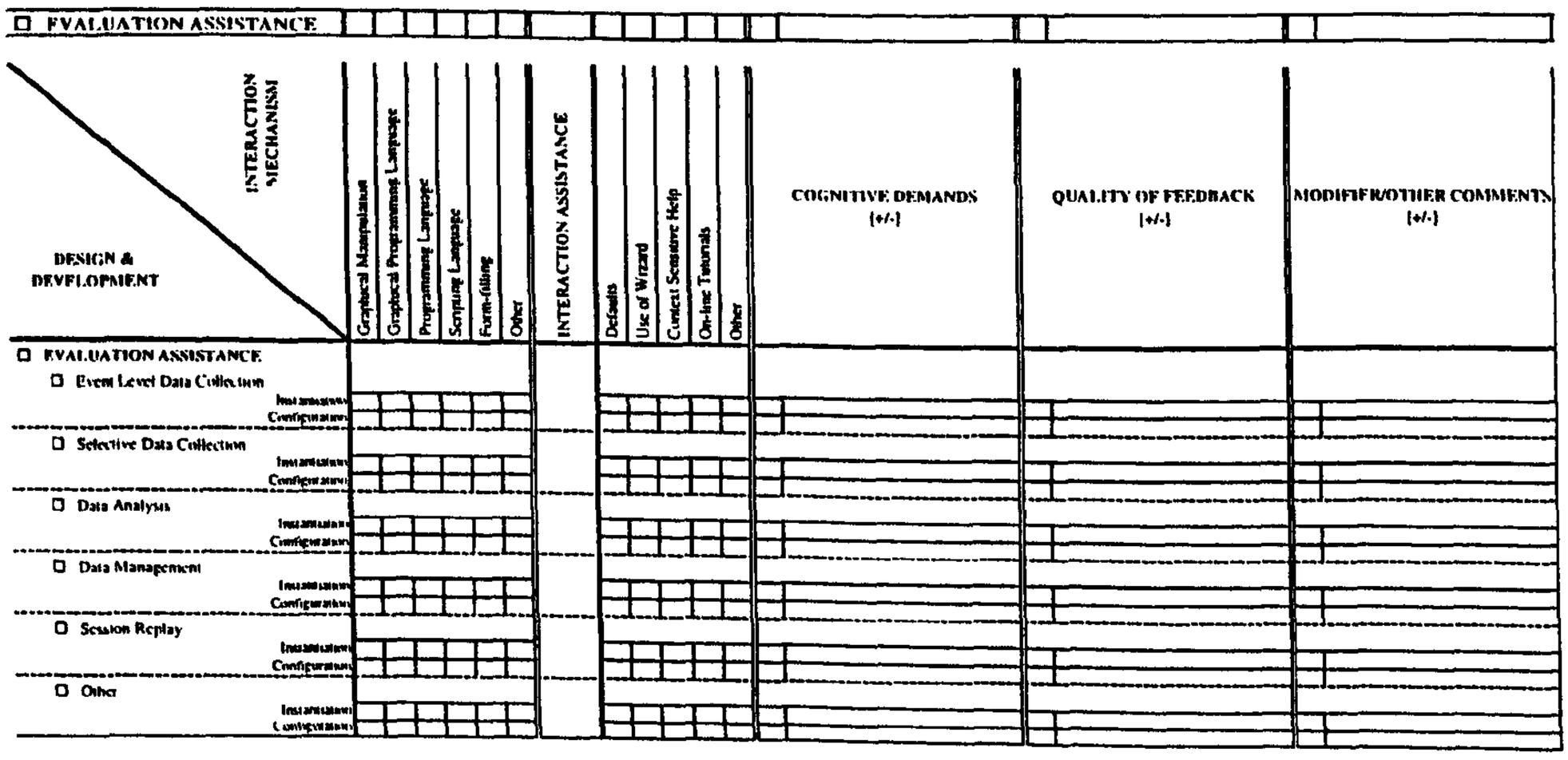

TEXT AREA (typed input)
UI FEATURE
INTERACTION MECHANISM
Graphical Manipulations
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
TEXT AREA (typed input)
Simple Typed Input
Input With Required Arguments
Input With Optional &/or Required Arguments
Input Language With Arbitrary Grammatical Structure
Other
Instantiation
Configuration
Layout
TEXT AREA Properties : Configuration
Default Input Values
Inter-token Value Constraints
Text String Pattern Recognition
Token Completion
Spell Checking
SUIT
UI FEATURE - TEXT AREA
Page 11

UI DESIGN & DEVELOPMENT STEPS is concerned with the processes involved with the design and development of a user interface.



DESIGN & DEVELOPMENT \ INTERACTION MECHANISM	Graphical Manipulation	Graphical Programming Language	Programming Language	Scripting Language	Form-filling	Other	Evolutionary	Revolutionary	INTERACTION ASSISTANCE	Defaults	Use of Wizard	Context Sensitive Help	On-line Tutorials	Other	COGNITIVE DEMANDS [+/-]	QUALITY OF FEEDBACK [+/-]	MODIFIER/OTHER COMMENTS [+/-]
☐ UI DESIGN & DEVELOPMENT STEPS																	
☐ Rapid Prototyping																	
☐ UI DESIGN & DEVELOPMENT STEPS																	
☐ Systems Analysis																	
☐ User Interface Software Design																	
☐ User Interface Interaction Design																	
☐ User Interface Software Implementation																	
☐ Formative User Based Evaluation																	
☐ User Based Evaluation																	
☐ User-Based Testing of the Interface Software																	
☐ Rapid Prototyping																	
☐ Horizontal Prototypes																	
☐ Vertical Prototypes																	
☐ Global Prototypes																	
☐ Local Prototypes																	
☐ Paper Based Prototypes																	
☐ Other Prototypes																	
☐ Other Design/Development Steps																	


EVALUATION ASSISTANCE is concerned with the provision of the ability to capture, analyse, and evaluate user data about the user interface use
The style(s) of interaction supported by the UIDT for the achievement of a goal
Graphical Manipulation: the development goal is achieved by direct manipulation of a recognisable representation of the related object (textual or graphical) - e g dragging & dropping a button onto a frame and then resizing it by manipulating handles attached to the button
Graphical Programming Language: iconic/symbolic tokens are used instead of textual programming tokens - e g state transition diagrams
Programming Language: a conventional programming language such as Java or C is used within the scope of the UIDT
Scripting Language: a non-standard programming language specially written for the UIDT
Form-filling: forms are used to collate declarational information
The mental effort required of the designer whilst performing the desired task
COGNITIVE DEMANDS [+/-]
Comments to record peculiarities (particularly good/bad/unusual aspects) of the UIDT interaction
MODIFIER/OTHER COMMENTS [+/-]
QUALITY OF FEEDBACK [+/-]
The quality of the feedback upon interacting with the UIDT - for example the use of visual or auditory feedback to confirm a user's action achieved his intended goal or feedback to alert the user of unseen progress/processing (e g progress bars)
DESIGN & DEVELOPMENT
INTERACTION MECHANISM
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
The type of assistance afforded by the UIDT during interaction
Defaults: the attributes of the feature are automatically awarded default values - e g the colour of text on the menu items in a standard menu
Use of Wizards: a wizard (or assistant) is provided for step by step guidance during the achievement of the intended goal
Context Sensitive Help: the help information provided is dependent on (or tailored to) the current context/situation (as opposed to generic help)
On-line Tutorials: the tool provides on-line tuition for achievement of goals - usually with the inclusion of worked and active examples
EVALUATION ASSISTANCE
Event Level Data Collection
Capture of lowest level events such as keystrokes mouse clicks etc
Selective Data Collection
Capture of events above lowest level - e g commands or words
Data Analysis
Ability to analyse collected data
Data Management
Assistance in organising collected data
Session Replay
Playback/videotape of experimental session
Other
Instantiation
Configuration
SUIT
DESIGN & DEVELOPMENT - EVALUATION ASSISTANCE
Page 615


EVALUATION ASSISTANCE
DESIGN & DEVELOPMENT
INTERACTION MECHANISM
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS [+/-]
QUALITY OF FEEDBACK [+/-]
MODIFIER/OTHER COMMENTS [+/-]
EVALUATION ASSISTANCE
Event Level Data Collection
Instantiation
Configuration
Selective Data Collection
Instantiation
Configuration
Data Analysis
Instantiation
Configuration
Data Management
Instantiation
Configuration
Session Replay
Instantiation
Configuration
Other
Instantiation
Configuration
SUIT
DESIGN & DEVELOPMENT - EVALUATION ASSISTANCE
Page 35

DEVELOPMENT AIDS is concerned with the provision of development technology to assist in the implementation of the user interface software

The style(s) of interaction supported by the UIDT for the achievement of a goal

Graphical Manipulation: the development goal is achieved by direct manipulation of a recognisable representation of the related object (textual or graphical) - e.g. dragging & dropping a button onto a frame and then resizing it by manipulating handles attached to the button

Graphical Programming Language: iconic/symbolic tokens are used instead of textual programming tokens - e.g. state transition diagrams.

Programming Language: a conventional programming language such as Java or C is used within the scope of the UIDT

Scripting Language: a non-standard programming language specially written for the UIDT

Form Filling: forms are used to collate development information

The mental effort required of the designer whilst performing the desired task

COGNITIVE DEMANDS [+/-]

Comments to record particulars (particularly good/bad/unusual aspects) of the UIDT interaction

MODIFIER/OTHER COMMENTS [+/-]

QUALITY OF FEEDBACK [+/-]

The quality of the feedback upon interacting with the UIDT - for example the use of visual or auditory feedback to confirm a user's action achieved his intended goal or feedback to alert the user of unseen progress/processing (e.g. progress bars)

The type of assistance afforded by the UIDT during interaction

Defaults: the attributes of the feature are automatically awarded default values - e.g. the colour of text in the menu items in a standard menu

Use of Wizard: a wizard (or assistant) is provided for step by step guidance during the achievement of the intended goal

Context Sensitive Help: the help information provided is dependent on (or tailored to) the current context/situation (as opposed to generic help)

On-line Tutorials: the tool provides on-line tuition for achievement of goals - usually with the inclusion of worked and active examples

DESIGN & DEVELOPMENT

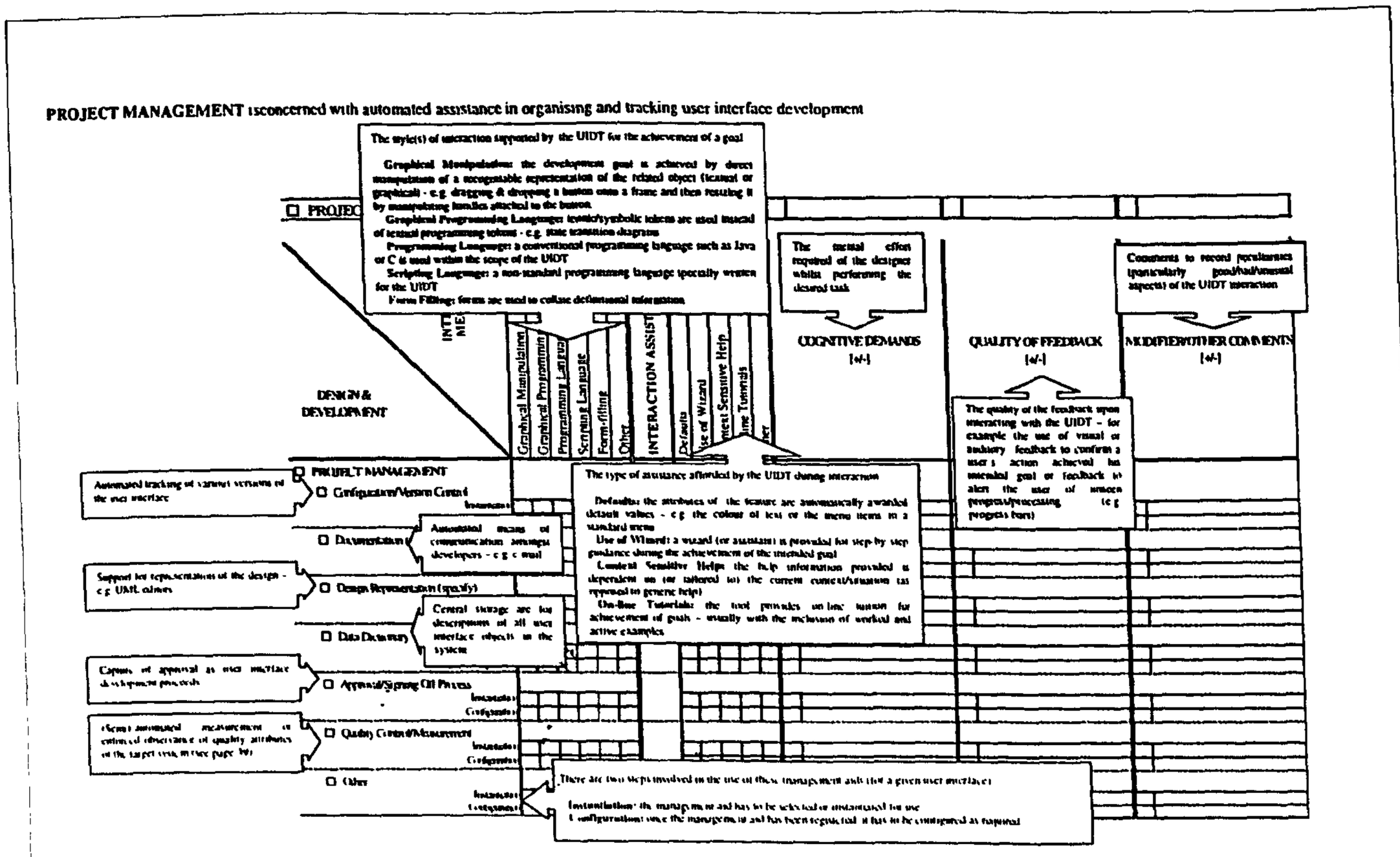
INTERACTION MECHANISM: Graphical Manipulation; Graphical Programming Language; Programming Language; Scripting Language; Form-filling; Other

INTERACTION ASSISTANCE: Defaults; Wizard; Sensitive Help; Tutorials

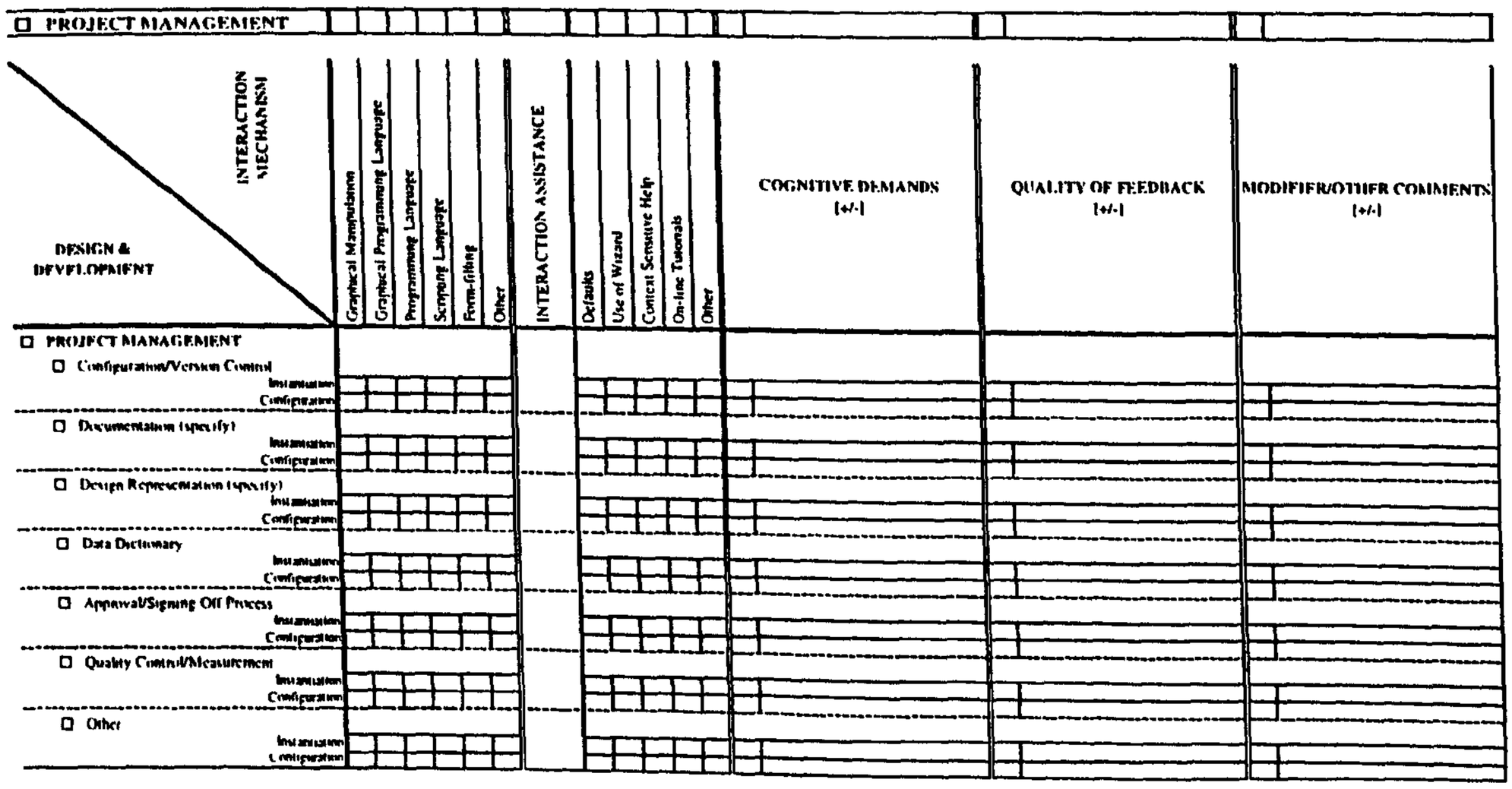
DEVELOPMENT AIDS

- Code Editor — Text editor configured to auto-format programming languages
- Interpreter — Programming language interpreter
- Compiler — Programming language compiler
- Step Through Execution — Ability to step line by line through the code and see the effects of each line when executed
- Debugger — Assistance to "fix" or find faults in code
- Property List — An editable list of properties for each component within the user interface
- Hierarchy Editor — Enables the hierarchical composition of the user interface components to be edited
- Class Browser — Ability to browse the classes used within a user interface or easily to observe their attributes, methods and inheritance
- File Comparison — Enables the contents of files (e.g. java class files) to be compared
- Other

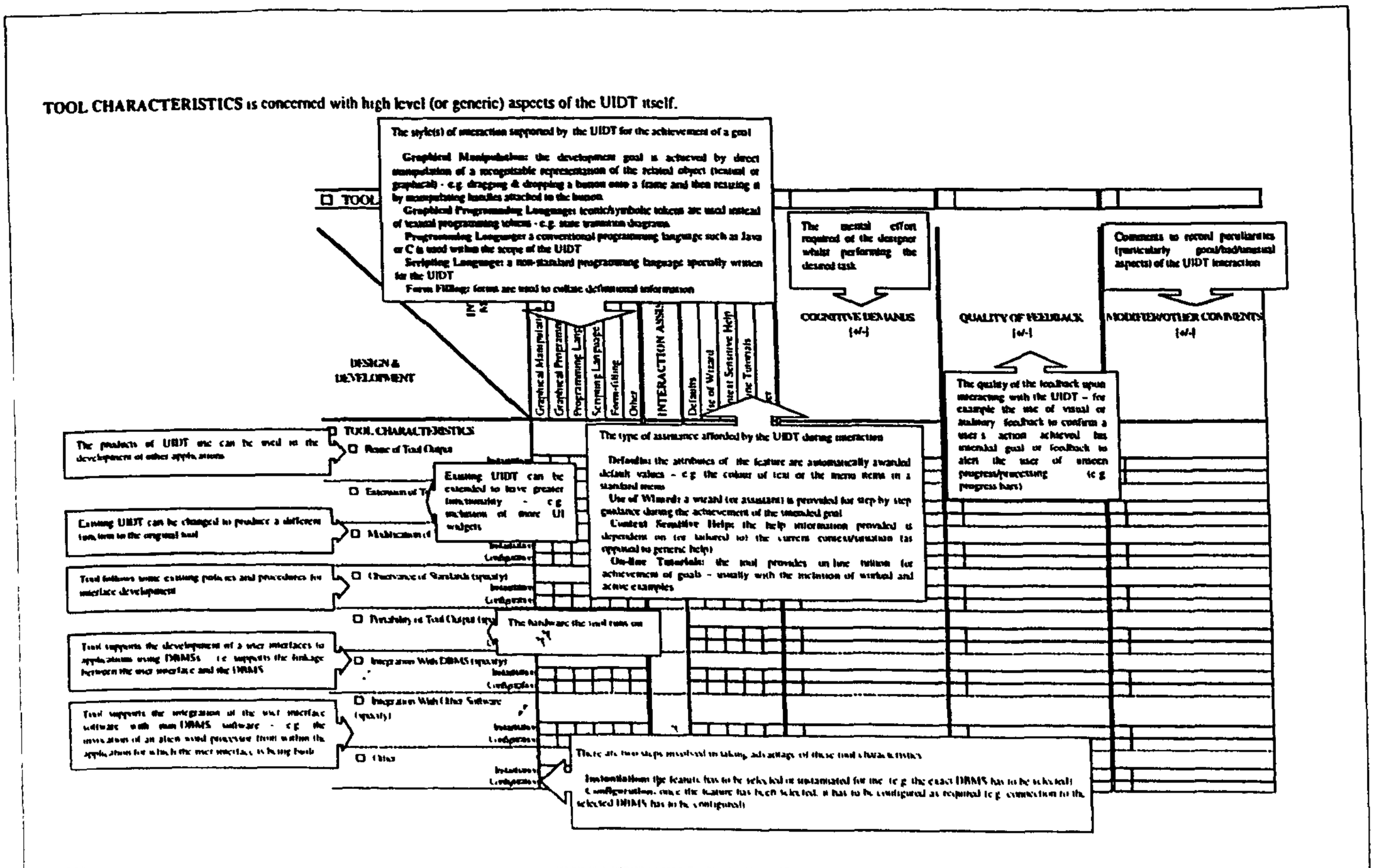
DESIGN & DEVELOPMENT \ INTERACTION MECHANISM	Graphical Manipulation	Graphical Programming Language	Programming Language	Scripting Language	Form-filling	Other	INTERACTION ASSISTANCE	Defaults	Use of Wizard	Context Sensitive Help	On-line Tutorials	Other	COGNITIVE DEMANDS [+/-]	QUALITY OF FEEDBACK [+/-]	MODIFIER/OTHER COMMENTS [+/-]
DEVELOPMENT AIDS															
Code Editor															
Code Editor — Instantiation															
Code Editor — Configuration															
Interpreter															
Interpreter — Instantiation															
Interpreter — Configuration															
Compiler															
Compiler — Instantiation															
Compiler — Configuration															
Step Through Execution															
Step Through Execution — Instantiation															
Step Through Execution — Configuration															
Debugger															
Debugger — Instantiation															
Debugger — Configuration															
Property List															
Property List — Instantiation															
Property List — Configuration															
Hierarchy Editor															
Hierarchy Editor — Instantiation															
Hierarchy Editor — Configuration															
Class Browser															
Class Browser — Instantiation															
Class Browser — Configuration															
File Comparison															
File Comparison — Instantiation															
File Comparison — Configuration															
Other															
Other — Instantiation															
Other — Configuration															



SUIT
DESIGN & DEVELOPMENT PROJECT MANAGEMENT
Page G17

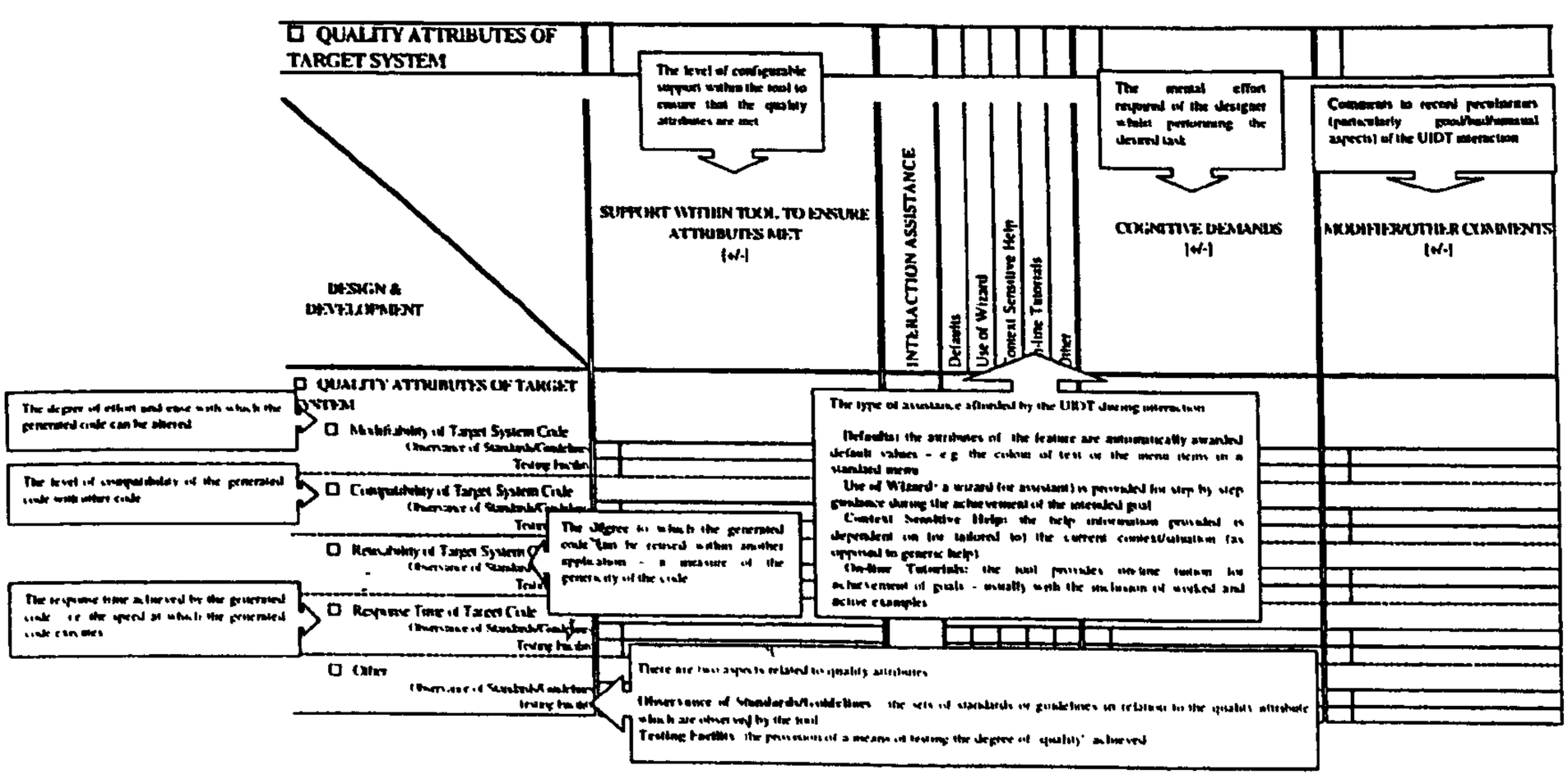


SUIT
DESIGN & DEVELOPMENT PROJECT MANAGEMENT
Page 17



TOOL CHARACTERISTICS
DESIGN & DEVELOPMENT
INTERACTION MECHANISM
Graphical Manipulation
Graphical Programming Language
Programming Language
Scripting Language
Form-filling
Other
INTERACTION ASSISTANCE
Defaults
Use of Wizard
Context Sensitive Help
On-line Tutorials
Other
COGNITIVE DEMANDS (+/-)
QUALITY OF FEEDBACK (+/-)
MODIFIER/OTHER COMMENTS (+/-)
TOOL CHARACTERISTICS
Reuse of Tool Output
Instantiation
Configuration
Extension of Tool Features
Instantiation
Configuration
Modification of Tool Features
Instantiation
Configuration
Observance of Standards (specify)
Instantiation
Configuration
Portability of Tool Output (specify)
Instantiation
Configuration
Integration With DBMS (specify)
Instantiation
Configuration
Integration With Other Software (specify)
Instantiation
Configuration
Other
Instantiation
Configuration

QUALITY ATTRIBUTES OF THE TARGET SYSTEM is concerned with the way in which the user interface code generated by the UIDT conforms to defined quality attributes



SUIT DESIGN & DEVELOPMENT - QUALITY ATTRIBUTES OF TARGET SYSTEM Page G10

QUALITY ATTRIBUTES OF TARGET SYSTEM											
DESIGN & DEVELOPMENT	SUPPORT WITHIN TOOL TO ENSURE ATTRIBUTES MET [+/-]		INTERACTION ASSISTANCE	Defaults	Use of Wizard	Context Sensitive Help	On-line Tutorials	Other	COGNITIVE DEMANDS [+/-]		MODIFIER/OTHER COMMENTS [+/-]	
QUALITY ATTRIBUTES OF TARGET SYSTEM												
Modifiability of Target System Code												
Observance of Standards/Guidelines												
Testing Facility												
Compatibility of Target System Code												
Observance of Standards/Guidelines												
Testing Facility												
Reusability of Target System Code												
Observance of Standards/Guidelines												
Testing Facility												
Response Time of Target Code												
Observance of Standards/Guidelines												
Testing Facility												
Other												
Observance of Standards/Guidelines												
Testing Facility												

SUIT DESIGN & DEVELOPMENT - QUALITY ATTRIBUTES OF TARGET SYSTEM Page H

APPENDIX C: HANDOUTS FOR THE EMPIRICAL STUDY OF SUIT

HUMAN-COMPUTER INTERACTION 4

1998-99 **Exercise A**

Hand Out Week 3
Value 3%

Hand In Week 9

Evaluation of User Interface Design Environments

Introduction

The aim of this exercise is for you to gain experience of evaluating various user interface design environments in the context of a specific design problem. User interface design environments cater for a variety of application domains. One such domain is web site design and this exercise will focus on the evaluation of web authoring tools.

Before starting the exercise, read the following problem scenario and instructions carefully.

Problem Scenario

A team of two people has been given the job of renewing the university web site. Whilst all information on the current web site is to be included in the new site, additional video, animation and sound-bites are to be introduced. The project is severely time restricted and has to meet imminent deadlines.

The team members are as follows:

- *team member 1*: graphic designer with experience of numerous graphical manipulation tools for graphics generation, no knowledge of HTML but familiar with using web browsers.
- *team member 2*: expert user interface designer who has limited knowledge of HTML but extensive experience of graphical user interface development environments and makes frequent use of web browsers.

The University of Glasgow has a selection of web authoring tools which are available for the project team to use. Neither team member has experience using any of the available tools. Given the imminent deadline for the project, they do not have the time to try out each tool in order to decide which is the most appropriate for their use, nor can they afford to make the wrong choice and have to change environments therefore wasting development time. The university has hired you (as an expert in user interface design evaluation) to advise the team on their choice of tool. Your job is therefore to assess the suitability of the web authoring tools with respect to the outlined project profile.

Instructions

- Working in pairs, select 2 web authoring tools from the following list: SiteBuilder[1], QuickSite[1], HomeSite[1], and AOLPress[1], and DreamWeaver[1].

[1]available via the NT network from workstations in the NT lab (GO81) in Lilybank Gardens:

- log onto an NT workstation;
- select the Network Neighbourhood;
- select Tobago;
- select HCI4 folder;
- select the tool of your choice & run the executable*.

* please note that these are evaluation copies and some of the products are time limited (e.g. 30 days) - as such we recommend that you try to complete tool use prior to the end of November.

- Following the method provided, complete an evaluation form for each of your chosen web authoring tools. **You are advised to complete the evaluation forms in pencil**. In accordance with the evaluation method, analyse the results obtained in order to select the most appropriate tool for the given problem scenario.

  While performing each evaluation, please use the provided log sheets to record the amount of time taken to complete the process for each tool and to note down any comments regarding the methods and forms (see later).

- Each person should then write a 2 page report which presents a recommendation for the use of a particular web authoring tool. The report should include:
  - a brief outline of the evaluation process;
  - a brief summary of each tool;
  - a rationale behind the recommendation to use one tool in preference to the other - i.e. based on the results of the evaluation, what made one tool more suitable than the other?

What You Should Submit

Each person should submit their individual report. In addition, as a team, you should submit all evaluation sheets and log sheets.

How Your Results Will Help Research

Jo McKirdy is a Ph.D. student in this department who is researching the selection process for user interface design environments. Some of you will be using Jo's method whilst some of you will be using an older method for the evaluation of the web authoring tools. The results obtained from your exercise will provide a valuable comparison of the two different methods and will therefore greatly assist Jo in her work. In particular, the log sheets you have been asked to fill in will provide an indication of the time required to use each of the methods and any comments will give her essential feedback.

Guidance

You will be given a short tutorial on the evaluation method you will be using for this exercise. Following this, if you have any questions at all please contact Jo McKirdy either by email - **jo@dcs.gla.ac.uk** - or at her office - **F153** - where she will be happy to assist. If you require additional copies of the methods or forms please also contact Jo.

Time Management

We suggest that each person should spend approximately 15 hours to complete this exercise giving a total of 30 man hours for the evaluation. A suggested breakdown of tasks is given below.

- prepare for evaluation (team) - 2 man hours
- gather data for each web authoring tool (team) - 10 man hours per tool [total 20 man hours]
- compare results (team) - 4 man hours
- write up (individual) - 2 man hours per person [total 4 man hours]

If you find that you are required to spend considerably more time than is suggested above, please let us know as soon as possible.

Log Sheet

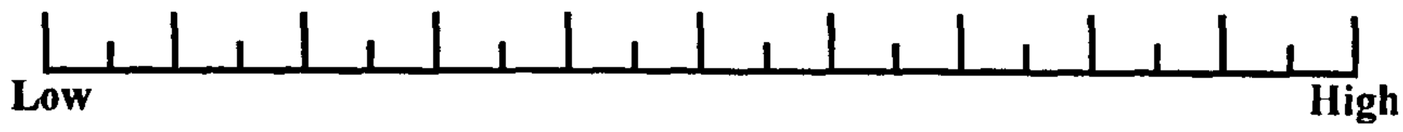
Date	Time Spent	Comment

Which evaluation method were you using? _____

This questionnaire is designed to assess the workload you experienced while performing your web authoring tool evaluation task. Workload can be caused by many different factors which I would like you to rate separately. Please read the descriptions of each of the following workload factors carefully and if you have any questions ask me. For each factor, rate the task of web authoring tool evaluation by marking the scale at the point which matches *your* experience. Please consider your responses carefully. Consider each scale individually. Your ratings will play an important role in the experiment being conducted so your active participation is essential to the success of this experiment, and is greatly appreciated.

MENTAL DEMAND How much mental and perceptual activity was required (e.g. thinking, deciding, calculating, remembering, looking, searching etc.) ? Was the task easy or demanding, simple or complex, exacting or forgiving?

MENTAL DEMAND

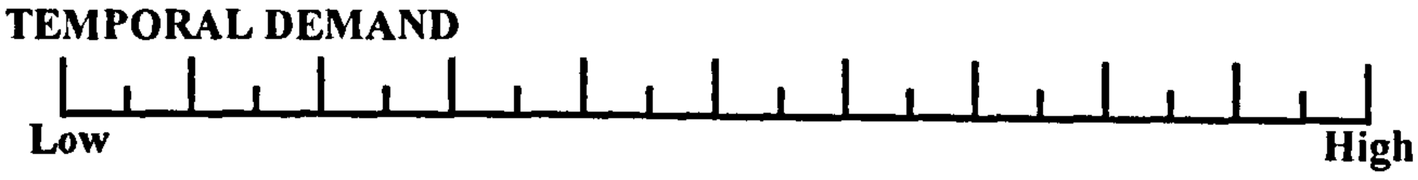


PHYSICAL DEMAND How much physical activity was required (e.g. pushing, pulling, turning, controlling, activating etc.)? Was the task easy or demanding, slow or brisk, slack or strenuous, restful or laborious?

PHYSICAL DEMAND

TEMPORAL DEMAND How much time pressure did you feel due to the rate or pace at which the task elements occurred? Was the pace slow and leisurely or rapid and frantic?

TEMPORAL DEMAND



PERFORMANCE How successful do you think you were in accomplishing the goals of the task set by the experimenter? How satisfied were you with your performance in accomplishing these goals? (Please note that the scale goes from "good" on the *left* to "poor" on the *right.)*

PERFORMANCE

EFFORT How hard did you have to work (mentally and physically) to accomplish your level of performance?

EFFORT

FRUSTRATION LEVEL How insecure, discouraged, irritated, stressed and annoyed *versus* secure, gratified, content, relaxed, and complacent did you fell during the task?

FRUSTRATION

There follows a series of pairs of the workload factors which were introduced over the page (for example, Temporal Demands vs. Mental Demands). For each pair, choose the factor which *you* consider was more important to *your* experience of workload during the web authoring tool evaluation task. For example, if you considered temporal demands as a more important contributor to workload than mental demands during your evaluation task, you would circle the temporal demands title in the pair [Temporal Demands v Mental Demands].

Mental Demand	v	Effort
Performance	v	Physical Demand
Frustration	v	Effort
Temporal Demand	v	Performance
Physical Demand	v	Frustration
Mental Demand	v	Physical Demand
Effort	v	Performance
Mental Demand	v	Frustration
Physical Demand	v	Effort
Frustration	v	Performance
Temporal Demand	v	Frustration
Effort	v	Temporal Demand
Performance	v	Mental Demand
Temporal Demand	v	Physical Demand
Mental Demand	v	Temporal Demand

APPENDIX D: FORMULAE USED FOR STATISTICAL ANALYSIS IN CHAPTER 5

Calculations for t-Test

$$t = \frac{\overline{X}_1 - \overline{X}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

formula for *t*-Test where:
$\overline{X}_1$ = mean for group 1
$\overline{X}_2$ = mean for group 2
s_1^2 = (standard deviation)^2 for group 1
s_2^2 = (standard deviation)^2 for group 2
n_1 = number of subjects in group 1
n_2 = number of subjects in group 2

$$variance\ (s^2) = \frac{\sum(x_i - \overline{x})^2}{n - 1}$$

formula for *variance* where:
x_i = i^{th} value of x
$\overline{x}$ = mean value of x
n = number of subjects

Calculations for Mann-Whitney Test

$$U_1 = mn + \frac{m(m+1)}{2} - T$$

formula for *Mann-Whitney* where:
m = size of group 1
n = size of group 2
T = sum of rank values for smaller group

$$U_2 = mn - U_1$$

Calculations for X^2 (Chi-square) Test

$$X^2 = \sum\left[\frac{(O - E)^2}{E}\right]$$

formula for X^2 where:
O = observed value
E = expected value

$$E = \frac{\text{row total} * \text{column total}}{\text{grand total}}$$

Calculations for Pearson (r)

formula for r where:
n = number of data points
X = value in set X
Y = value is set Y

$$r = \frac{n(\sum XY) - (\sum X)(\sum Y)}{\sqrt{\left[n\sum X^2 - (\sum X)^2\right]\left[n\sum Y^2 - (\sum Y)^2\right]}}$$

(Lindley and Scott, 1984, Woods et al., 1986)

APPENDIX E: SUIT VISUALISATION ENVIRONMENT USER MANUAL

SUIT
by

JOANNA LUMSDEN

User Manual

Table of Contents

Table of Figures

1 Introduction

Thank you for choosing to use SUIT. Making extensive use of diagrams (some of them annotated), this *minimal* scenario-based user manual will show you how to access the functionality provided in SUIT, in what is hoped is a clear and easy-to-follow manner. SUIT should be used with reference to the SUIT Framework and Methodology documentation.

The SUIT data visualisation environment is a research tool for the representation of data required for the selection of user interface development tools. Should you have any problems with, or questions about SUIT, if you would like further information, or if you would like to give the author some useful feedback please contact her:

e-mail: jo@dcs.gla.ac.uk
tel. no: 0141 330 6045
www: http://www.dcs.gla.ac.uk/~jo/SUIT

2 Installing SUIT

To install SUIT follow the simple steps outlined below:

1. insert the SUIT CD/3½ floppy;
2. double click on **`suit.exe`**;
3. by default, SUIT is installed into **`C:\suit`** on your PC. This and other defaults are therefore incorporated into the **`suit.bat`** file provided. *Please check the classpath specified in* **`suit.bat`** *to ensure that all settings are correct for your machine;*
4. copy the **`SuitTemplateDB.mdb`** Access Database into the **`C:\suit`** folder (or equivalent);
5. create an ODBC Driver for the supplied **`SuitTemplateDB.mdb`** Access Database (see system help);
6. create an ODBC Driver for each of the supplied tool profile Access databases (see system help);
7. move **`shortcut to suit.bat`** onto your desktop;
8. launch SUIT by double-clicking on the SUIT  icon.

3 Entering Data

SUIT is a research *data visualisation* environment. As such, it has not focused on the provision of a data-entry environment. Instead, it provides a template database (`SuitTemplateDB.mdb`) which is a standard Microsoft Access™ database, the schema for which has been defined according to the SUIT framework. To enter data for use in the SUIT tool, complete the following steps:

For ***each*** required tool template (including ideal templates):

1. create a copy of `SuitTemplateDB.mdb` within the `C:\suit` folder (or equivalent) and rename it – e.g. `ToolX.mdb`;
2. create an ODBC Driver for the new database (see system help);
3. open the new database;
4. enter data as collected using the SUIT Framework and Methodology. Note: to "include" a data item in the selection process, check the inclusion - ☑/☒ - box for that row in the database form – see Figure 1.

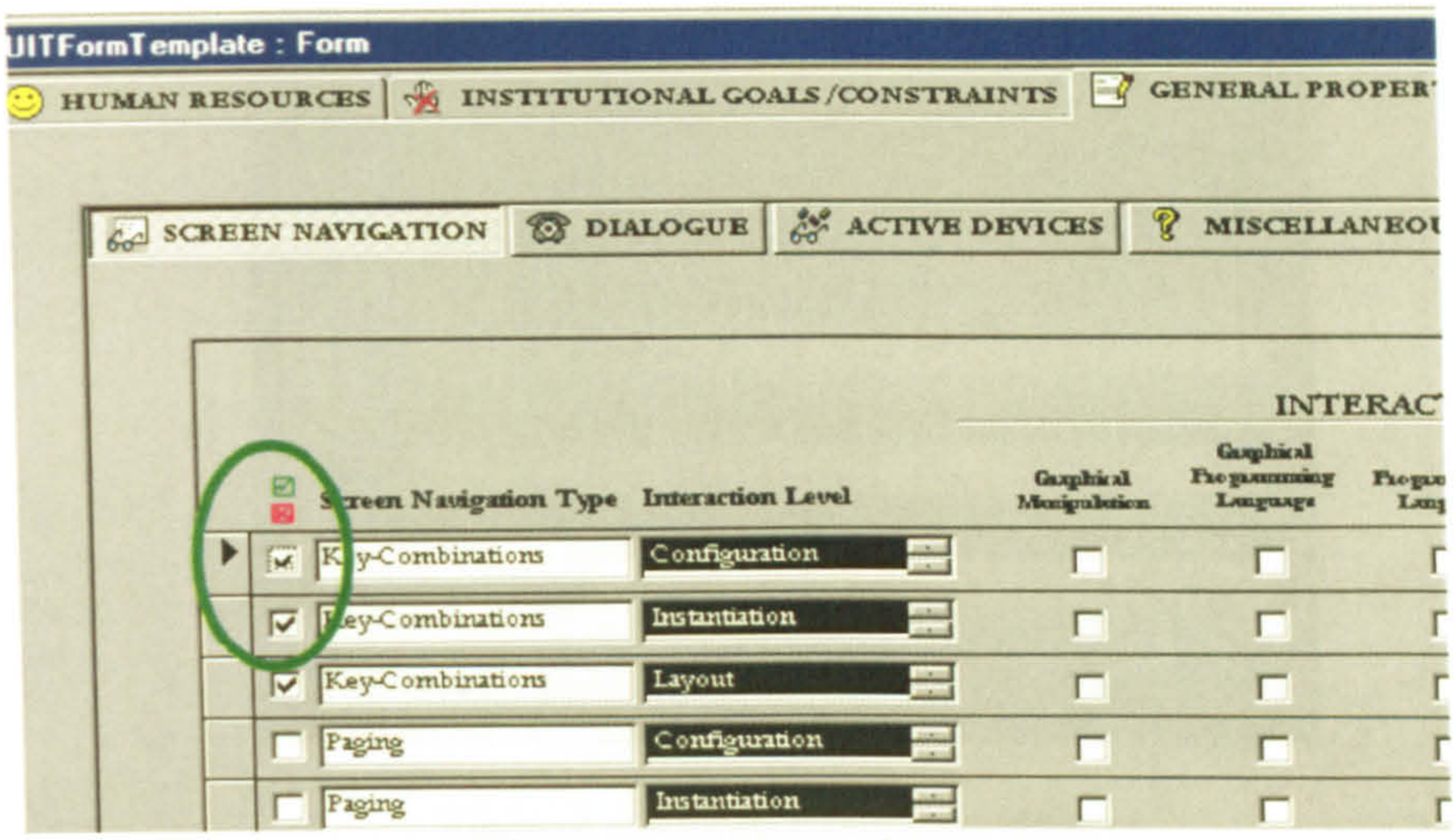


Figure 1 - *inclusion of data items*

4 Creating Your Data Visualisation

Once you have generated and completed the databases required for your comparison, you are ready to create a visualisation of the data in SUIT and then use the facilities provided by SUIT to assist in the comparison (or analysis) of that data. Depending on the nature of your required comparison, follow the appropriate steps below:

4.1 Performing A Generic Comparison

1 launch **SUIT**;

2 select `New Visualisation...` from the Visualisation menu – see Figure 2;

Figure 2 – *creating a new visualisation*

3 select to perform a `Generic Comparison`, and provide: the name of the **ODBC driver** for the template database; and the names of the **ODBC drivers** for the tool databases as shown in Figure 3 – and select '`OK`';

4 once SUIT has completed the generation of the data visualisation based on the information specified in step 3, you will be ready to begin analysis of your data.

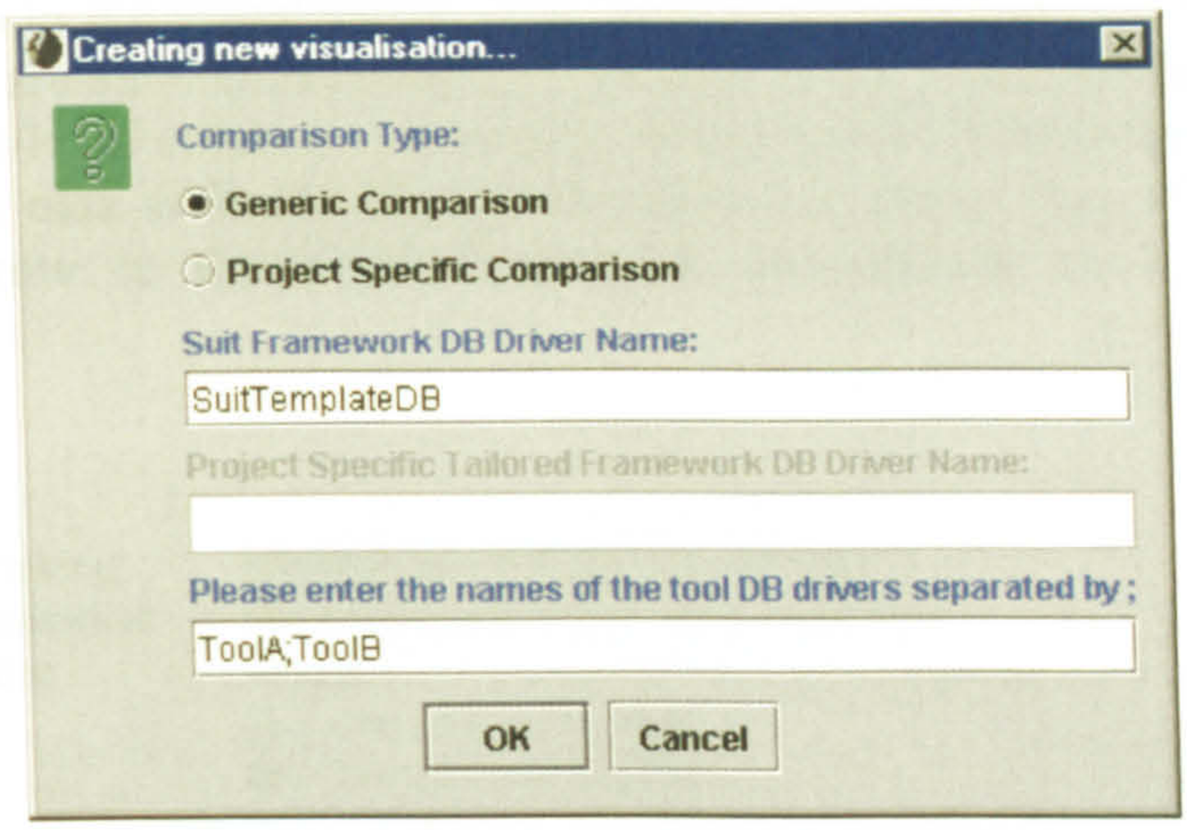


Figure 3 - *entering comparison details*

4.2 Performing A Project-Specific Comparison

1 launch SUIT;

2 select `New Visualisation...` from the Visualisation menu – see Figure 2 above;

3 select to perform a `Project-Specific Comparison`, and provide: the name of the **ODBC driver** for the project-specific ideal tool profile database; and the names of the **ODBC drivers** for the tool databases as shown in Figure 4 - and select '`OK`';

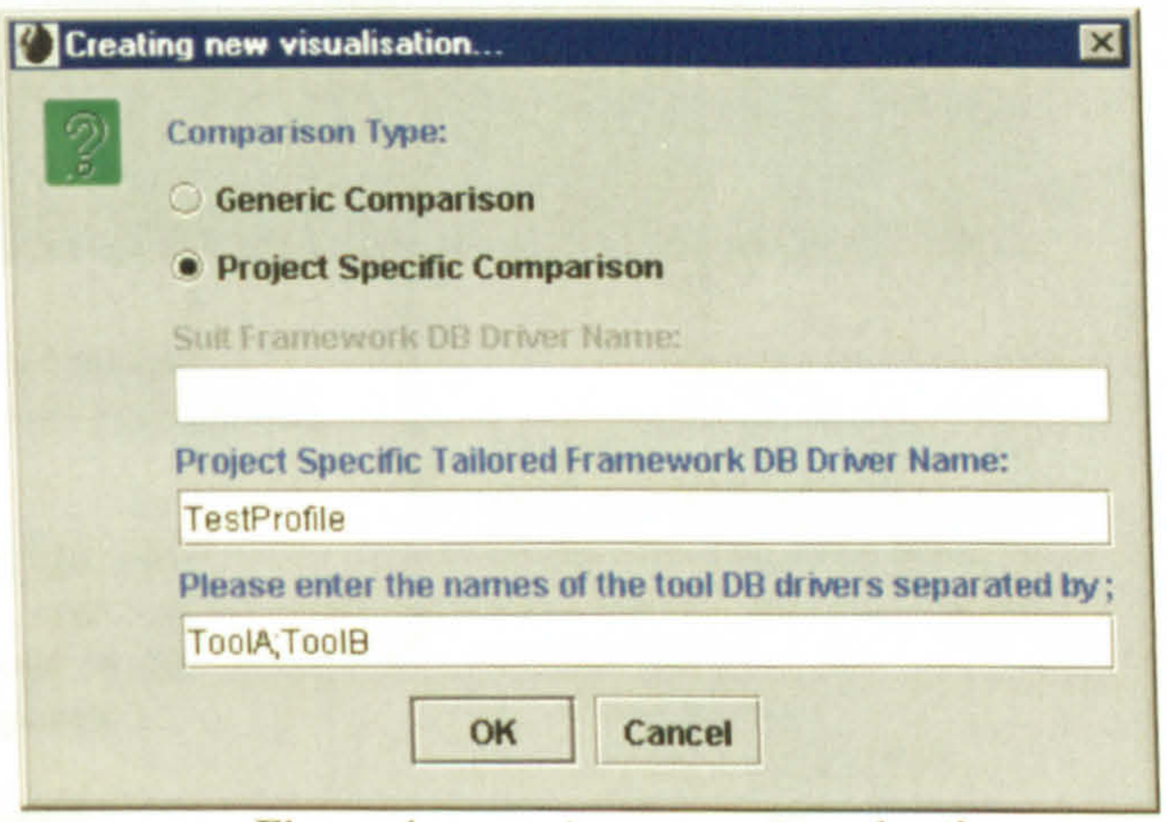


Figure 4 – *entering comparison details*

4 once SUIT has completed the generation of the data visualisation based on the information specified in step 4, you will be ready to begin analysis of your data.

5 General Introduction to the SUIT Data Visualisation

SUIT uses a combination of pattern matching and colour-coding to represent, and allow analysis of, data. Figure 5 shows a visualisation which has been created for a generic comparison of two tools – Tool A and Tool B. The diagram has been annotated to highlight the main features of a SUIT visualisation. The means by which some of the data can be displayed and the data representation manipulated is covered in section 7.

As can be seen from Figure 5, the SUIT data visualisation (and its environment) explicitly represents most of the data which is contained within the SUIT framework (the remaining data is also accessible on demand – see later). Each of the categories in the SUIT framework is

represented by a column in the visualisation (e.g. Input/Output Devices). The components within each of these categories are also represented by a column in the visualisation, and can be examined by expanding the parent-level column. Similarly, detail can be hidden by contracting a column, and thus visualising the data at the level of abstraction one above that which was selected. In essence, therefore, a row in the SUIT framework corresponds to a column in the SUIT visualisation.

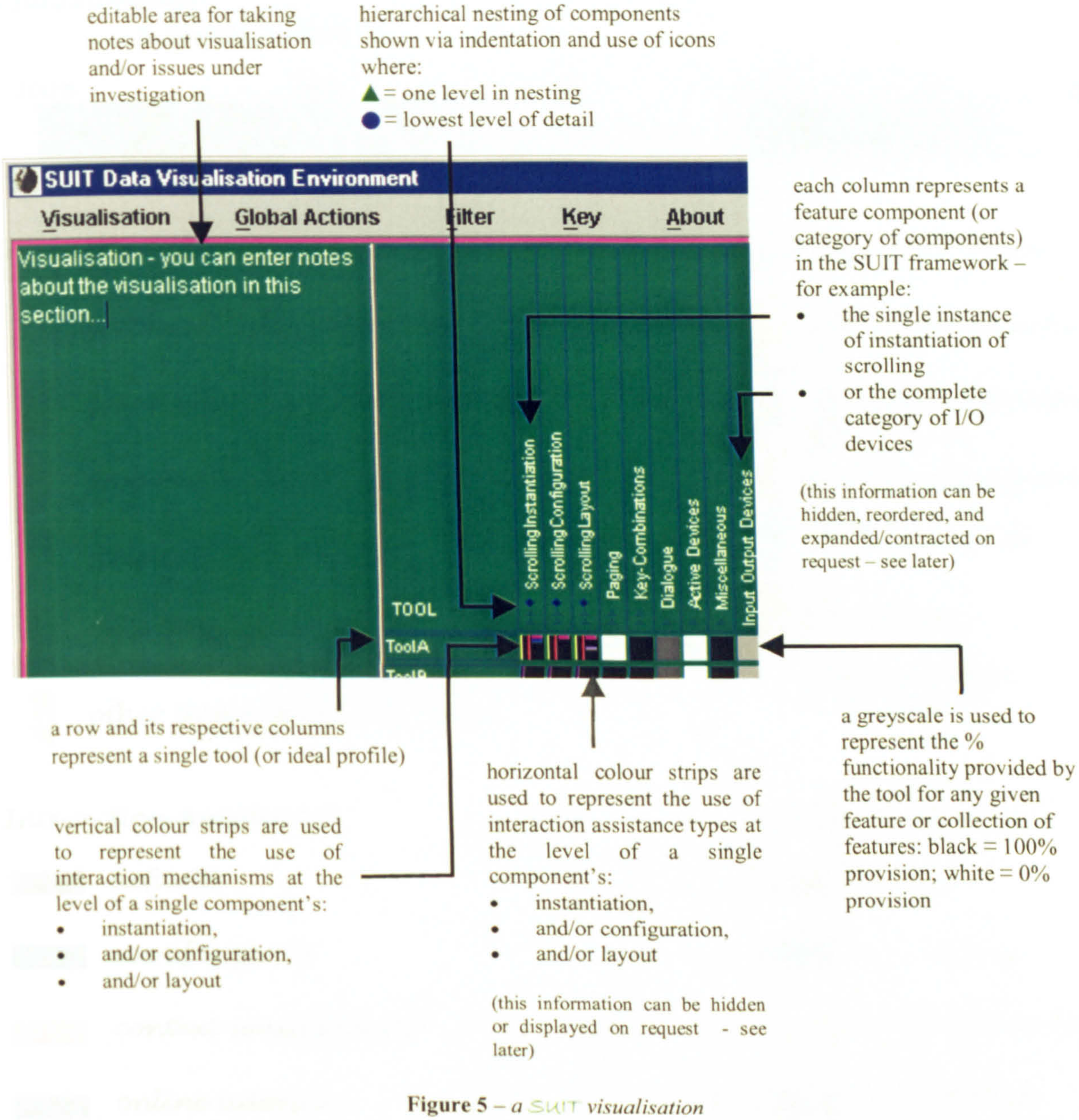


Figure 5 – *a SUIT visualisation*

Columns can be re-ordered via drag-and-drop to allow selective juxtapositioning of data during analysis. Columns can be hidden (or excluded) from the visualisation to eliminate selected data and thus narrow the field of focus. At an instance level (i.e. the instantiation, configuration, or layout of a single component) the data about the interaction mechanisms used, and assistance types provided, can be viewed. If a feature is not provided by a given tool, these cells will appear blank in the visualisation at this level.

The following page shows the key for the SUIT visualisation. The "tear-off" section can be extracted from this manual and attached to the side of your monitor for ease of reference if this proves useful.

6 Key to SUIT Data Visualisation

The following represents the colour-coding which is used within the SUIT data visualisation environment.

% functionality

% functionality

100% 50% 0%

Interaction mechanisms:

- graphical manipulation
- graphical programming
- programming language
- scripting language
- form-filling
- other interaction mechanism

Interaction Assistance:

- defaults
- use of wizards
- context-sensitive help
- online tutorials
- other interaction assistance

SUIT Key

100% 50% 0%

Interaction mechanisms:

- graphical manipulation
- graphical programming language
- programming language
- scripting language
- form-filling
- other interaction mechanism

Interaction Assistance:

- defaults
- use of wizards
- context-sensitive help
- online tutorials
- other interaction assistance

7 How do I...?

This section demonstrates how to use the SUIT visualisation environment via the use of a selection of scenarios. Given that the manner in which individuals compare data is relatively unpredictable, and a matter of personal choice or technique, it is not feasible to cover every situation in which a user of SUIT might find themselves. Instead, a representative set of scenarios will be discussed in order to demonstrate the *kind of* facilities which are available to you within SUIT. It is hoped that these scenarios will prompt you to further investigate how best to utilise SUIT for your own given data comparison/analysis needs or situation.

Each scenario is identified by a question – for example, "How do I eliminate null data from consideration?" – and is described briefly, after which *a possible path* to answering the question is provided. Should you have any queries regarding the use of SUIT which are not adequately dealt with in this user manual, please contact the author who will be happy to assist.

scenario 1: How do I record my comparison efforts so that I can refer back to them in the future ?

Scenario: *I have used the various facilities available to interrogate my visualisation, and want to record its status, together with the intention behind the analysis, and my conclusion, so that I can refer back to them when making my final decision, and writing my evaluation report.*

Solution:

First record the state of the visualisation by taking a snapshot of the visualisation. To do this, choose the [*Menu*]`Visualisation`::[*Menu Item*]`Take snapshot...` option, as demonstrated in Figure 6. A snapshot is an independent copy of an original visualisation, or previous snapshot, which can be further manipulated independently of its source.

Figure 6 – *taking a snapshot of visualisation state*

Then record comments regarding your new snapshot using the notepad to the left of the visualisation (simply type into the notepad section to the immediate left of the data visualisation snapshot) – see Figure 7. Comments can be recorded independently for each active visualisation or snapshot.

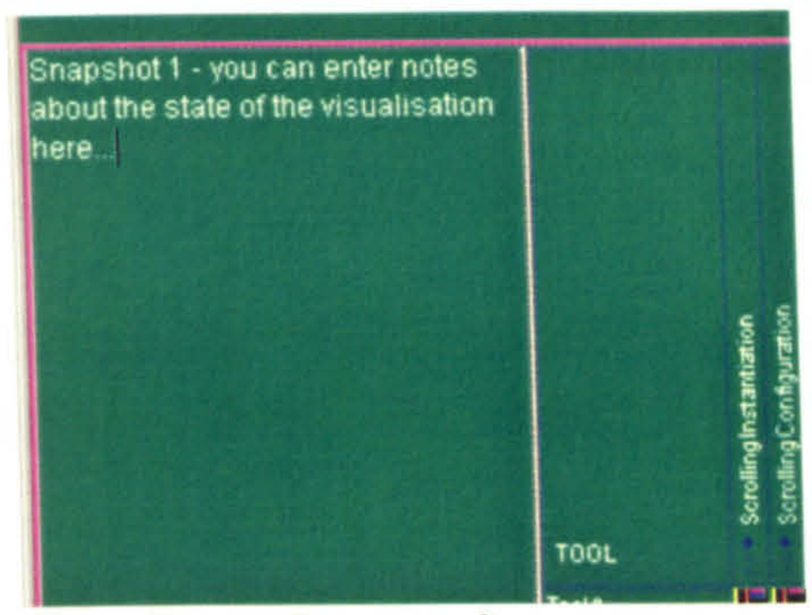


Figure 7 – *recording comments*

When recording comments regarding a snapshot, you might need to remind yourself of any filters you have applied to the data during the course of its manipulation. To do this, access the filter history by selecting the `[Menu]Filter::[Menu Item]Filter History...` option as shown in Figure 8.

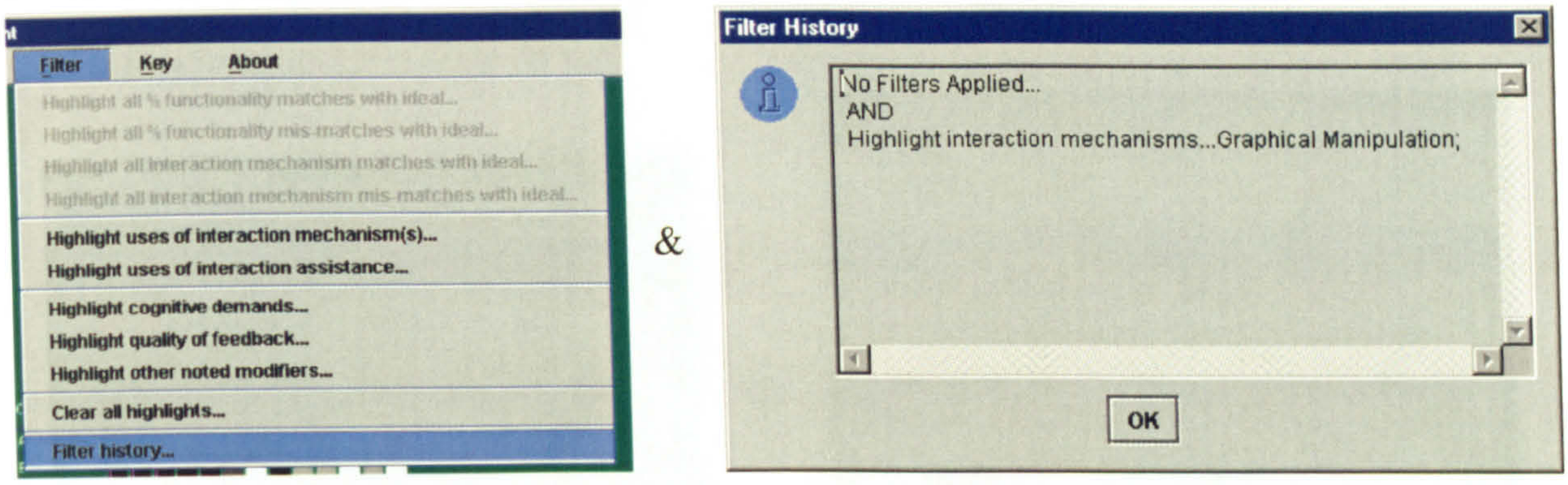


Figure 8 – *accessing the filter history for a given data visualisation*

To make a permanent record of your analysis, you can either save the file under the existing name *or* save it under a new name - either select the `[Menu]Visualisation::[Menu Item]save visualisation...`option *or* the `[Menu]Visualisation::[Menu Item]Save as...` option, both of which are shown in Figure 9. Whether you choose to save under the existing file name or as a new file may depend on the strategy of recording you have adopted. For instance, you may decide to generate a new file for each issue investigated *or* you may decide to continually add snapshots to a single evaluation file. Whichever strategy you adopt, you should ensure that files are saved within the **`C:\SUIT`** directory (or equivalent) and have the **`.suit`** extension. Saved visualisations can be re-opened and manipulated further at any stage.

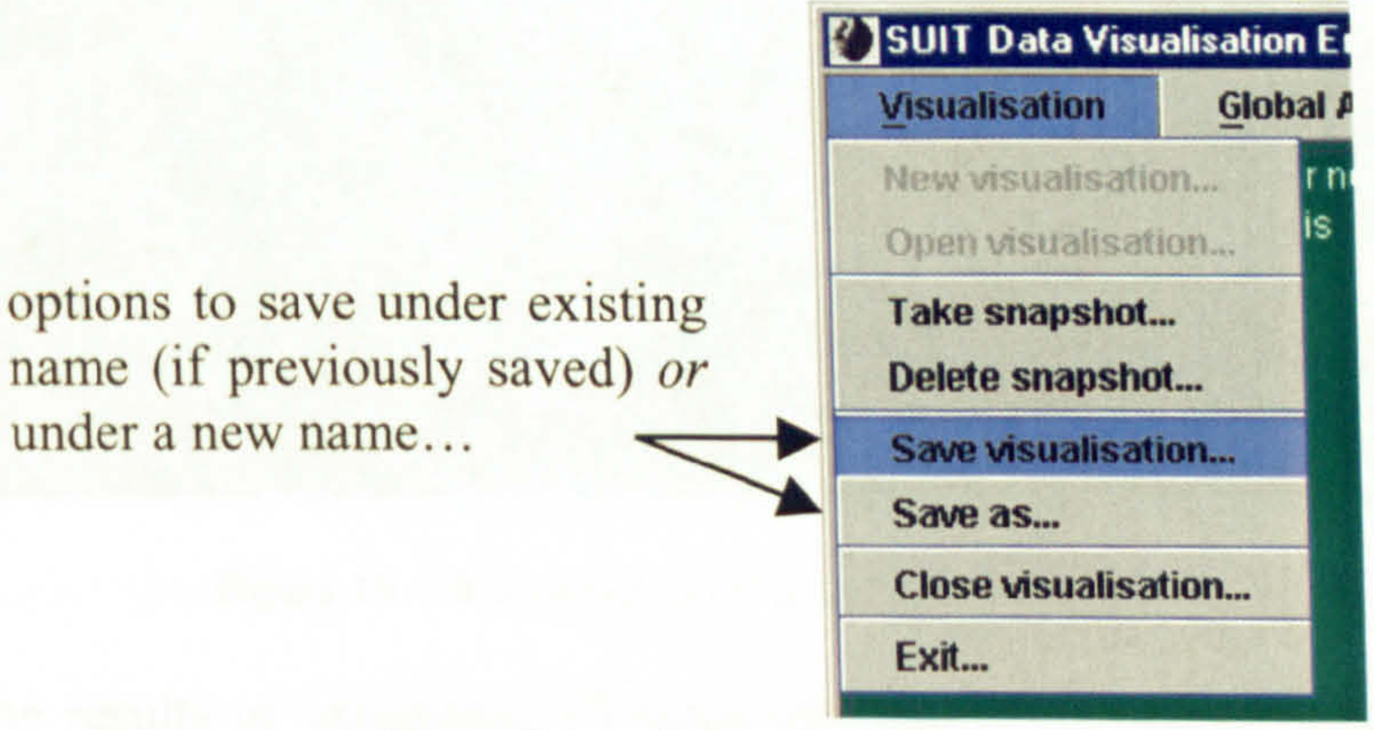


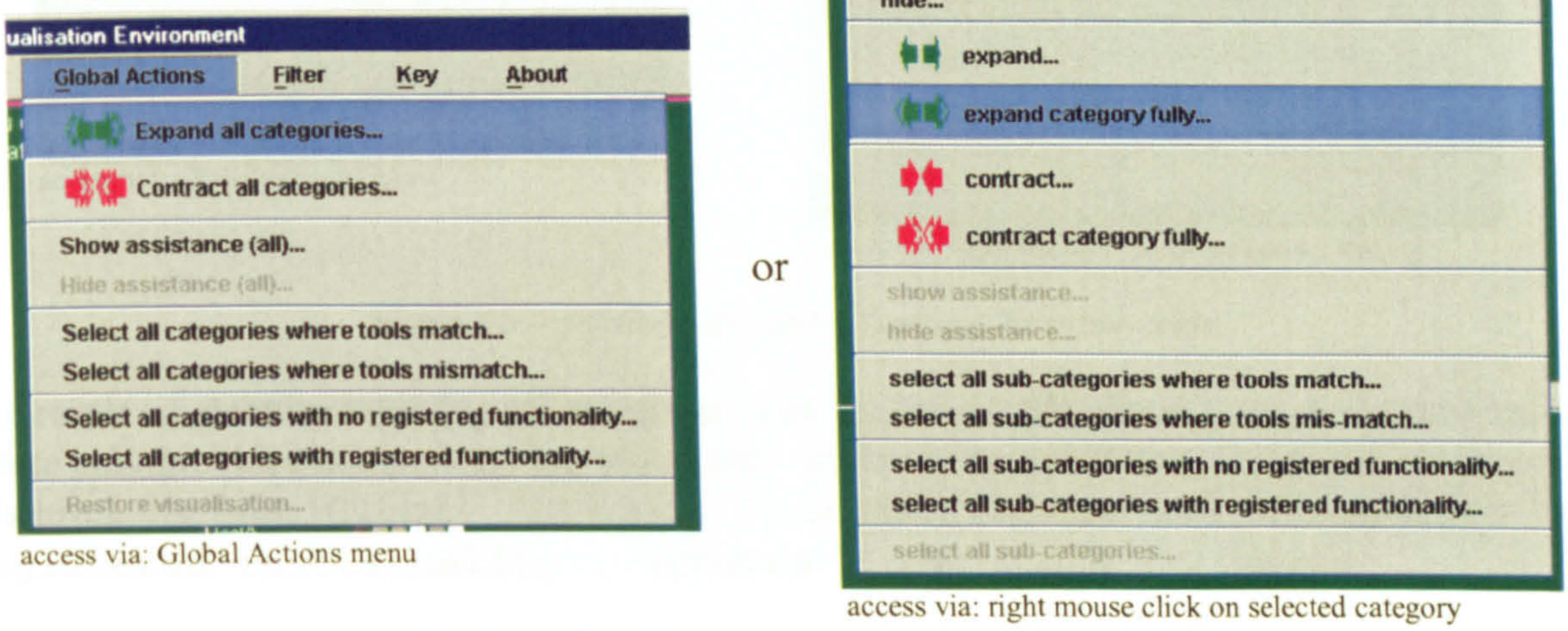
Figure 9 – *saving a visualisation*

scenario 2: How do I compare features at the level of interaction mechanisms - specifically those features which are provided by at least one tool ?

Scenario: *I want to compare the interaction mechanisms used across all features which are provided by at least one of my tools.*

Solution:

Depending on the extent to which you want to examine the data – i.e. the number of components you wish to view at once – you can choose to expand, down to the level of the interaction mechanisms used, either a single category *or* the entire data set. When generated, a visualisation shows data at the highest level of abstraction, which is at the highest category level. In order to reduce the abstraction, choose the most appropriate expansion by selecting *either* the `[Menu]Global Actions::[Menu Item]Expand all categories...` option *or* `right-click` on a category (column header), and select the `[Menu Item]expand category fully...` option as shown in Figure 10.



access via: Global Actions menu

access via: right mouse click on selected category

Figure 10 – *expansion of categories to display interaction mechanism data*

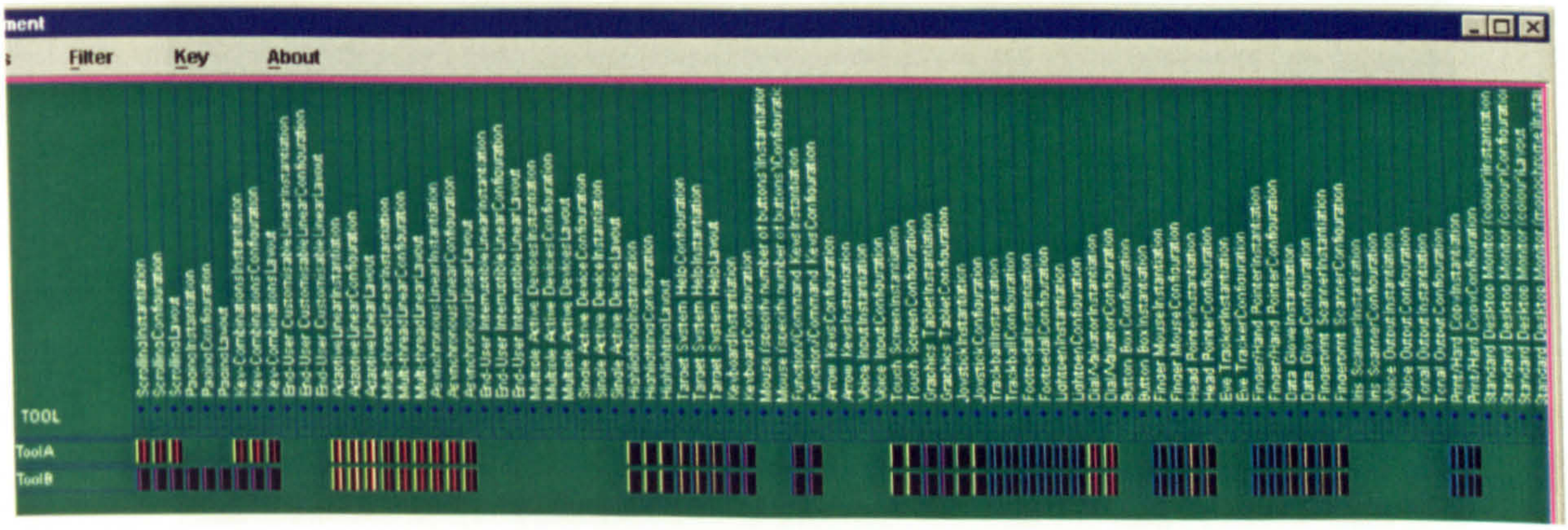


Figure 11 – *data visualised at lowest level of detail*

Figure 11 shows the results of expanding all categories within the visualisation. As explained previously, the data for interaction mechanisms is represented via vertical colour-coded strips, per component. Should you require reminders about the colour-coding, refer to the online

visualisation key (accessed via the [*Menu*]Key::[*Menu Item*]Show key... option), or to section 6 of this manual.

As can be seen from Figure 11, data is visualised both for features which are provided across tools, and which are absent across tools. To isolate, and therefore focus on, those components where at least one tool provides the given component, you can choose to view only those components that *are* provided (by at least one tool). Depending on your previous choice at the expansion stage, you can choose to apply the selection to the entire data set – in which case, choose the [*Menu*]Global Actions::[*Menu Item*]Select all categories with registered functionality... option – or to a specific category – in which case right-click on the category (column header), and select the [*Menu Item*]select all sub-categories with registered functionality... option. Both options are demonstrated in Figure 12.

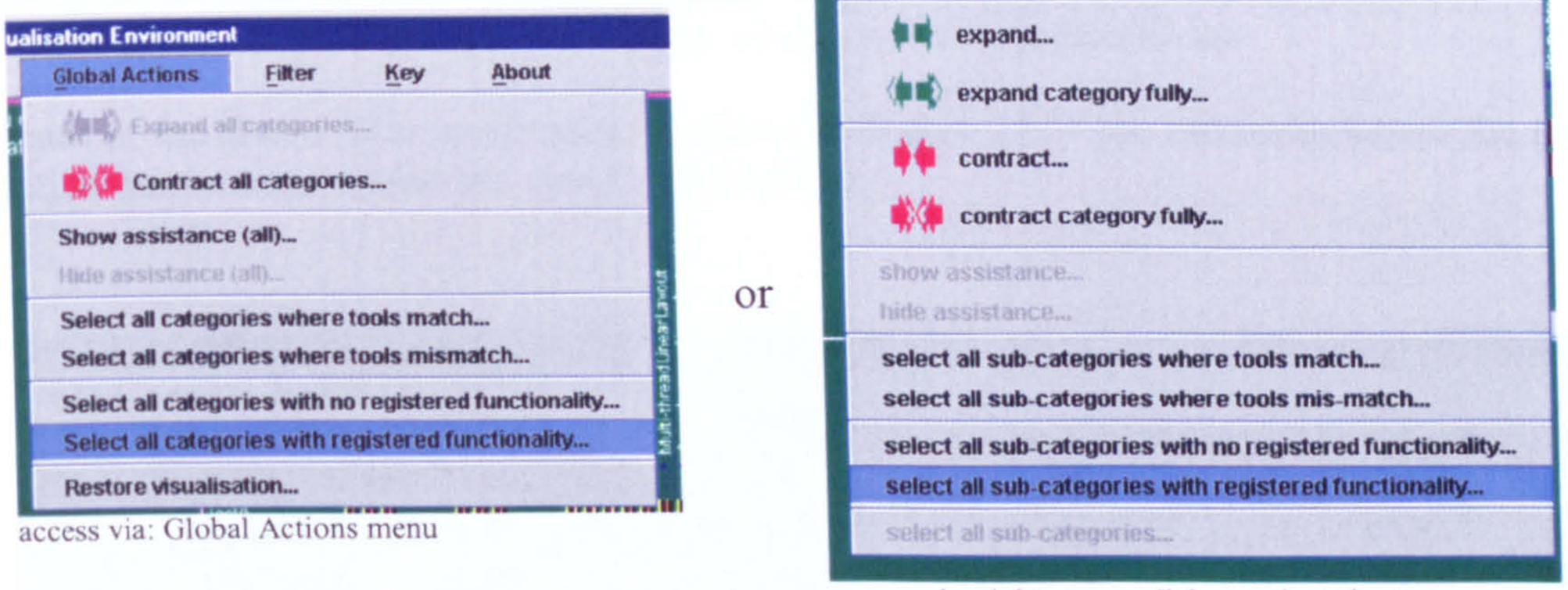


Figure 12 – *selecting only existing functionality across tools*

The result of globally selecting all categories with registered functionality is shown in Figure 13, where only provided components are visualised. It should be noted that, for global examination at this level, the [*Menu*]Global Actions::[*Menu Item*] Select all categories with registered functionality... option can be selected in the first instance.

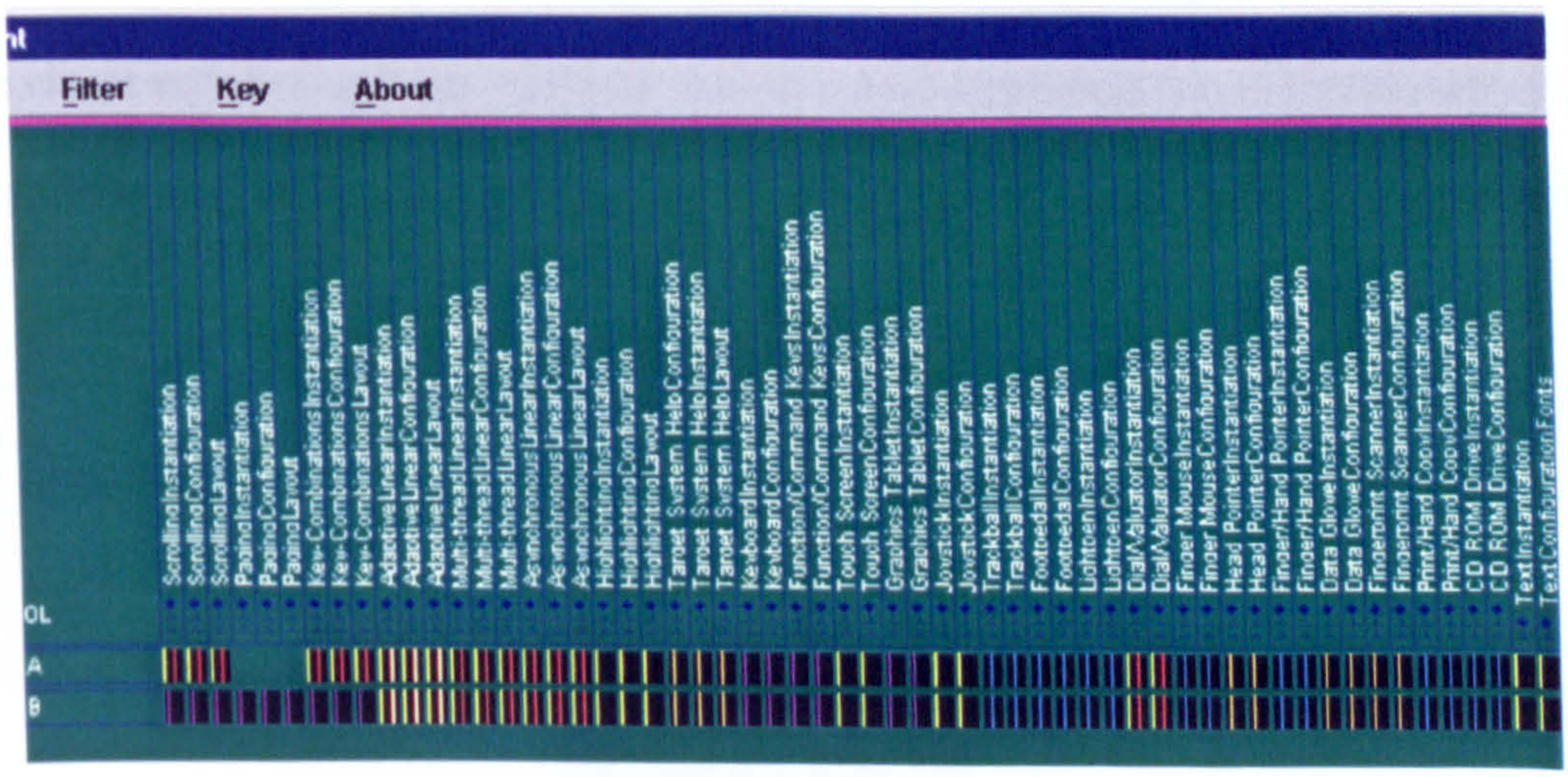


Figure 13 – *visualisation showing interaction mechanisms for only existing components*

At this point, you might want to highlight the use of specific interaction mechanisms – perhaps ones which are either suitable, or unsuitable, within a given context. To do this, select the [*Menu*]Filter::[*Menu Item*]Highlight uses of interaction mechanism(s)...

option and, from the list of options that will be presented, select the mechanism(s) you wish to highlight - as demonstrated in Figure 14.

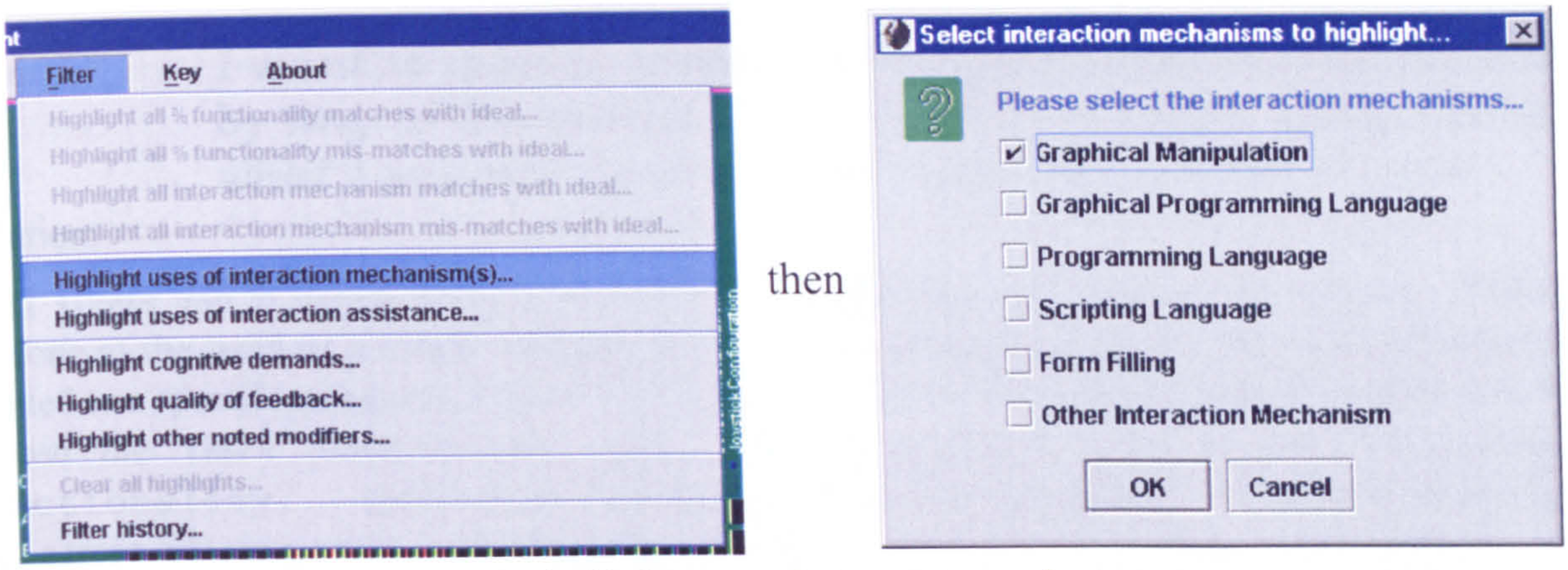


Figure 14 – *highlighting specific interaction mechanism use*

The result of the above filter application is shown in Figure 15 – the instances where the specified interaction mechanism is used are clearly highlighted.

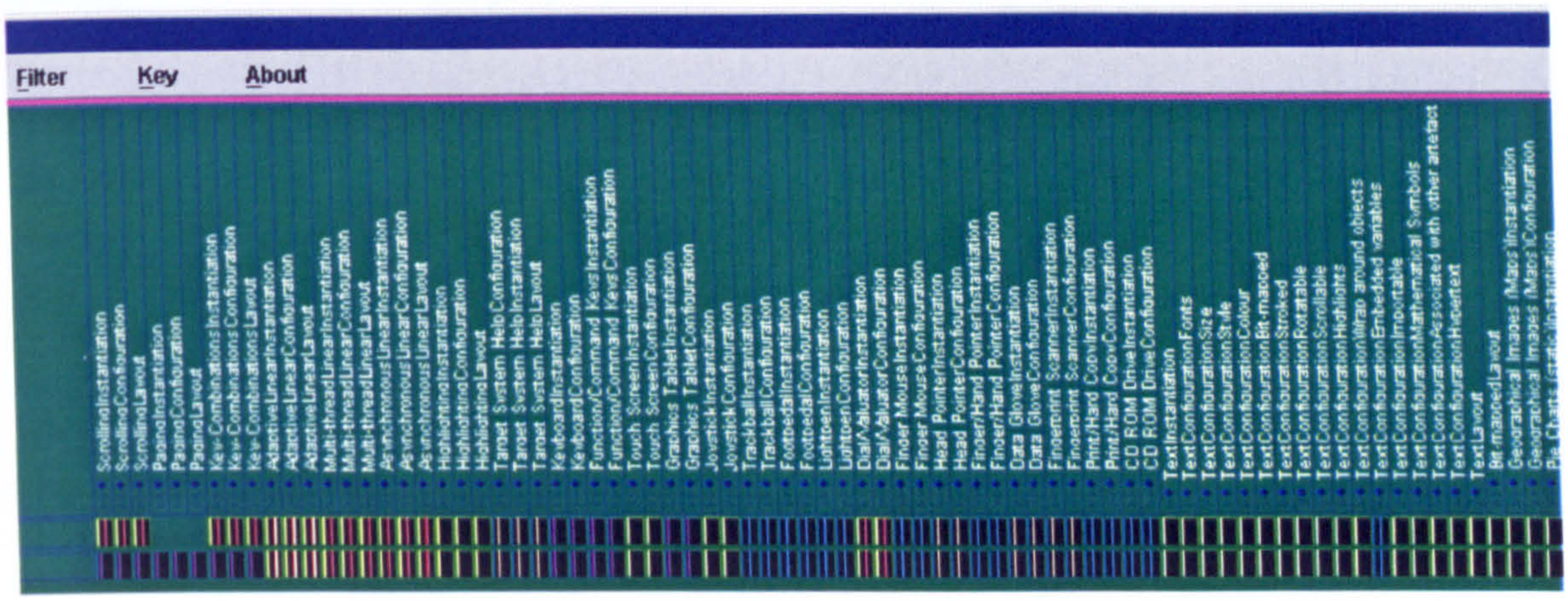


Figure 15 – *highlighted use of specified interaction mechanism*

In order to closer examine, or draw closer attention to, any particular interaction mechanism use for a specific component, simply stretch out the component's column in the visualisation. This is done by applying a `click-and-drag-right` action to the right hand edge of the component's column header – the effect of which is shown in Figure 16.

Figure 16 – *magnifying the visualisation for a selected component*

scenario 3: How do I quickly identify all the instances where listed functionality is missing from all tools ?

Scenario: *I want to quickly identify which facilities are not provided by any of the evaluated tools so that I can easily identify what I will not be able to do regardless of tool choice.*

Solution:

SUIT allows you to very quickly identify those facilities which all tools fail to support. This can be done at the level of a single category or across all categories. In the case of the former (i.e. applied to a specific category), `right-click` on the required category (column header), and then choose the `[Menu Item]select all sub-categories with no registered functionality...` option from the pop-up menu which will appear. In the case of the latter (i.e. across all categories), select the `[Menu]Global Actions::[Menu Item]Select all categories with no registered functionality...` option. Both options are shown in Figure 17, and the result of applying the selection globally is shown in Figure 18.

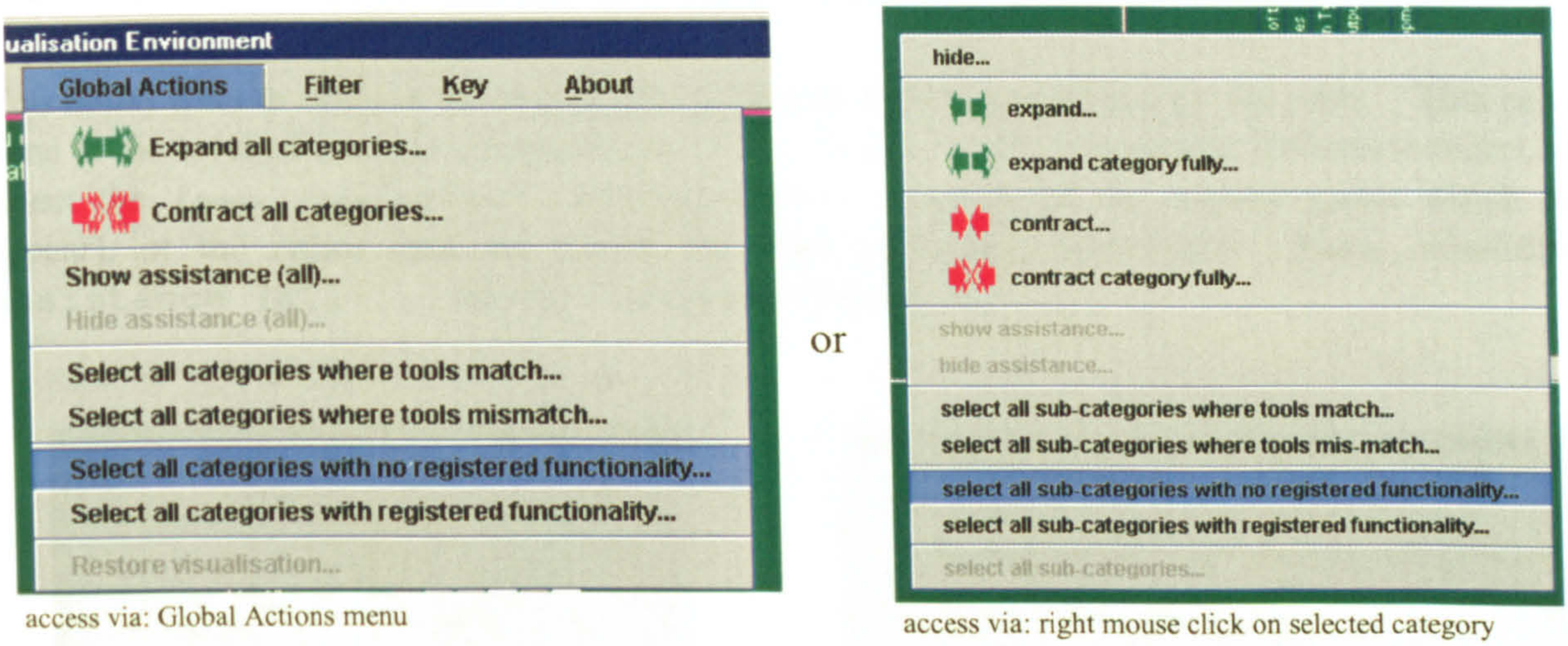


Figure 17 – *identifying functionality which is not provided by any tool*

Figure 18 – *visualisation of all components which are not supported by any tool*

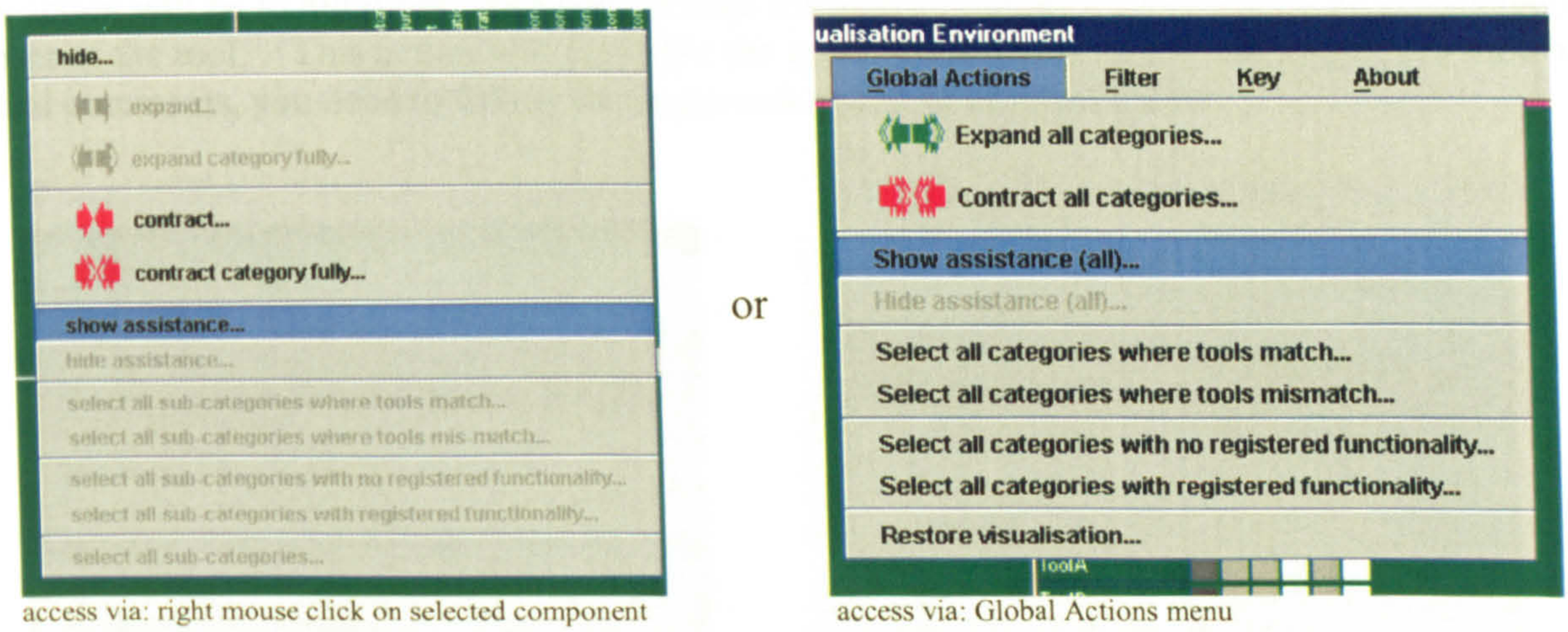
scenario 4: How can I get extra information to help decide between two tools which are identical in terms of the functionality provided and interaction mechanisms used ?

Scenario: *I have observed the functionality provided and interaction mechanisms used for the tools under consideration and have realised that they are identical in these respects. I therefore need assistance from additional information about the tools in order to make a final decision about the tools.*

Solution:

The SUIT framework records information about the following (on a per-component basis): the interaction assistance provided; the cognitive demands placed on the developer during the course of utilising a specific component of functionality; the quality of feedback returned to the developer by a tool when attempting to utilise some component of functionality; and any other modifier comments recorded by the evaluator of the tools. Each of these can be accessed during the course of data comparison as, and when, required.

You might wish to observe the interaction assistance which is provided by the tools. This can be done at the level of a single component (`right-click` on the component (column header), and select the [*Menu Item*]`select assistance...` option in the pop-up menu which will appear), or the entire data set (select the [*Menu*]`Global Actions::`[*Menu Item*]`Show assistance (all)...` option) – as shown in Figure 19.



access via: right mouse click on selected component

access via: Global Actions menu

Figure 19 – *request to visualise interaction assistance data*

Within the extent of the request, the appropriate columns will be changed to show interaction assistance – where provided – as shown in Figure 20. Interaction assistance is visualised via horizontal colour-coded lines and so, after initiating a request for interaction assistance information, if no such lines appear for any given column, this indicates that there is no assistance provided for the corresponding component.

Figure 20 – *visualisation of interaction assistance*

Aside from interaction assistance information, you may wish to consider the auxiliary information which may have been recorded for individual components. Such information is recorded under the headings: cognitive demands; quality of feedback; and other modifier comments. Each information instance is recorded as a modifier (+/0/-) and an English language comment. Hence, you can search for comments in any of the three categories (or a combination thereof) based on their modifier. To do this, you select the appropriate comment type, and then identify the modifier upon which you wish the selection to take place.

Figure 21 demonstrates this process for comments regarding cognitive demands for which a positive modifier has been recorded. As can be seen, you need to select the `[Menu]Filter::[Menu Item]Highlight cognitive demands...` option, and then choose the required modifier from the list of options which will be presented. Hence, the result of this action will be to highlight those components for which cognitive demands are an advantageous aspect of the tool. This action will *highlight* the appropriate components, but in order to view the actual comments, you need to follow the suggested action as outlined below.

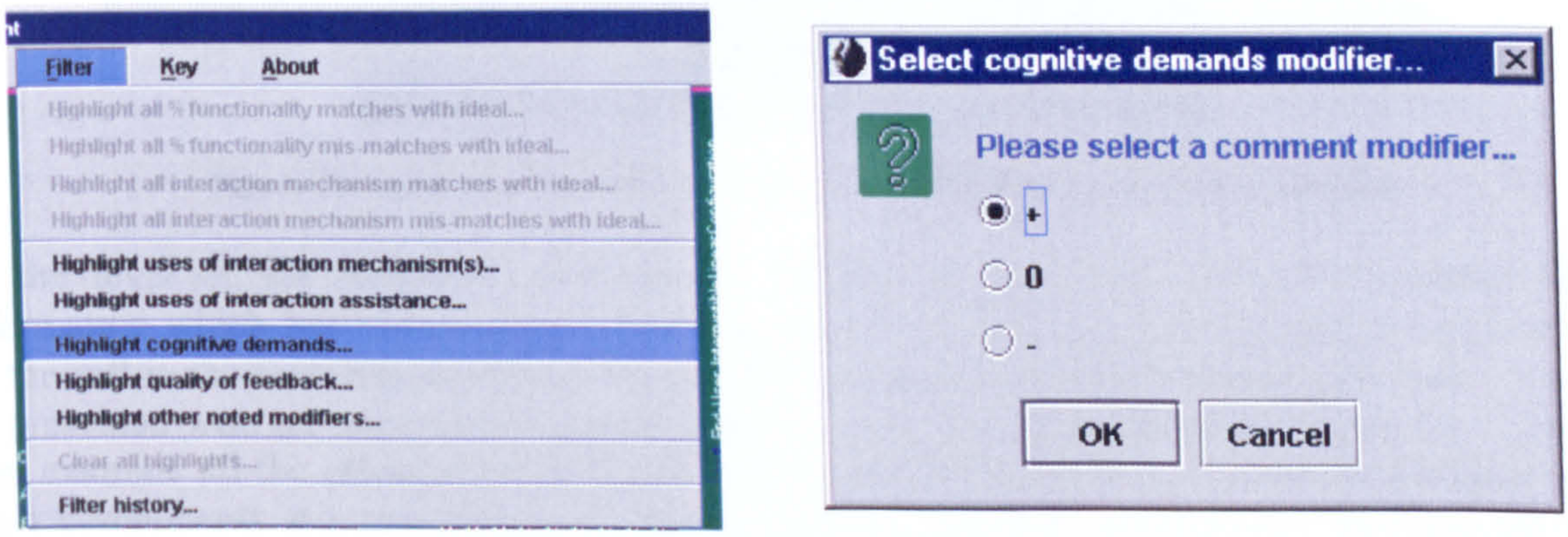


Figure 21 - *comment selection based on recorded modifier*

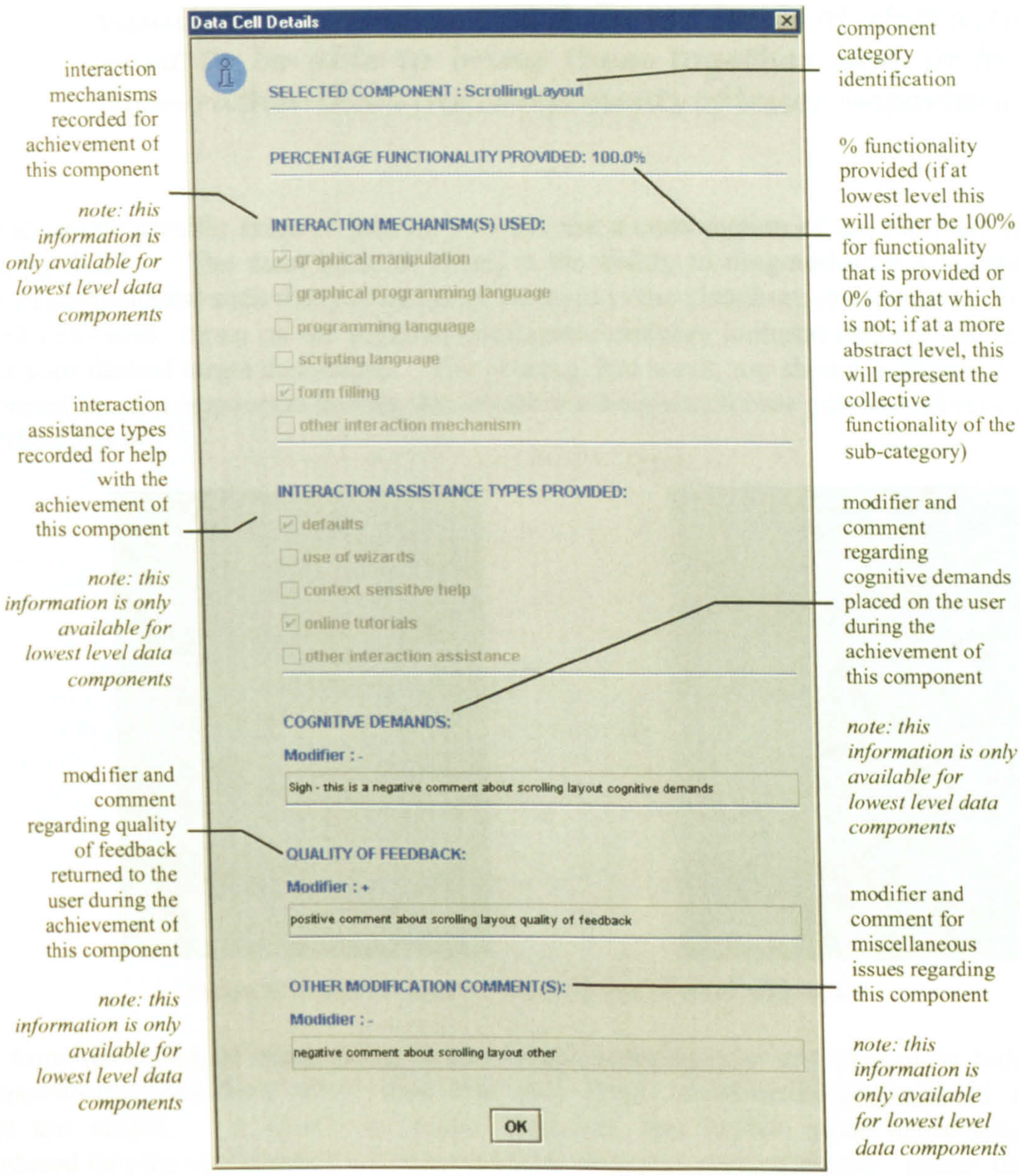


Figure 22 – *annotated example of an information display for a selection component*

At the level of the individual components comprising each tool, you can examine all the information which has been recorded in the corresponding database. To do this, `right-click` on the **cell** in the visualisation which represents the component for the appropriate tool. You will be presented with an information display similar to that which is shown in Figure 22. You can then examine all the information in detail for the selected component in order to compare it with other components at a non-visual, or abstract, level. In particular, you can examine the actual comments which have been recorded.

scenario 5: How can I isolate, or bring together, the components I am most interested in so that I can more easily focus on them or compare them across the tools ?

Scenario: *I have identified a handful of components which are critical to my adoption of a tool, and therefore most significant to my evaluation process. Unfortunately, they*

are physically distant from one another in the visualisation, and are at different levels of abstraction. I want to be able to bring them together and to focus on them rather than the components of lesser importance.

Solution:

In order to isolate specific areas of interest you can use a combination of the facilities which are provided by SUIT. The most basic of these, is the ability to drag-and-drop a component (or category of components) such that you can relocate them in the visualisation sequence. To do this, simply `click-and-drag` on the required component/category (column header) and release the mouse at your desired target destination. The process, and result, are shown in Figure 23. It can be performed for all components barring the lowest level (in which case you should relocate at the level of their parent).

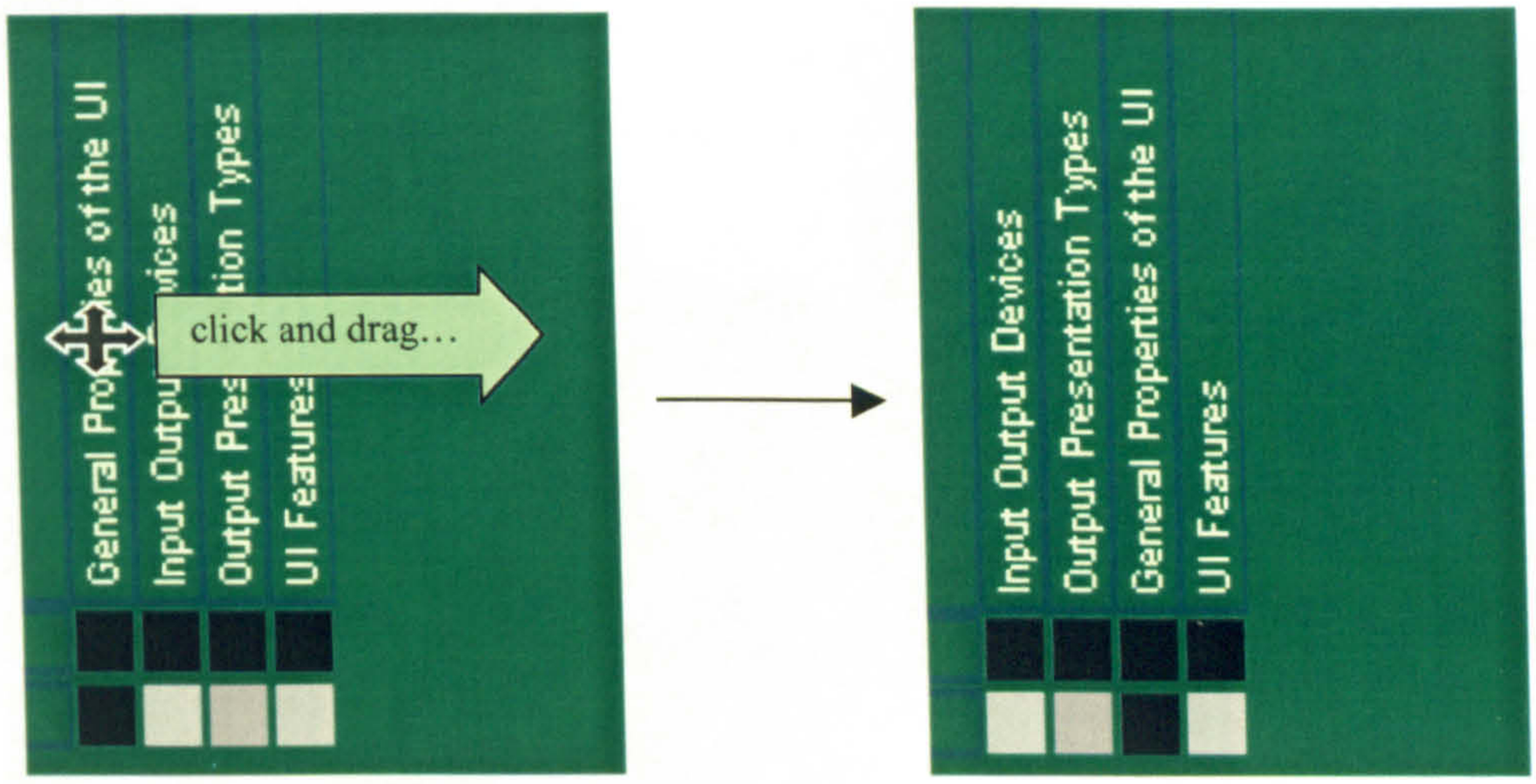
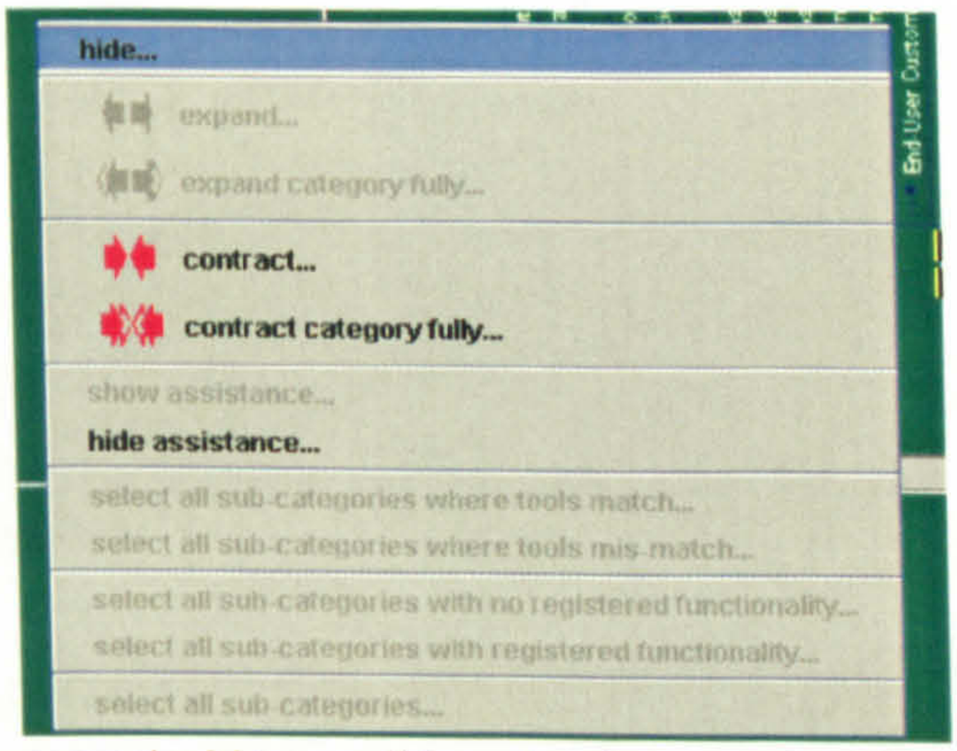


Figure 23 – *action to move a component and the result of the action*

Aside from relocation of components in the visual ordering, you can choose to hide selected components/categories from view, such that only those components/categories of immediate interest are visible. It should be noted, however, that hidden components/categories are reintroduced into the visualisation when component categories are contracted. Hence the effect of hiding a selected component/category is temporary. To hide a component/category, `right-click` on the component/category (column header) and select the `[`*`Menu Item`*`]hide...` option from the pop-up menu which will appear. This is shown in Figure 24.



access via: right mouse click on selected component/category

Figure 24 – *hiding a selected component/category*

Levels of detail can also be hidden via the various contraction options which are available at the level of a single component, a category, or the entire data set. To contract all categories, select the [*Menu*]Global Actions::[*Menu Item*]Contract all categories... option. To contract any given category, right-click on the category (column header), and select the [*Menu Item*]contract category fully... option form the pop-up menu which will appear. Finally, to contract a single component up to the level of its parent, right-click on the component (column header), and select the [*Menu Item*]contract... option from the pop-up menu which will appear. Access to these facilities is demonstrated in Figure 25. It should be noted, that a category-based contraction collapses all of the components/categories at the sibling level (and below that) of the selected category, irrespective of their physical location within the visualisation.

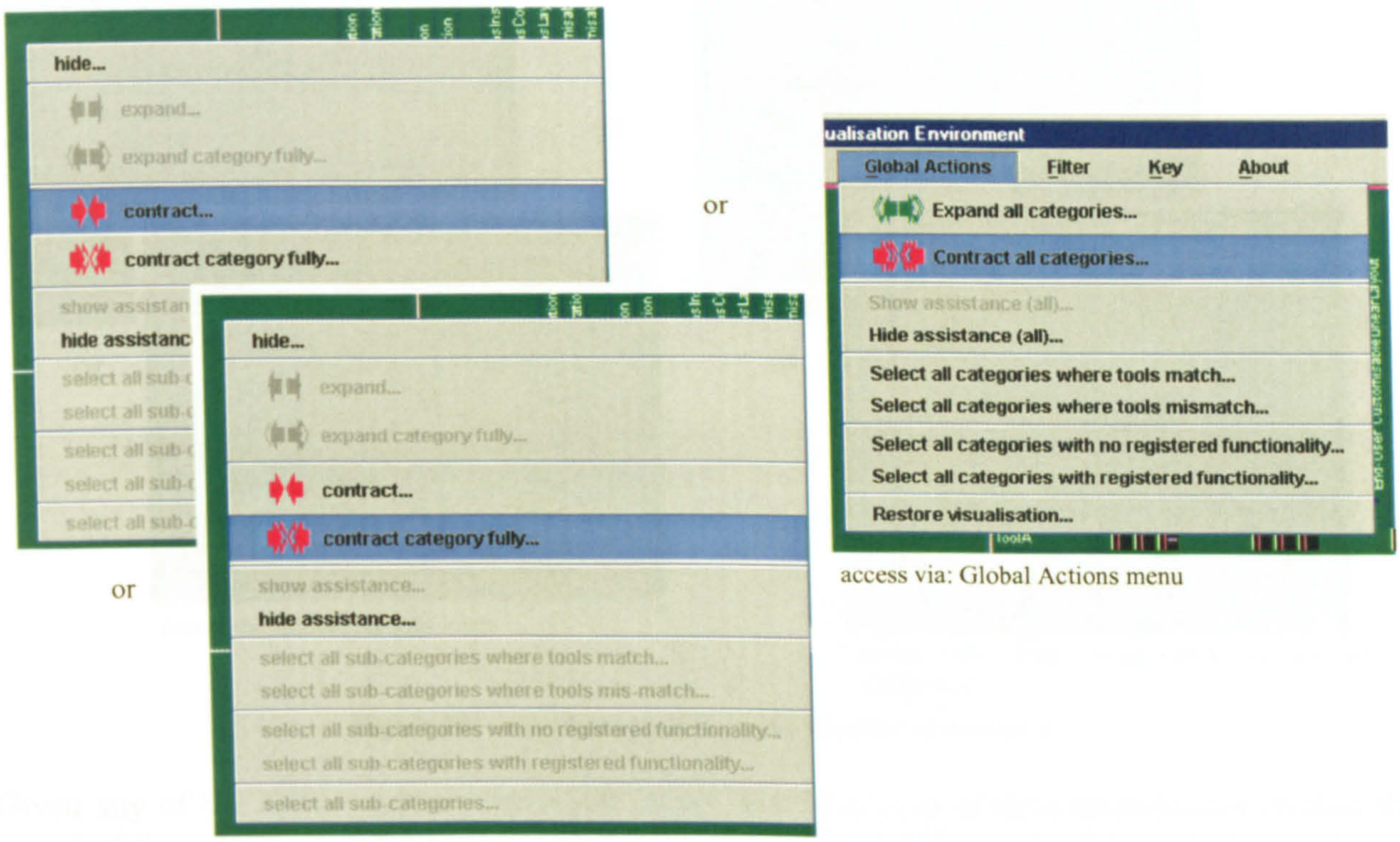


Figure 25 – *contraction of categories*

scenario 6: How can I quickly find where tools do and don't match so that I can identify interesting aspects for further investigation?

Scenario: *I have just created a new visualisation and want to quickly identify where the tools match and where they are different so that I can immediately focus on the areas of difference (since that is where potential evaluative interest lies).*

Solution:

SUIT allows you to (mutually exclusively) select either the matches or the mis-matches across tools. This can be done at the level of a specified category of components (i.e. the selected category and all its sub-categories), or across all data in the visualisation. Figure 26 demonstrates the selection process for both matches and mis-matches, for a specified category, and across all visualised functionality. As can be seen, to perform a global selection of matches (i.e. across all

components), choose the [*Menu*]Global Actions::[*Menu Item*]Select all categories where tools match... option; conversely, to select mis-matches globally, choose the [*Menu*]Global Actions::[*Menu Item*]Select all categories where tools mismatch...option. In order to select matches at the level of a specific category, right-click on the category (column header) and then choose the [*Menu Item*]select all sub-categories where tools match... option from the pop-up menu which will appear; finally, to select all mis-matches at the level of a specific category, right-click on the category (column header), and then choose the [*Menu Item*]select sub-categories where tools mis-match... option from the pop-up menu which will appear.

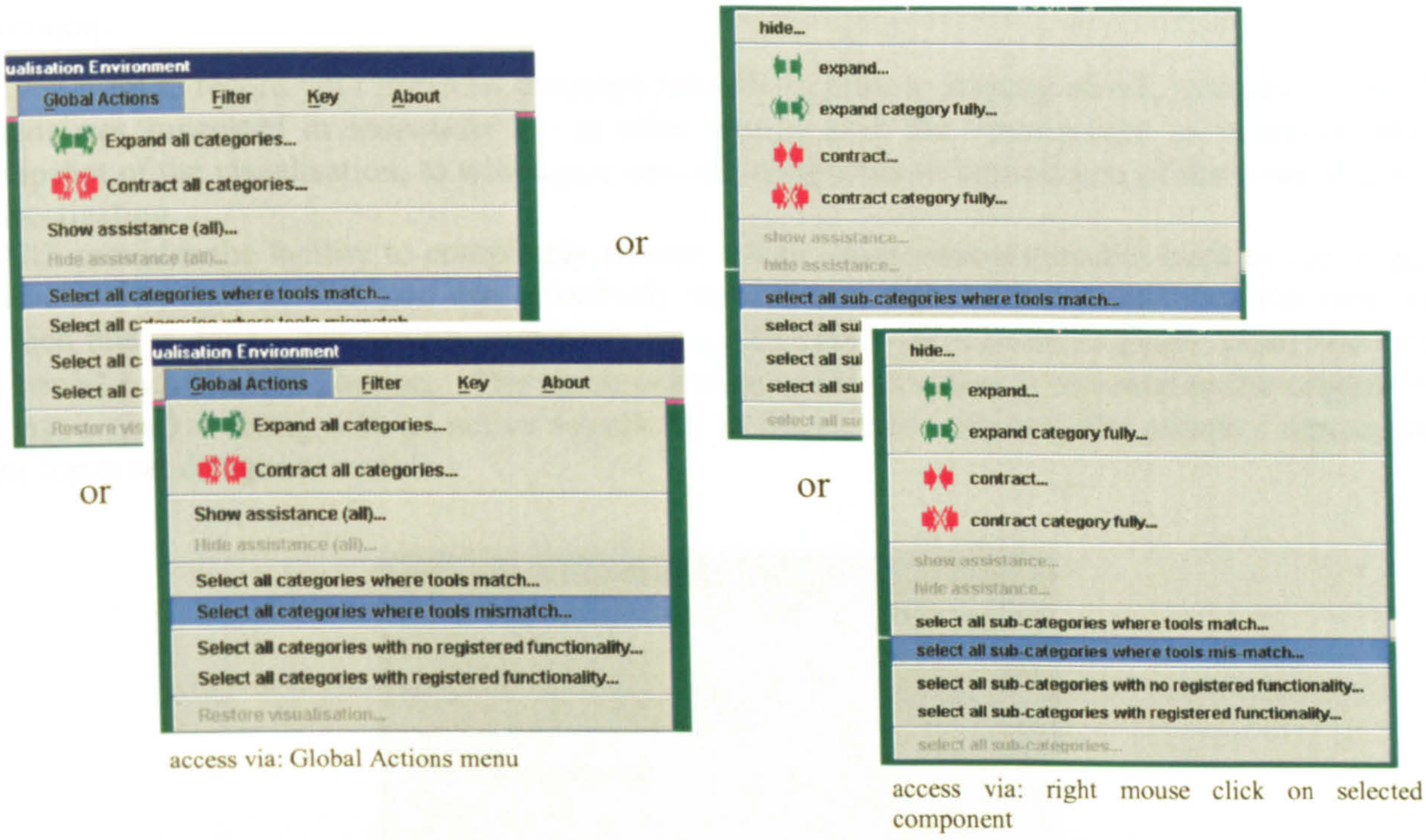


Figure 26 – *selection of matches/mis-matches across tools*

Given any of the above options, the result of the selection is to expand components (within the extent of the selection) down to the lowest level of detail, and show only those which match/mis-match. An example of a global selection of all mis-matches across two tools is shown in Figure 27. As can be seen, the handful of discrepancies between the two tools is easily identified for further investigation, and so you can quickly focus on those issues which are most likely to influence a selection.

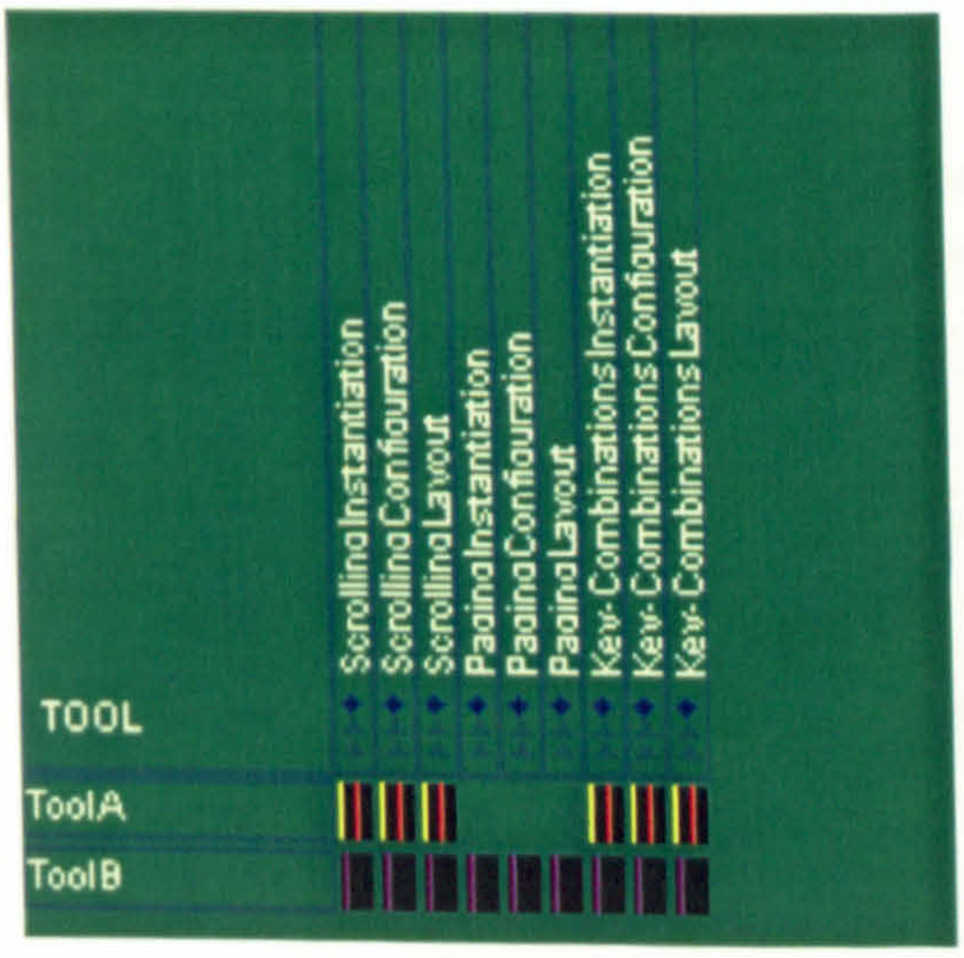


Figure 27 – *result of a global selection of mis-matches across all tools*

scenario 7: How can I wipe the slate clean and start on a new track of investigation ?

Scenario: *I have been working with my visualisation - querying and re-ordering the data in order to investigate important issues. I now want to start afresh with my data so that I can investigate alternative issues.*

Solution:

If you want to record your previous investigative efforts prior to starting afresh, you can follow the guidelines suggested in *scenario 1* – in other words, save the visualisation as is and/or take a snapshot of the visualisation, to which you can add comments to remind you of the issue you were investigating.

SUIT provides the facility to completely restore a data visualisation/snapshot back to the original visualisation state so that you can effectively start afresh. Figure 28 demonstrates the ease with which this is done – you simply select the `[Menu]Global Actions::[Menu Item]Restore visualisation...` option. The result is that your visualisation is returned to the original (on construction) ordering with no active selections, or filters, and with only the columns representing the top-level categories visible.

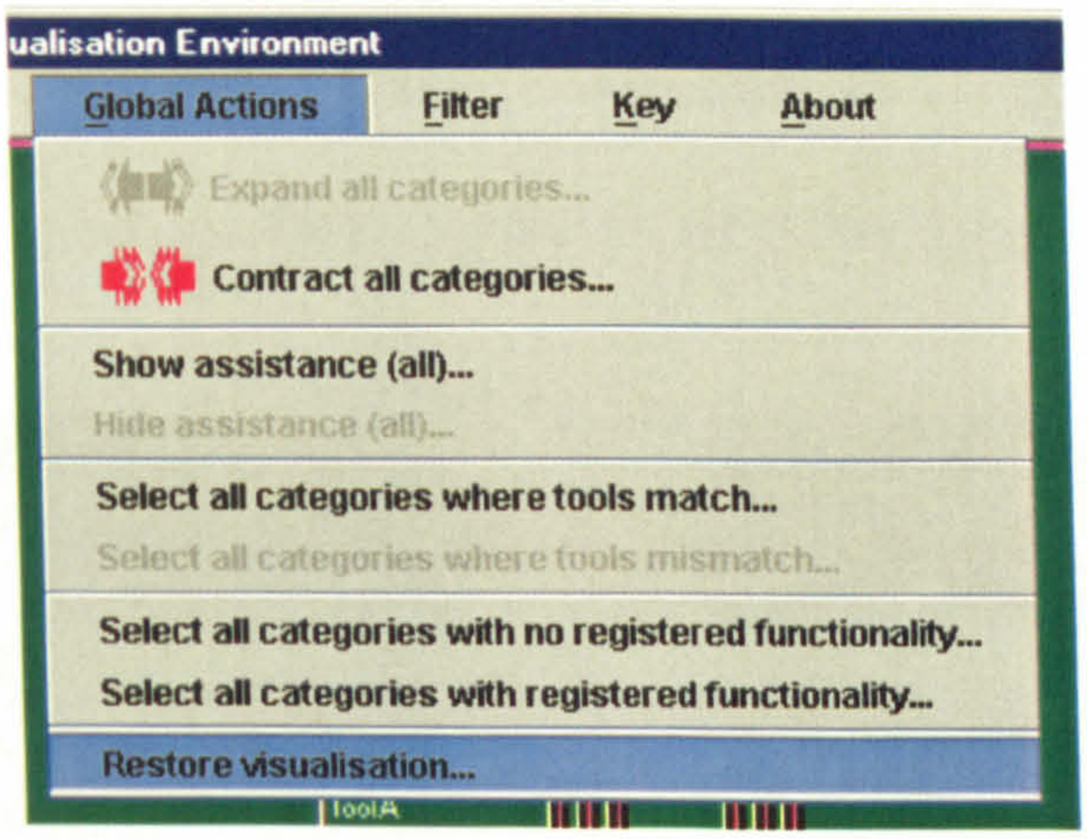


Figure 28 – *restoring a visualisation to its original state*

Alternatively, if you simply want to clear all previous queries (or filters) applied to the data, but maintain ordering and visibility settings, you can do this by selecting the `[Menu]Filter::[Menu Item]Clear all highlights...` option as demonstrated in Figure 29.

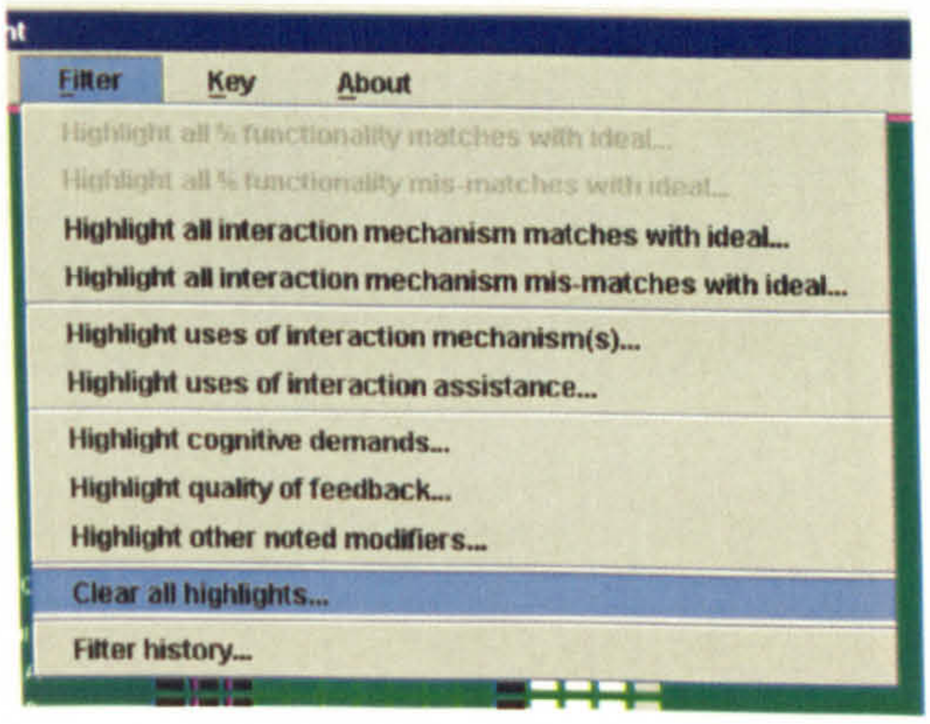


Figure 29 – *clearing highlights from previous data analysis*

scenario 8: How can I identify how closely individual tools mirror the ideal tool profile I have generated?

Scenario: *I am conducting a project-specific evaluation of tools. As such, my visualisation includes a profile of the ideal tool given my project context. I would like to quickly identify where tools mirror the ideal and where they are not a match with what I have specified as ideal.*

Solution:

As demonstrated in *scenario 6*, generic matches/mis-matches across tools can be easily identified. However, in addition to this basic matching mechanism, SUIT also provides a number of filters which can be applied to the data visualisation for project-specific evaluations. These (project-specific) filters include the facility to highlight matches/mis-matches between the tools and the ideal tool profile on the basis of % functionality provided, and interaction mechanisms used.

Suppose you are interested in first examining the basic match in terms of the % functionality provided – that is, matching those components which are provided by the tools against those that have been identified as requirements of the project. Depending on whether you prefer to highlight the matches or the mis-matches – i.e. whether you prefer to focus on what is missing or what is provided – will determine which of the following two options you choose.

To show the % functionality matches, select the [*Menu*]`Filter`::[*Menu Item*]`Highlight all % functionality matches with ideal...` option. In contrast, to show the % functionality mis-matches, select the [*Menu*]`Filter`::[*Menu Item*] `Highlight all % functionality mis-matches with ideal...`option. Both cases of filter application are shown in Figure 30.

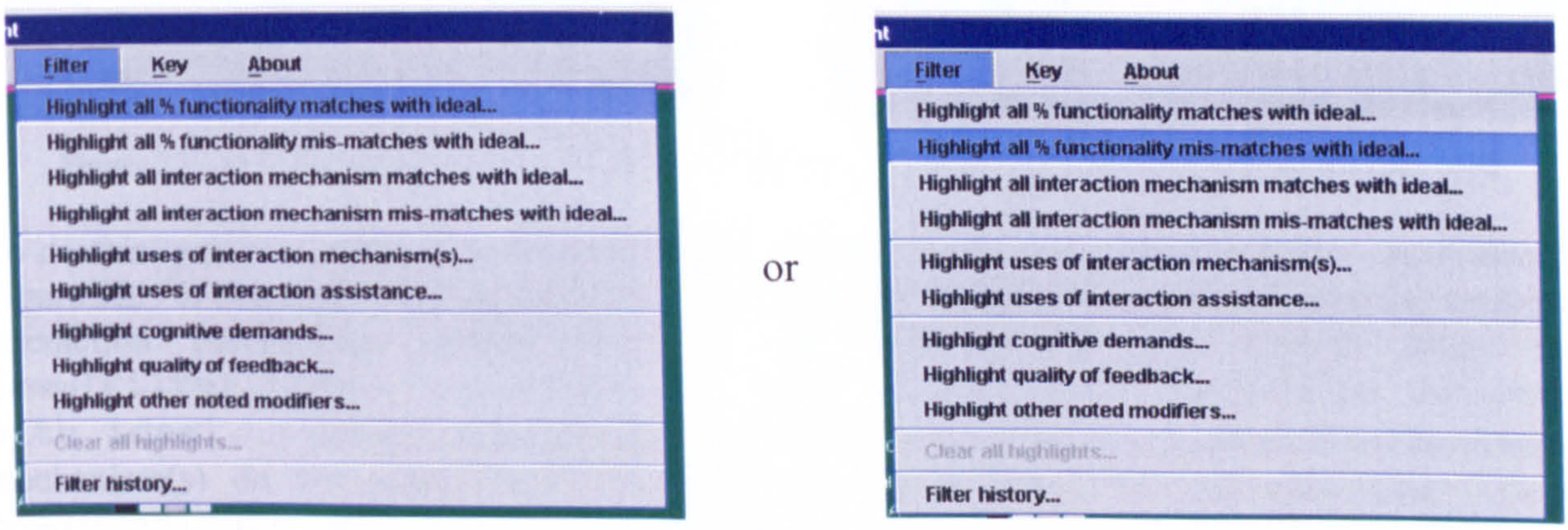


Figure 30 – *highlighting tool matches/mis-matches with ideal profile in terms of functionality provided*

Figure 31 shows the result of highlighting % functionality mis-matches with the ideal. It illustrates that components can be highlighted at the lowest level, or at any level, of abstraction – in the case of the latter, the highlight indicates the existence of a (currently contracted) low-level component which conforms to the filter. As can be seen, your attention is quickly drawn to those components (or categories thereof) for which matches are not found against the requirements of the project.

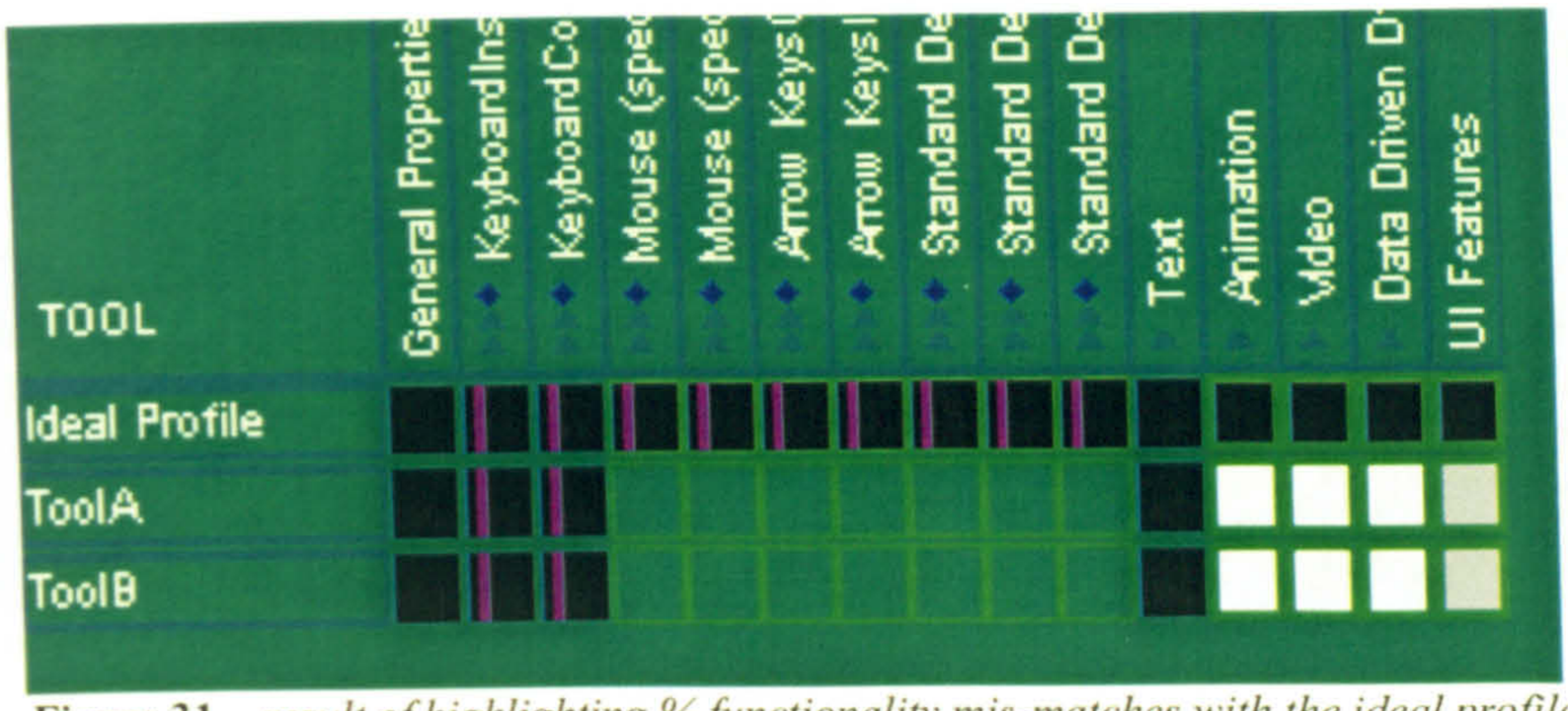


Figure 31 – *result of highlighting % functionality mis-matches with the ideal profile*

Suppose you have identified (highlighted) those components which match the needs of the project in terms of requirements (i.e. matches at the % functionality level). You might now want to see whether or not they utilise those interaction mechanisms which you have identified as being the most appropriate for the team members working on the project – it may be on this basis that you can eliminate some tools from those being evaluated (i.e. if tools are equal in terms of functionality provision, the means by which the functionality is provided is perhaps the most significant discriminator – especially with respect to the needs of the team members).

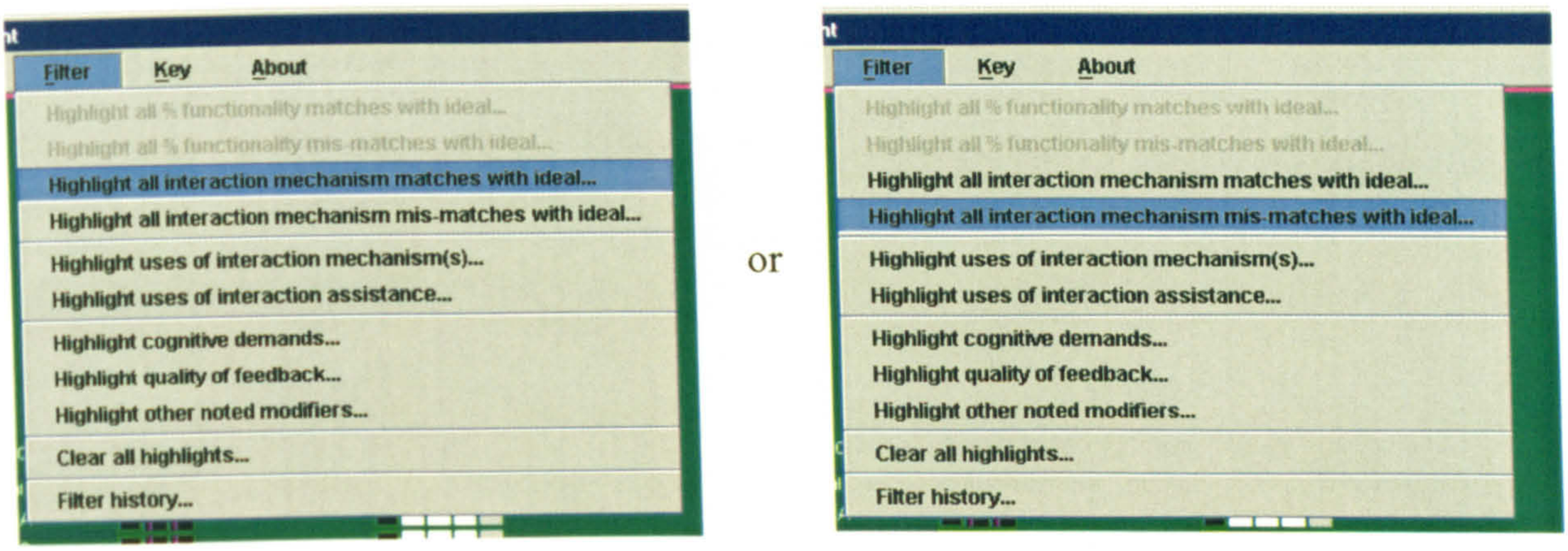


Figure 32 - *highlighting tool matches/mis-matches with ideal profile in terms of interaction mechanisms used*

To do this, you can apply an additional project-specific filter to the active data set - as shown in Figure 32. It demonstrates that, in order to highlight the component instances where the recorded interaction mechanisms match those of the ideal profile, you should select the `[Menu]Filter::[Menu Item]Highlight all interaction mechanism matches with ideal...` option. Conversely, to highlight the instances where the recorded interaction mechanism(s) do not match with the ideal, you should select the `[Menu]Filter::[Menu Item]Highlight all interaction mechanism mis-matches with ideal...` option. Filters are combined using AND such that all components highlighted at any one time conform to all applied filters.

The result of the filter application is shown in Figure 33 which illustrates those components for which there is a match at the % functionality level, but discrepancies at the level of interaction mechanisms employed. Hence, in the example given, it can be concluded that, in several instances, despite both tools meeting the % functionality requirements, neither are appropriate in terms of the interaction mechanisms used. As a reminder, you can confirm the history of filter application as was demonstrated in *scenario 1*.

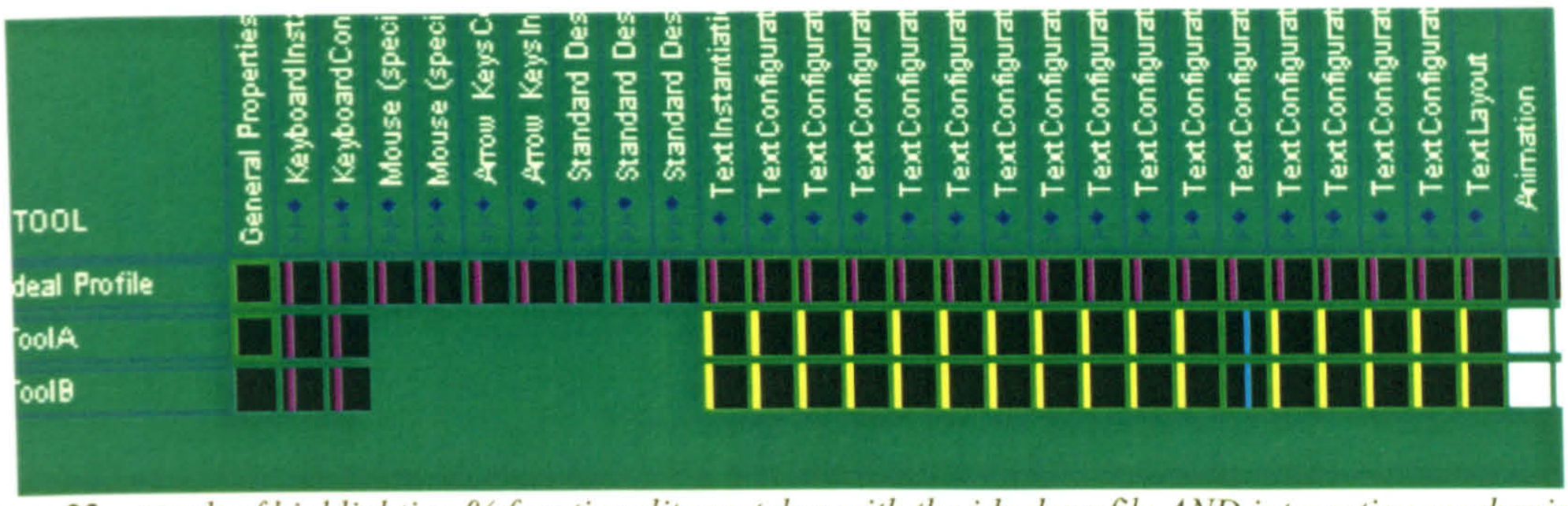


Figure 33 – *result of highlighting % functionality matches with the ideal profile AND interaction mechanism mis-matches*

SUIT

Analysis Record Sheets

The following collection of data record sheets **should be used as templates and photocopied as required** for the completion of a tool selection.

The first sheet is provided to assist in the recording of high-level details concerning the comparison/selection – it need only be **copied once per comparison task**.

The second sheet is provided to guide the recording of selection decisions or observations – for example, you would use this sheet to record observations made during comparison of the various tools such as the fact that ToolX does not, in any instance, use Graphical Manipulation, and therefore should be rejected. The data recorded on these sheets forms the argument on which your tool selection is based. This sheet should be copied **for every observation made,** and should be attached to a screen dump of the visualisation supporting the observation as required.

The third and final sheet should be used to record an explicit statement about the tool which you are recommending as a result of the comparison process, and should be used to include a summary of the rationale behind that decision. This sheet should be copied **once per comparison task**.

At the end of a comparison, it is recommended that the above copied and completed sheets be stored with the **SUIT** framework sheets used to gather the initial data. In this way, the process is recorded for future reference and can therefore inform a later tool selection.

Evaluation Ref: [ ]

Evaluation Details

Evaluation completed as part of a: ☐ Generic Comparison ☐ Project-Specific Comparison

Evaluation performed by:.....

Date: .....

Project Details

Project Ref: [ ]

Project Title:.....

Project/Team Leader:.....

Project Description:.....

.....

.....

Comments

.....

.....

.....

.....

.....

.....

.....

.....

Observation/Decision Ref: [ ]

Observation/Decision Note

Screen dump of visualisation attached: [ ] yes [ ] no

Filters applied: .....

.....

.....

Description: .....

.....

.....

.....

.....

.....

.....

.....

Decision: .....

.....

.....

.....

.....

.....

.....

Tool Recommendation

Recommended Tool: .....

.....

Rationale for recommendation: .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Other comments: .....

.....

.....

.....

.....

.....

.....

.....

.....

Signed: ..... Date: .....

APPENDIX F: HANDOUTS FOR QUALITATIVE STUDY

Tutorial Questions

SUIT

Tutorial Tasks/Questions

The following tutorial tasks and questions have been designed to ensure that you have a basic knowledge of the SUIT visualisation environment. Please try to complete each of the tasks in turn and to answer the questions where appropriate. If you are unsure how to complete a task or how to interpret the data in order to answer a question, please ask us for assistance.

Your tutor will launch SUIT for you, and will open up a pre-prepared tutorial visualisation for which you should complete the following tasks and answer the questions as appropriate.

Task/Question 1: Identify the components for which tools provide the required functionality but adopt inappropriate interaction mechanism(s) – i.e. ones which do not conform to the ideal profile.

Answer:.....

For the components listed above, identify the types of interaction assistance provided by the various tool(s) and if possible suggest, on the basis of this issue, which tool would be the most suitable.

Answer:.....
.....

Task/Question 2: What required functionality is missing from which tool(s) ?

Answer:.....
.....

Task/Question 3: Take a snapshot of your visualisation and record a comment to outline what the snapshot represents.

Task/Question 4: On the original visualisation contract/expand the categories identified in tasks 1 and 2 so that you are looking at the features one level higher than the interaction mechanism level (i.e. one level higher than the lowest level).

Task/Question 5: Re-order the categories identified in tasks 1 and 2 so that they are at the head of the table.

Task/Question 6: Hide the `other hardware devices` category.

Task/Question 7: Clear any existing highlights.

Task/Question 8: Are there any features for which a positive comment/modifier has been recorded regarding the quality of feedback ?

Answer:.....
.....

Task/Question 9: Save your tutorial work.

Evaluation Scenario

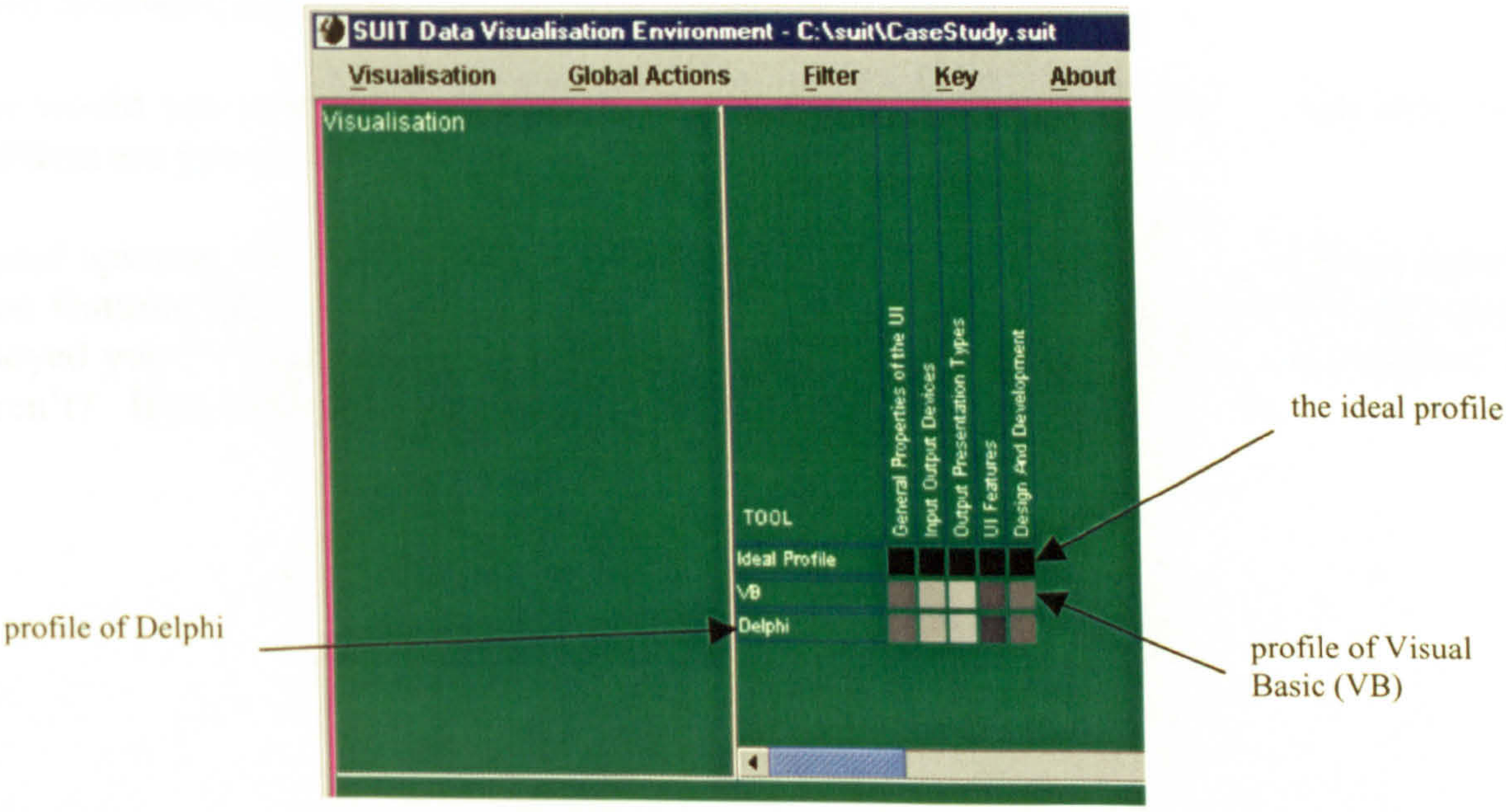
THE SCENARIO

Your organisation needs to select a user interface development tool (UIDT) for use on a *specific* software development project. The selection has already been narrowed down to two well known UIDTs – Visual Basic™ and Borland's Delphi™. Both of these tools are expensive to purchase and deploy (in terms of training) and so it is important to your organisation that the correct choice is made. The project for which a tool is to be purchased, is facing strict and pressing implementation and product delivery deadlines. Hence, the UIDT must be chosen to best expedite project completion.

Your organisation has allocated the project team members – of which there are five – and has identified those components which are essential within the selected UIDT. It has been decided that the SUIT Evaluation Methodology is to be adopted in order to assist and inform the UIDT selection. To this end, the organisation has obtained SUIT formatted tool profiles for both Visual Basic™ and Borland's Delphi™ and, on the basis of the project requirements and team's skill base, has completed a profile of the 'ideal tool'.

YOUR TASK

You have been allocated the task of comparing the above data in order to make a reasoned recommendation for the purchase of one of the two tools. The initial visualisation of the data has already been set up for you. As demonstrated below, it shows the data for the ideal profile and the data for each of the two development tools. Using the facilities available within the SUIT Visualisation Environment, you are required to examine the data with a view to making a rationalised recommendation for the selection of one.



Interview Questionnaires

SUIT Evaluation

Possible Interview Questions

Pre-evaluation session

Do you have an overall strategy in mind before you start ? If so, what is it ?

Post-evaluation session

Were you able to follow your intended strategy when using the SUIT system as a whole? If not, how did your strategy/approach develop as you went along – what influenced it etc?

Did the SUIT methodology influence the approach you adopted to comparing the data ? If so, in what way? What data comparison strategy did you adopt – i.e. what were your main areas of interest when examining the data and why? Was there data that you would have liked to have seen represented which was not included in SUIT? Was there any significance to the order in which you performed the tasks?

Would you have performed an evaluation of tools without SUIT ? How would you have tackled this problem without SUIT? In your opinion, did SUIT – and in particular the visualisation environment – make the evaluation of these tools easier than you think it would otherwise have been ?

How would you rate your confidence in your performance with SUIT? Similarly, how confident are you with the results you obtained through using SUIT ?

In your opinion, what were the strengths and weaknesses of the SUIT visualisation system? What features (if any) were you particularly happy with? What features (if any) really annoyed you ? Were there any commands which you felt should have been provided but weren't? If so, what are they and why would they have been useful?