Using GOMS to predict the Usability of User Interfaces of small off-the-shelf software products.

A Dissertation Presented in Fulfilment of the
Requirement for the M.Sc. Degree

August 1990

Aine P. O'Neill
School of Computer Applications, Dublin City University

Supervisor : Dr. A. Moynihan

Declaration

This dissertation is based on the author's own work . It has not been submitted for a degree at this or any other academic institution.

Aine P. O'Neill

August 1990

Acknowledgements

I am grateful to my supervisor, Dr. Tony Moynihan for his help and guidance. Thanks due to all my colleagues at Carlow R.T.C. for their support and encouragement throughout the two years. I wish to thank Mary Cashman for her constant encouragement as she participated with me on this course. Finally, I wish to acknowledge the financial support for the DCU for waivering a portion of the course fees.

TABLE OF CONTENTS

Abstract :

The design of user interfaces and how usable they are, are both important research topics in computer science. This thesis is a research effort aimed at exploring the whole concept of usability and measuring the quality of a user interface in terms of how usable it is. Usability means how easy a system can be learned and used. In order to have usable products, they must be initially designed with usability in mind. A survey of methods for designing user interfaces which incorporate usability are outlined and they include some or all of the principles for designing for usability, proposed by various authors.

Evaluating the quality of existing interfaces can be done by various methods.The method used in this dissertation is the GOMS (goals, operators, methods and selection rules) approach. This model was initially proposed by [Card, Moran & Newell 83] and the approach is based on constructing an explicit model of the user's procedural knowledge, entailed by a particular system design. [Kieras & Polson 85] expanded this model to suggest that quantitive measures defined on this explicit representation of the user's knowledge can predict important aspects of usability. The predictions are obtained from a computer simulation model of the user's procedural knowledge that can actually execute the same tasks as the user.

To test the reliability and accuracy of the GOMS model predictions, the author carried out a pseudo-experiment on four inexperienced users using two different types of word-processors. The actual results from the experiment were compared with the GOMS predictions.The GOMS model was found to have some limitations and some enhancements to the approach are proposed. It was also found that the experiment had some limitations and improvements for a better experiment are proposed.

Chapter 1

Introduction

Have you ever used a computer package and found that you don't know the right commands or command syntax, that you are receiving meaningless error messages, that you are reading poorly formatted displays, or have found yourself 'lost' in the system?. Overall you become confused and frustrated and having to use far more effort than necessary to get the computer to perform, even the most simplest task .

This is the problem of usability.

Usability goals can be included in the design stage of software development and there are numerous methods developed which demonstrate how this is done. eg. usability specifications, user derived interfaces, executable specs etc. which will be discussed later.

But, wouldn't it be nice to be able to select a piece of off-the-shelf software (ie. popular PC application software) , and just by running it through a few 'tests', be able to predict how easy this piece of software is to use ?. The approach used should be relatively economical and quick to perform. It should also be reliable and accurate.

To address the problem of usability, an approach is presented which is based on the GOMS model of human-computer interaction

[Card,Moran & Newell 83]. The GOMS model, an acronym for Goals, Operators, Methods and Selection rules describes the user's goals for executing a task, the constituent subgoals, the methods available to accomplish particular subgoals on a specific system, the operations and actions necessary to execute each method. [LaLomia & Coovert89] defined it as an application of a general theory that assumes humans are symbol processors and when performing a complex task a user's behaviour is described as the repeated application of a small set of processing steps or elements.

The GOMS methodology has been used and extended by many researchers. The basis of this dissertation is a production system approach which was developed by [Kieras & Polson 85]. This approach is an extension to the GOMS model, in that it uses a task modelling language , which is used to build a computer simulation model of the user's procedural knowledge that can actually execute the same tasks as the user.

The procedural knowledge is represented in the form of a production system. This representation provides a description of the knowledge in terms of units of roughly equal 'size', namely the statements of the procedures, which can then be counted to yield quantitative estimates of the amount of knowledge required in the execution of a task. I manually examined the set of productions produced rather than using a computer simulation because to do this I would need some type of interpreter to translate the productions. This idea of computer simulation via production systems was initially proposed by [Newell & Simon 72] as a tool for evaluating problem solving models.

The objective of this dissertation, was to become familiar with the GOMS model, it's components and it's operation, in sofar that I could apply the GOMS method to particular off-the-shelf products. This was used to predict the usability of the products. As an extension to this, the reliability and accuracy of the model was tested. To do this, I conducted a pseudo-experiment, comparing the predicted results of GOMS with the actual results achieved from experiments. I refer to it as a pseudo-experiment because proper experiment controls and conditions were not enforced. Throughout this dissertation, I shall refer to the pseudo-experiment, as an experiment (because the word 'experiment' is faster to type and to read).

The experiment took place in 4 phases :

1. The GOMS models of 2 word-processing packages were drawn up and the learning times were predicted.
2. Four in-experienced users were given a set of standard tasks to complete, and their learning times were recorded, as they worked on the packages.
3. The users were also given a questionnaire which was to determine their attitudes towards the products.
4. Assessment of the results of the experiment. That is, the predicted results were compared to the actual results obtained and conclusions were drawn.

I chose text-editing rather than any other PC application because it is the most popular microcomputer application. Note, though that

text-editing doesn't occur in isolation from other applications. The user frequently needs to interact with other areas of a computer system in the process of editing a document. The user may want to perform a quick calculation or a detailed statistical analysis and incorporate the results directly into the document.

1.1 A Usable User Interface

Many software packages available are of an interactive nature ie. user enters data and the computer responds to it. This type of interaction is called human-computer dialogue - a two-way exchange of symbols and actions between human & computer. The user interface is the supporting software and hardware through which this dialogue occurs.

The terminal dialogue is the central and most intensive medium of human-computer interaction. The quality of the dialogue depends on the match between the technical elements of the computer system and the cognitive characteristics of the user. Some of the more important facets of the system and user are summarised in fig.1.1.

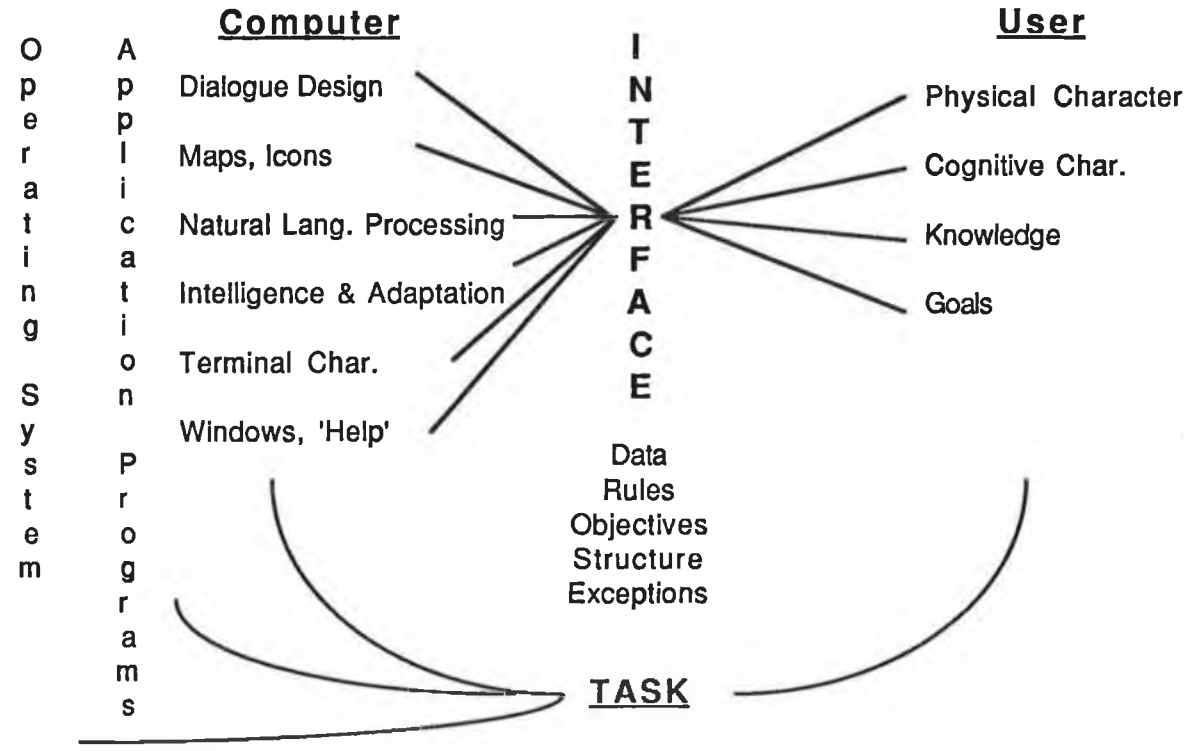

Fig 1.1: The central elements of human-computer interaction

A good interface is an essential part of any system. Even an excellent system would be useless without a proper interface. Many researchers have suggested criteria and desirable properties of an interface. An interface should be simple and reliable, yet flexible and easy to modify. It should offer proper guidance and support and be compatible throughout it's menus and submenus. It should provide the appropriate functionality and information feedback. Documentation and error messages should be very clear and explicit. It should be easy to learn and easy to use.

The latter two properties are the basis of this dissertation - Usability.

Usability is not easily defined, but everyone knows what it is. It is affected by the types of tasks that are to be performed ie. it is task-related. It is also people-related. The characteristics that make a system usable for one set of users may render it unusable for another.

This was illustrated by SCORPIO, a bibliographic search system which was installed in the Library of Congress. The staff were required to learn and use this new system, and they did so very successfully. Then, the general public had access to it , in order to locate books in the library. For even a computer-knowledgeable individual, learning to use the commands, understanding the cataloguing rules and formulating a search strategy were found to be challenging tasks. In brief the system was seen as.an intrusion or interference with their work. The SCORPIO system that worked so well for one community of users was inappropriate for another.[Shneiderman 86]

Usability is still too often discussed in abstract terms. [Barnard et al 81] 'To be truly usable a system must be compatible not only with the characteristics of human perception and actions, but also most critically, with users cognitive skills in communication, understanding memory and problem solving."

Although this may be valid, it doesn't offer specific guidance.

Others do offer advice.

[Heckel 82] takes the point of view that 'friendly' software is software that communicates well. In his book, he shows that the place to look for understanding on how to make software friendly is in any number of communication crafts.

[Rubenstein & Hersh 84] claims that a computer system described as 'user friendly' or as having 'ease of use' features frequently means only that certain parts have been added in order to eliminate user problems, whereas ease of use can only be designed into the product as a whole.

In this dissertation, I used Paul Reed's definition from [Bury et al 86] that 'usability is the ease with which a system can be learned and used'.

Chapter 2

Usability

The general framework for usability embraces the 4 principal components of any work situation ; user, task, system and environment. Good design for usability depends on achieving successful harmony in the dynamic interplay of these four components. Therefore, usability can be defined in terms of the interaction between user, task and system in the environment.

[Shackel 86] has proposed an operational definition of usability along four factors : effectiveness, learnability, flexibility and attitude.

Effectiveness : is defined as performance which is better than some required level (measured eg in terms of speed and errors), and achieved by a required proportion of the population in the range of usage environments.

Learnability : is criterion performance achieved within some specified time based upon a specified amount of training and user support.

Flexibility : is defined as adaptation to some specified range of variation in user tasks.

Attitude : refers to acceptable levels of human cost (fatigue, discomfort, frustration, personal effort etc.) and perceived benefits (which promote continued and enhanced usage of the system), is defined in subjective as well as performance terms. Subjective measures, collected primarily through questionnaires and rating scales, are important since they provide information about the quality of the interface that are difficult to obtain in any other way.

2.1 Usability & Functionality

When measuring the quality of a user interface, explicit criteria are required. These criteria relate to the objectives of the evaluation. Currently, the most frequently adopted criteria are functionality and usability. Functionality relates primarily to the general objective of assessing the capabilities of the design and refers to the tasks that the system enables the user to perform. Usability relates to the assessment of the impacts of specific design decisions and refers to the ease of use of the interface [Rubin 88].

Too often designers of computer systems equate functionality with usability or view usability features as limiting functionality. A critical step in defining the design philosophy for the user interface is to establish the appropriate balance of ease of learning, ease of use and functionality. Ease of learning is the extent to which a novice user can become proficient in using a system with minimal training and practice. Ease of use is the extent to which the system allows a knowledgeable user to perform tasks with minimal effort. Functionality is the number and kind of different functions the system can perform.

Opinions on the importance of usability in system design are not particularly new or unanimous. [Martin 73] has written that a user's ability to use a system powerfully will depend on the ease with which he or she can communicate with it. [Brooks77] considers usability "the proper criterion for success" and [Bennett78] argues that "user acceptance is strongly affected by how the function is

invoked as well as what function the system contains. [Foley & VanDam82] conclude that usability is at least as important as functionality.

On the other hand, [Fried82] cautions us that "in general, there is little hard evidence to support the idea that ease of use leads to improved (traditional) productivity, or that specific ease of use characteristics truly make software easier to use for a majority of users". In a sense, functionality itself can determine usability; if the functions provided do not match task requirements , a system will not be usable. People must understand what the functions do and how to use them.

Although usability is not an easy concept, investing in usability is as important as investing in functionality. Failure to consider usability can lead to system failure. At best, a system with poor usability will cost its users time and effort; at worst, it will not be used at all, and its functions may be removed because their utility has not been demonstrated. As an integral part of system design, usability contributes to overall system functionality by making it accessible to users and facilitating effective use of functional capabilities.

2.2 Criteria for a Good Interface Design

According to [Mehlmann 81] a 'good' program or package is one that displays few usability defects.

These defects are :

the Speed criterion : it must not take longer to do the job with the help of the package than without.

Accuracy : Information to which the user has access must not be less accurate with the package than without.

Complete : the user must have access to as much relevant information that he wants when using the package.

Pleasure : the package must not take over the parts of the work the users enjoy doing and leave them with the parts they find boring (ie. let the computer do any repetitive work).

Also, a package should be flexible and should meet its objectives.

2.3 Designing for Usability

[Shackel86] outlined 5 fundamental features of design for usability. These are

1. User Centred Design - focus from the start on users and tasks
2. Participative Design - with users as members of the design team.
3. Experimental Design- with formal user tests of usability in pilot trials, simulations and full prototype evaluations.
4. Iterative Design - design, test and measure, and redesign as a regular cycle until results satisfy the usability specification
5. User Support Design - training, manuals, quick reference cards, on-line help etc.

Also [Gould & Lewis85] at IBM Watson Research Centre have devised a methodology from their experiences and have proposed four precepts for design for use . These principles for system design are :

early and continual focus on users, integrated design, early and continual testing and iterative design (the cycle of design, test , measure and redesign repeated as often as necessary).

These are in essence, very similar to Shackel's. The paper also outlines examples of the use of simulation and prototyping as part of the usability development process.

[Hewett & Meadow86] report in their paper an example of how these principles proved their worth in a successful system design project. Their project Individualized Instruction for Data Access IIDA, involved development of a computerized intermediary to assist end users doing bibliographic database searches.. It was designed to enable end users of information retrieval systems to perform their own searches by

1. instructing them in how to search as needed, and by
2. assisting them with the performance of the search providing diagnostic analyses and feedback.

Also [Boies et al 87] evaluated a computer system design methodology. The paper reports on the 1984 Olympic Message System OMS- a voice mail system that was developed according to the four design principles, outlined earlier. Their research demonstrated that any project that followed these principles is do-able , doesn't take too long, and doesn't cost too much. The principles made possible an integration of all aspects of usability.

They led to a reliable, responsive, easy to learn system containing the right functions.

Throughout the following review of methods of designing user interfaces, it becomes apparent how the principles of design for usability outlined above, have been used.

2.4 Survey of Interface Design Methods :

"Simply, applying the latest technology doesn't insure a good human interface. Only careful, iterative design can do that.....We believe that iterative design by itself, should be universally applied to the design of user interfaces". [Good et al. 84]. It will become obvious how nearly all the design methods summarised below, have used iterative design as their main criterion for design. This is due to the fact that feedback from users is the basis on which interfaces are designed.

2.4.1 Usability Specifications :

[Carroll & Rosson85] developed an approach to the usability problem based on usability specifications. These are precise, testable statements of performance goals for typical users carrying out tasks typical of their projected use of the system. These in turn are factored into their behavioural prerequisites which are called

subskills, in order to pinpoint and remedy specific problems in design.

Because they are concerned with the design of user interfaces, they focus on users throughout the design process. They outline that the usability specifications and the subskills they imply, are viewed as being iteratively elaborated and refined throughout the design process. And, since the computer system development process is already organised around the development, refinement and implementation of specifications, this affords a means of incorporating user interface issues into the existent development process.

Figure 2.1 shows how [Carroll & Rosson85] viewed the design of user interfaces. The cycle begins with the generation of design objectives, followed by a looping process where specifications are generated, behavioural subskills are analysed, the subskills are tested in a qualitative fashion and the information fed back into the design specifications. The loop is to provide a process of refining and rejecting interim solutions and discovering new ones.

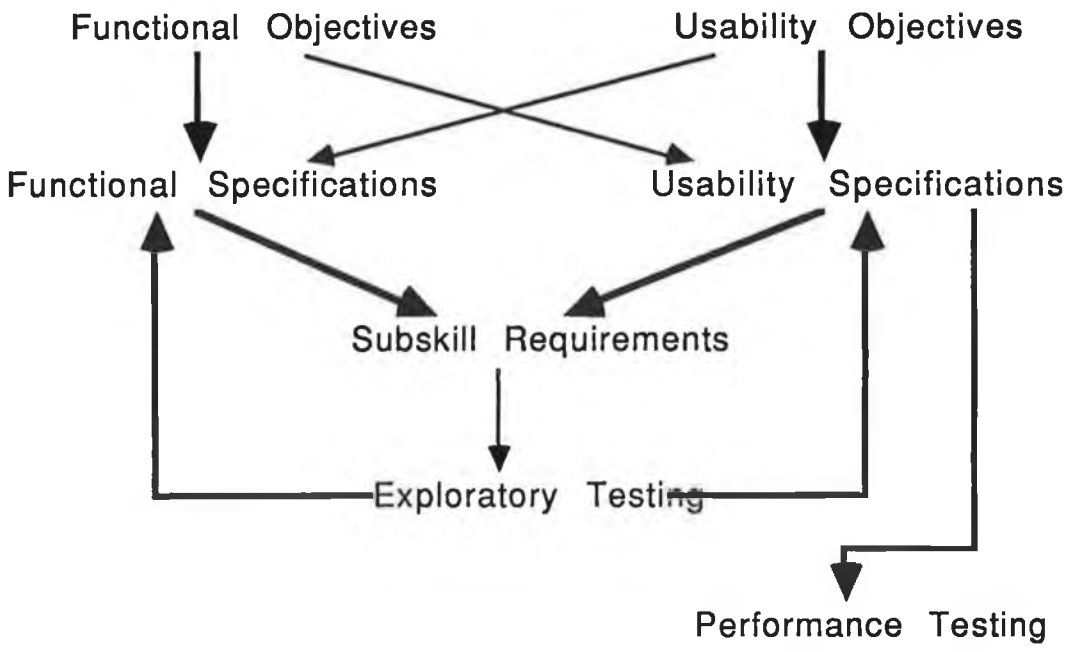

Fig 2.1 : Iterative Design Process

a) Design Objectives

This information is gathered from the intended users of the system. It provides the designer with the relevant functional and usability objectives on what the people need to know and on how the people want to work.

b) Design Specifications

These are motivated by the objectives drawn up earlier. They represent a codification of the design plan. Functional specifications codify the function that is to be intended for the system and usability specifications formulate the user behaviour to be supported.

c) Subskill requirements

The design team now determines the behavioural subskills implied by fulfilment of the specifications. For example, do the users understand how to use a mouse in a mouse-driven menu system.

Thus the decomposition of usability objectives into specifications, and finally into subskill requirements, is an iterative empirically driven process. This design loop of iterative decomposition, refinement and redefinition is at the heart of their view of the design process. Also, this view on design has particular implications for the role of behavioural expertise in user interface design. It shouldn't be seen as something that can be brought in at particular points or stages of the design plan. One must think of behavioural work as part of the design process, as intrinsic to it. Their final word is that ' useful and usable systems can only be designed deliberately'.

2.4.2 Interface Metaphors :

[Carrol, Mack & Kellog 86] developed an approach to control the complexity of user interfaces by designing interface actions, procedures and concepts to exploit specific prior knowledge that users have of other domains. Example : designing an office information system using the metaphor of a desk top.

Instead of reducing the absolute complexity of an interface, this approach seeks to increase the initial familiarly of actions, procedures and concepts by making them similar to actions,

procedures and concepts that are already known. The use of interface metaphors has dramatically impacted actual user interface design practice.

In their paper, they outline a structured methodology for developing user interface metaphors. The method involves four steps:

- identify candidate metaphors

these can be got from predecessor tools and systems, human propensities or sheer invention

- detail metaphor / software matches with respect to representative user scenarios.

- identify likely mismatches and their implications.

- identify design strategies to help users manage mismatches eg UNDO command on Mactintosh, reference information and on-line help.

The approach that they have advocated emphasizes the overall context and process of metaphor use. Interface presentations using metaphors interact with, and frame user's problem solving efforts in learning about the problem domain. Metaphors have been employed to increase the initial familiarity of the target domain- and they do - but, the authors feel that they have an inevitable further role to play.

The ultimate problem that the users must solve is to develop an understanding of the target domain itself, a mental model. The authors point out that to develop a successful interface, the design will need to take into account accumulating observations of user's experience with other metaphoric interfaces, the knowledge that

users can be expected to gain through the metaphoric comparison, as well as the inevitable consequences of the metaphor for the user-as-learner.

Although the concept of metaphors is a widely used design technique for controlling interface complexity, there is no predictive theory of metaphor. Metaphors still need to be generated on a case by case basis.

2.4.3 Executable Interface Definitions:

[Hayes85] proposed an approach to spread the high cost of developing interfaces over a large number of applications by building a single central computer program, called an interface system, to provide user interfaces for all different applications.The interface system finds the details of the interface required by each application in an external, declarative database called an interface definition - one interface definition for each application. This results in the situation in diagram fig 2.2, where the user communicates with the application only indirectly through the interface system.

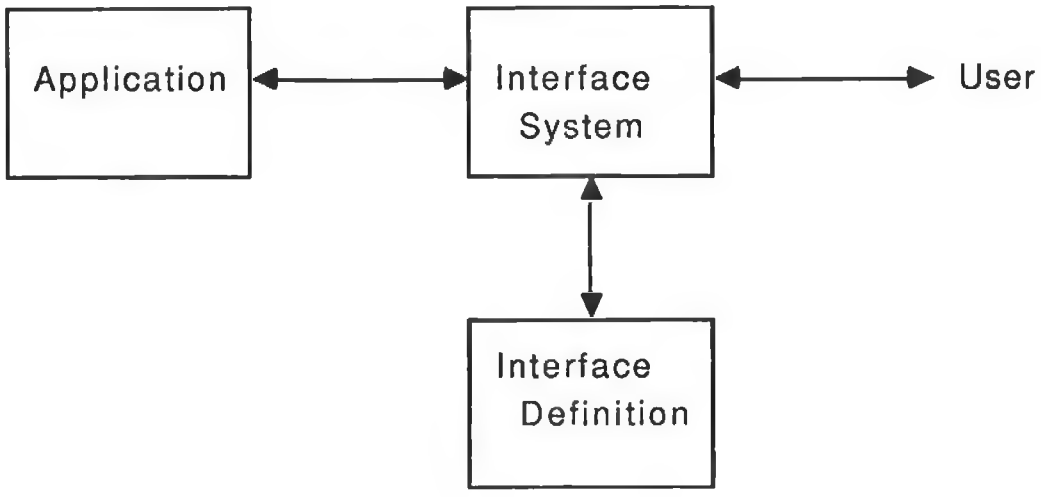

Fig 2.2 : External Definition of a User Interface

This diagram also shows that the interface definition must, in fact, define two interfaces - the user interface, plus an application interface through which the interface talks to the application. However, the latter interface is invisible to the user.

Hayes developed a language to be used for the interface definitions. This language had to be more abstract than the programming languages normally used for interface implementation. It was built around abstractions of the kinds of communication required by the applications.

The advantages of this approach are :-

- o the interface definition will be much smaller and therefore easier and quicker to construct and modify than a conventionally implemented interface.
- o since the definition is external to the application, many modifications can be made without corresponding changes to the application itself. This encourages experimentation with different interface characteristics

and a more extensive and more effective refinement phase.

The interaction required by an application program is specified at a level more abstract than the implemented language of the application. The idea of using these interface definitions is critically dependent on finding suitable abstractions of the interactions required by the applications. The abstractions used are centred around a form-based metaphor of communication. This form contains a field for each piece of information that the user and the application need to exchange. These form-based interface abstractions form a good basis for the following user-friendly behaviour :- correction of erroneous or abbreviated input, interactive error resolution, integral on-line help and automatically generated online documentation.

2.4.4 User Interface Tools:

Creating good user interfaces for software is very difficult. There are no guide-lines or techniques that guarantee the software will be easy to use, and software implementors have generally proven to be poor at providing interfaces that people like. Consequently, interface software must often be prototyped and modified repeatedly.

Interface software is difficult to write because frequently it must control many devices, each of which may be sending streams of input events asynchronously. Also, interfaces typically have stringent performance requirements to ensure that there is no perceived delay between a user's actions and the system's response.

Therefore there is great interest in developing tools that help design and implement interfaces.

As experience with tools for developing human computer interfaces increases, more will be understood about the requirements for such tools. Even though all these tools should be usable and user friendly , they should also be functional, complete and offer structured guidance.

The advantages of these tools is that

1. They produce better interfaces.
2. the interface code is easier to create and more economical to maintain.

User interface tools come in 2 general forms : User interface tool-kits and User interface development systems

User interface tool-kits :

is a library of interaction techniques where an interaction technique is a way of using a physical input device (such as mouse, keyboard etc) to input a value (such as command, number, location etc) , along with the feedback that appears on the screen.

Examples of interaction techniques are menus, scroll bars and on screen buttons operated with the mouse. A programmer uses a user-interface tool-kit by writing code to invoke or organize the interaction techniques. Toolkits do not provide much support for the design or the specification of sequencing and dialogue control.

User interface development system :

is an integrated set of tools that help programmers create and manage many aspects of interfaces. These systems are usually called user interface management systems (UIMS).

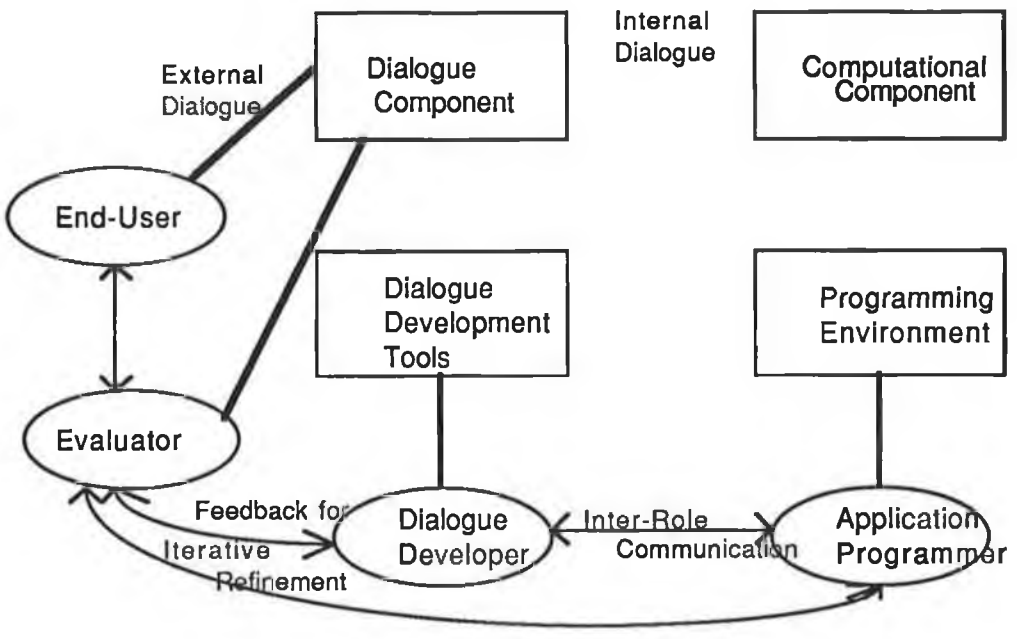

Fig 2.3 : Typical Structure of a UIMS

The typical structure of a UIMS is illustrated, along with appropriate roles involved. A dialogue developer interacts with automated tools for developing the application's systems human computer interface : these tools produce an internal stored representation of the dialogue that is executed at run-time to produce the interface. An application programmer produces the application system's computational software, providing its functionality. These two developer roles communicate and coordinate their development efforts. End users and the system evaluators give feedback about the interface and the system functionality. The entire process forms a cycle of iterative refinement.

Some examples of UIMS are :
Dialogue management system (DMS) [Ehrich & Hartson 81], is a research UIMS that has been developed as a test bed for interface management concepts.
RAPID/USE [Wasserman 85] is a direct manipulation dialogue development tool which uses state transition diagrams to provide a graphical language.

I shall outline the DMS tool in more detail :

Dialogue management system (DMS) [Ehrich & Hartson 81]:

DMS was developed at Virginia Tech by H. Rex Hartson, Deborah Hix, and Roger Ehrich and it is a comprehensive system for interface management. DMS 3.0 is built on a Smalltalk-80 (object

oriented) platform running on a Macintosh II. DMS is a complete system for defining and managing human-computer dialogues. It is based upon the hypothesis that dialogue software should be designed separately from the code that implements the computational parts of an application, and different roles are defined for the dialogue author and the programmer to achieve that goal.

DMS contains an integrated set of interface development tools called Author's Interactive Dialogue Environment (AIDE), in earlier versions of DMS. These tools embody a structural model, methodology, representational notation, life cycle management and rapid prototyping. Tools include a display tool, several menu tools, a forms tool and primitive libraries. In addition it contains several generic tools for developing interfaces not supported by specific tool. DMS itself has a direct manipulation interface. The DMS approach to interface development considers human computer interface management as an integral part of software engineering.

An application system developed using DMS is viewed as having three components : a dialogue component through which all communication between the end user and the application system is carried out, a computational component that contains all semantic processing algorithms, and a global control component that governs logical sequencing among dialogue and computational components. Dialogue independence forms the fundamental philosophy of DMS and helps ensure easy modification of the interface allowing two or more very different interfaces to be used with the same computational and global control components.

2.4.5 Usability Engineering :

Usability Engineering is defined as a process whereby the usability of a product is specified quantitatively, and in advance. Various authors [Gilb 77], [Bennett 84], [Butler 85] , [Good et all85] etc have proposed or described the use in practice of specifying measurable usability goals as a means of planning and controlling software development. I shall do a brief review of their contribution to this area.

[Brooke 86] describes how usability goals and usability engineering techniques are being applied in the context of the development of computer-based office products. Whether or not the specification of usability goals is a formal part of product development procedures, the application of usability testing techniques can help in the identification of design and implementation flaws which affect the usability of products. However, the power of such testing is much improved when usability goals are incorporated into product requirements, because they then become another target for a product to meet, rather than something that is treated as an afterthought.
In the paper, he outlines 2 categories of usability criteria : user performance measure and user attitude measure. The metrics are firstly defined, and then the usability goals are set for each metric. The worst case and the best case values are set for each criterion, for both expert and novice users.. He stresses that the usability goals set must be a realistic reflection of the likely use of the product ie depending on whether experts or novices will be using the product. Note that, setting the goals does not imply anything about how the

goals should be reached by the software developer. In order to ensure that the usability goals can be reached, empirical testing of the product takes place. The purpose of such testing is that it enables the product to be measured against the usability criteria and also a qualitative analysis of the testing sessions allows the human factor engineer to identify those problems with the product which will contribute most to improving the overall usability of the product. [Tyldesley 88] also wrote a paper on employing usability engineering in the development of office products. He outlines the steps that are traditionally involved in usability engineering in Digital. They carry out much the same steps that Brooke described earlier, but Tyldesley stresses the point that the design is done iteratively, incorporating user feedback until the planned levels of usability are achieved.

[Good et all 86] introduce user-derived impact analysis as a tool for usability engineering. It involves applying usability engineering to a specific product. Again, they follow the same steps :

defining usability through metrics,
setting planned levels of usability
analysing the impact of design solutions
incorporating user-derived feedback, and
iterating until the planned levels are achieved.

Impact analysis [Gilb84] is a method of estimating the probability that a set of proposed design solutions will result in successfully meeting the engineering goals of a product. Impact analysis is a technique for estimating which solutions will be most effective for meeting planned levels for various attributes, as well as estimating the likelihood that the solutions will be sufficient for meeting these

attributes. It is an aid for deciding how to allocate scarce engineering resources.

The approaches and tools outlined, in this section, are all means of designing user interfaces. The characteristic of all the methods, is that they all design products with usability as the main objective. That is, the design team are conscious to fulfil the usability criterion, from the initial stage of the design process.

In the next chapter, approaches for assessing usability in the post-implementation stage are outlined.

Chapter 3

Assessing the Usability of existing interfaces

The methods adopted for evaluating a product will vary widely with the product in question and the metrics established for usability.Even though, it is now more common for evaluation to occur throughout the design process, allowing more frequent, more rapid and earlier evaluations of the design, it is actually easier to evaluate a system that already exists. This is because much of the information needed for evaluation can be obtained from the system itself, it's documentation, it's designers and it's present users.

3.1 Existing Evaluation Methods

Evaluating designs has become a very important concept. It has progresses from what was an informal discussion between designers to a planned, careful and methodical enquiry.
Numerous evaluation methods have been proposed and developed, of which there are two main types. : Empirical methods & Formal Methods.

Empirical methods :
These methods collect evaluation data about the user interface, and once collected, it can be analysed. The principle evaluation methods of this type are experiments, observations and surveys. In general,

they are a sort of common sense testing method in which all the functions to be provided by the system are reviewed. The key objective in these methods is to assure that no obvious errors have been made and to make minor adjustments that seem reasonable.

[Ravden & Johnson 89] developed a survey method, based on a practical tool, in the form of a check-list. They outline the full checklist in their book, and the same checklist can be used to evaluate many different products. Thus, there is no need to tailor it to suit particular applications. The questions on the check-list are based on a set of 'goals' which a well-designed user interface should aim to meet. To evaluate a system, the check-list is distributed to the end-users. They answers all the questions and then the check-list is assesses by the evaluator. This method provides a standard and systematic means of enabling those evaluating an interface to identify and make explicit problem areas, areas for improvements etc..

Formal methods :

These methods have formulated the application of the evaluation methods. These methods have emerged from attempts to model user interaction with the interface.

[Kiss & Pinder 86] assess the quality of a user interface in terms of the user effort required for the operation of a system. User effort is interpreted to mean the computational work done by the user in terms of interface operations in carrying out tasks on the system. The execution of the operations by the user are regarded as the execution of computational algorithms ('procedures'). Complexity

theory [1] is then applied to them, in order to analyse the properties of them in terms of 'ease of use'.

[Reisner 82] has argued that user interfaces should be described in terms of a formal grammar expressed in BNF notation and that the length of sentences generated by this grammar is to be used as a measure of task difficulty. She has also suggested that the number of BNF rules is to be regarded as an indicator of the complexity of the interface design.

[Moran 81] developed Command Language Grammar (CLG). This shares a close relationship with the GOMS model as it was developed around the same time. The CLG framework consists of a set of operators and methods (as in GOMS), and a set of goals called tasks, which are organized functionally. The components are stratified into distinct levels. The same basic notation is used at each level :

task level :	analyses the set of tasks that the user wishes to accomplish
semantic level :	outlines the objects in the system and procedures for manipulating these objects
syntactic level :	translates the information gathered from the task and semantic levels into command language.
interactive level :	converts command language into dialogue used when working the system

[1] Complexity theory analyses the resources needed for the computation of functions in the execution of algorithms. The fundamental resources are space and time.

Example :

The task level for the task Reply-To-Message might be

FIND MESSAGE
SHOW MESSAGE
READ MESSAGE
COMPOSE TEXT
SEND MESSAGE

The semantic level specifies the operations within the system to complete the task. In this example, the semantic level might be

Task Level	Semantic Level
FIND MESSAGE	SHOW DIRECTORY
SHOW MESSAGE	SHOW MESSAGE
READ MESSAGE	READ MESSAGE
COMPOSE TEXT	COMPOSE TEXT
SEND MESSAGE	SEND MESSAGE
	SHOW LIST OF USERS
	SPECIFY RECIPIENT

To fulfill the task procedure FIND a MESSAGE, the operation within the system might be SHOW the DIRECTORY. The semantic level operation for SHOW MESSAGE, READ MESSAGE and COMPOSE TEXT are the same as depicted in the task level. The task procedure SEND a MESSAGE requires three operations, SEND MESSAGE, SHOW LIST OF USERS and SPECIFY the RECIPIENT.

The syntactic level would outline the commands available for completing the semantic level tasks. For example, the command available for the task SHOW MESSAGE might be

SHOW MESSAGE n where n is the message number.

The interaction level provides details of the keystrokes that the user will have to make in order to accomplish the syntactic level procedures. For example the command on the system to SHOW

MESSAGE n might be 'DISPLAY n', where n is the message number.

Each level provides a complete description of the system at its own level of abstraction. The descriptions consist of procedures for accomplishing tasks addressed by the system in terms of the actions available at that level (eg. methods). These formal descriptions can then be used to derive some evaluation measures such as learnability, efficiency of the system, optimality, memory load etc. CLG has been reported as being an efficient tool for evaluating user interfaces [Davis83].

Just as an aside, [Brown, Sharratt & Norman 86] reported in their article how CLG could be used as an interface *design* tool. They take advantage of the feature that the CLG structure provides a way of moving from an informal description (task level) to a formal description at the interactive level. They found it lacking in some ways particularly to the design of adaptive user interfaces. Future enhancements have been proposed.

3.2 The GOMS Model

GOMS is an approach for defining the cognitive procedures that a user must perform at the computer interface. This model describes the user's knowledge in terms of Goals (which the user must accomplish), Operators (the individual actions), Methods (step-by-step procedures for accomplishing goals) and Selection rules

(heuristics for specifying which method to use in specific circumstances).

The GOMS model is an approach to describe user behaviour.The Interface Metaphors approach to design described in Section 2.4.2, is another example of a user model. A User model describes an individual's behaviour when interacting with a computer system and represents the amount and structure of relevant 'how to use a system' knowledge.

The initial notation for GOMS was developed by [Card, Moran & Newell83] and was further enhanced by [Kieras & Polson85], because they found the first model clumsy to use and it didn't explain in any detail how the notation worked. The Kieras & Polson approach called the production simulation approach codifies GOMS into a set of production rules, which when executed by a computer, simulate a user performing a computer task. As I explained earlier, in this dissertation the rules are examined manually rather than using the computer. The basic architecture of a production system includes a set of production rules and a working memory. The working memory represents the current goals of the system including information about current and past actions as well as environmental information.

Kieras, himself developed a language called Natural GOMS Language (NGOMSL) to describe GOMS models,ie the production rules, which has a high degree of precision and is relatively easy to read and write. The goals are represented as the conditions in the production system. Methods are formed from the sequencing of the production rules. Selection rules are production rules that control

the execution of the method. The operators are scattered throughout the production rules. Maybe, in the future, this language could be compiled to make it useful for other areas of HCI ie. implemented as a running computer language to fully complete the production simulation approach.

The GOMS process consists of 2 parts :-

1. the GOMS task analysis : which describes how a GOMS model is constructed for a system using NGOMSL notation.
2. the use of this model to
   a) evaluate the design of the system, namely the user interface, and
   b) predict the human performance.in terms of learning and execution times.

The advantage of GOMS is that it carried out at any stage of the software lifecycle.
During design, the GOMS model can be described concurrently with the design of the system.
During development, the GOMS analysis can be carried out on components from the design stage.
After implementation, is the easiest stage to describe a GOMS model because much of the information needed can be obtained from the system itself, it's documentation, it's designers and the present users.
It is at the post implementation stage, that this dissertation carries out the GOMS analysis.

I shall explain a GOMS task analysis, giving examples using NGOMSL notation. Details of the method and the language are published in a document called ' A Guide to GOMS task-analysis' which Kieras refers to in [Kieras88].

3.2.1 Overview of GOMS Task Analysis

GOMS is a formal means of describing a system. Formal in that it contains a simplified model of the human operator and thus theories of human performance that are entailed in the model. A GOMS analysis is a description of the knowledge that a user must know in order to carry out some specific task on the system. It is a representation of the 'how to do it' knowledge. It is also a description of what the user must learn thus it could act as a basis for training or reference documentation. Each of the methods is made up of certain operators, key presses and hand motions as specified in the Keystroke Model [Card, Moran & Newell 83].
The aim of a GOMS analysis is to describe the system in terms of Goals, Operators, Methods and Selection Rules.

Goals :
The user tries to accomplish the goals. They define the state of affairs to be achieved, and determines a set of possible methods by which they can be accomplished. eg the goal to delete a word. A set of goals are arranged hierarchially, because usually to accomplish a goal, one or more subgoals need to be accomplished first.

Operators:

These are the actions that the user executes. An operator is the action that the user does to achieve a goal eg press the mouse button etc. When carrying out a GOMS analysis , the aim is to describe all the operators as primitive actions ie. they can't be further analysed eg drag the mouse, can be decomposed further to describe pressing the mouse button etc.

There are two types of operators :-

1: external operators : these are actions that the user performs in the system environment.

   a) perceptual operators : eg . 'find the insertion point', 'scan the screen' ie "looking" operators

   b) motor operators : eg. press key, move mouse , ie. "doing" operators.

2: mental operators : internal operators; ie actions that the user reads, then responds by doing what they instruct. eg. Find the goal to be accomplished, Retain information in working memory etc.

There is a need sometimes for a third type of operator, an analyst-defined operator that the analyst herself, (person doing the analysis) can define. This can be used in the instance whereby a process may be too complex to be represented and more importantly, it may have little to do with the specifics of the system. For example an analyst-defined operator can be used to outline the method to use the scroll

bar or to explain the method of how to read a message from the screen. Both of these types of processes can be by-passed ie. you just assume that they will be done without explaining how to do them.

Methods :

A method describes a procedure for accomplishing a goal. A step in the procedure typically consists of a combination of operators, external and/or mental. Describing the methods is the focus of the task analysis since much of the work in analysing a user interface consists of specifying the actual steps that the users carry out in order to accomplish the goals.

eg. **Method to accomplish goal of <goal description>**

1. **operator**
2. **operator**
3. **..**

   .....

n. **Report goal accomplished**

Also, a method can call another method by having as one of it's steps :

4. **Accomplish the goal of <goal description>**

Selection Rules :

When a goal is attempted, there may be more than one method available to the user to accomplish the goal. The selection rule set is the means for handling method selection. It is a series of IF statements which direct control to different methods depending upon some conditions.

eg. **Selection Rule set for goal of <goal description>**

If <condition> then Accomplish goal eg <goal desc.>

...

...

Report Goal accomplished.

The GOMS model analysis is then used to

a) evaluate the quality of the design, and

b) predict the learning and execution times.

The quality of the design is evaluated by general observations in the following areas:-

Naturalness : would the goals and subgoals make sense to a new user on the system or would they have to develop a new way of thinking when going to perform certain tasks ?.

Consistency: are similar goals accomplished by similar methods ?.

Completeness: are there methods for every goal and subgoal ?.

Cleanliness : are selection rule sets clear and easily stated ?. ie. is it easy to pick out which method is appropriate ?.

For the purpose of this dissertation, I only used the learning time prediction, because with the lack of resources, it would be very difficult to test the reliability of the predicted execution times.

The phrase production rules, is the collective name for all the statements of the GOMS model ie.the **Method** statement, the steps in each method, the **Selection Rule Set** statement and the IF statements of the **Selection Rule Set**.

The predicted learning time is estimated by counting the number of production rules [2] necessary to accomplish a specific task. Learning

[2] Throughout this dissertation, I use the terms 'production rules' and 'statements', but they actually mean the same thing.

time is a assumed to be a linear function of the number of productions. The more productions a subject must learn to complete a computer task, the longer the training time.
When all the statements have been added up, the total is used in the following equation to produce a prediction for learning time.

Predicted Learning Time =

(30-60) minutes + 30 seconds per NGOMSL statement.

The statements of the model are counted, depending on the type they are :

- the **Method** description statement counts as 1 statement
- all steps in method including the **Report Goal accomplished** statement, each count as 1 statement.
- **Selection Rule Set** statement and the concluding **Report Goal accomplished** statement, both count as 1 statement.
- All options in a **Selection rule set**, each count as 1 statement.

As well as predicting the execution and learning times of a task, the GOMS model can also be used to predict the transfer of learning. The transfer of learning when going from one task to another is quantified as the number of new productions one must learn. In cases where there are methods common to both tasks, these methods will transfer at no cost, that is they are not added into the overall total.

Chapter 4

Applying GOMS and Assessing the results.

This experiment was divided into 4 stages :

1. A limited number of operations in 2 different word processing packages were analysed in terms of the GOMS model- and 2 models (one for each w/p) were drawn up.
2. Using these models, the quality of the interface was evaluated and the learning times were predicted for the two packages.
3. Four in-experienced users used the two packages, and their learning times were recorded. They had also to complete a questionnaire regarding their views on the two packages.
4. The accuracy and reliability of the GOMS method was assessed.

4.1 Phase 1: Constructing the GOMS model.

The purpose of this phase was to describe a manuscript editing task in information processing terms. The general technique was to observe some experienced user doing the tasks on each word processing package, but I conducted the analysis on the ways that I did the tasks- I have some experience with both the packages. Effectively, I described my own behaviour using a GOMS model.

Ideally, it would have been more satisfactory to construct the model from the behaviour of frequent and skilled users of each of the packages.

The procedure I used, was a top-down , breadth-first expansion of the methods. That is, I described the most general goal (to edit a document) and worked down through it's subgoals until I reached the primitive operator at the end. All of the goals at each level were dealt with before going down to a lower level. It is better to use breadth-first rather than depth-first because when all the methods are described level-by-level, it is easier to pick out methods that are similar to each other. Identifying such method similarities is critical to capturing the consistency of the user interface. Consistency means that similar goals are accomplished by similar methods.

Figures 4.1 & 4.2 show brief outlines of the GOMS methods used for WP1 and WP2 respectively. The diagrams in both the figures are illustrated using the J.S.P. [3] method of design.

Appendix A & B contains the full GOMS models for each word-processor.

---

[3] Jackson Structured Programming : A Program Design Method developed by M. Jackson.

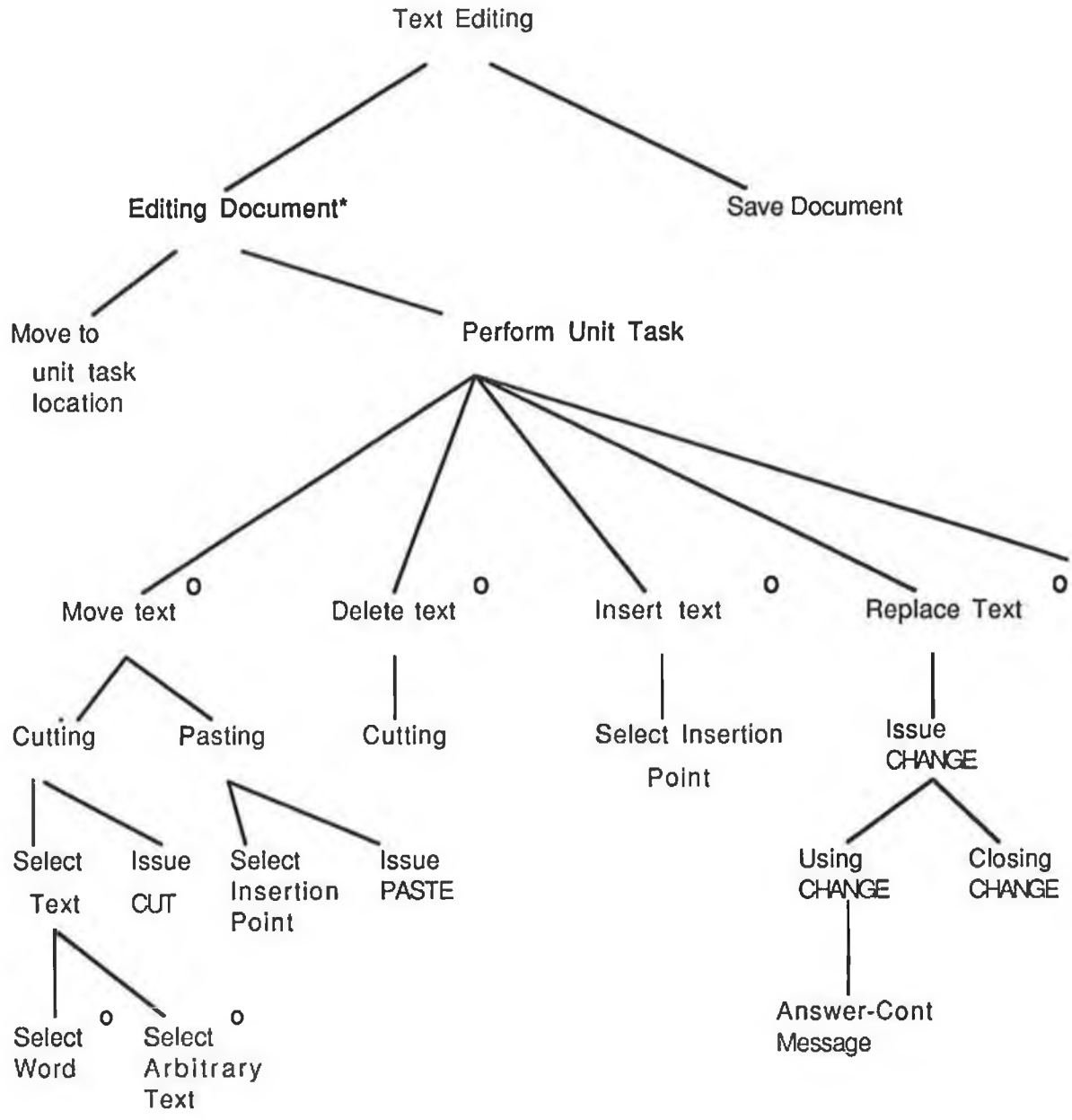

Fig :4.1 GOMS methods used for WP 1

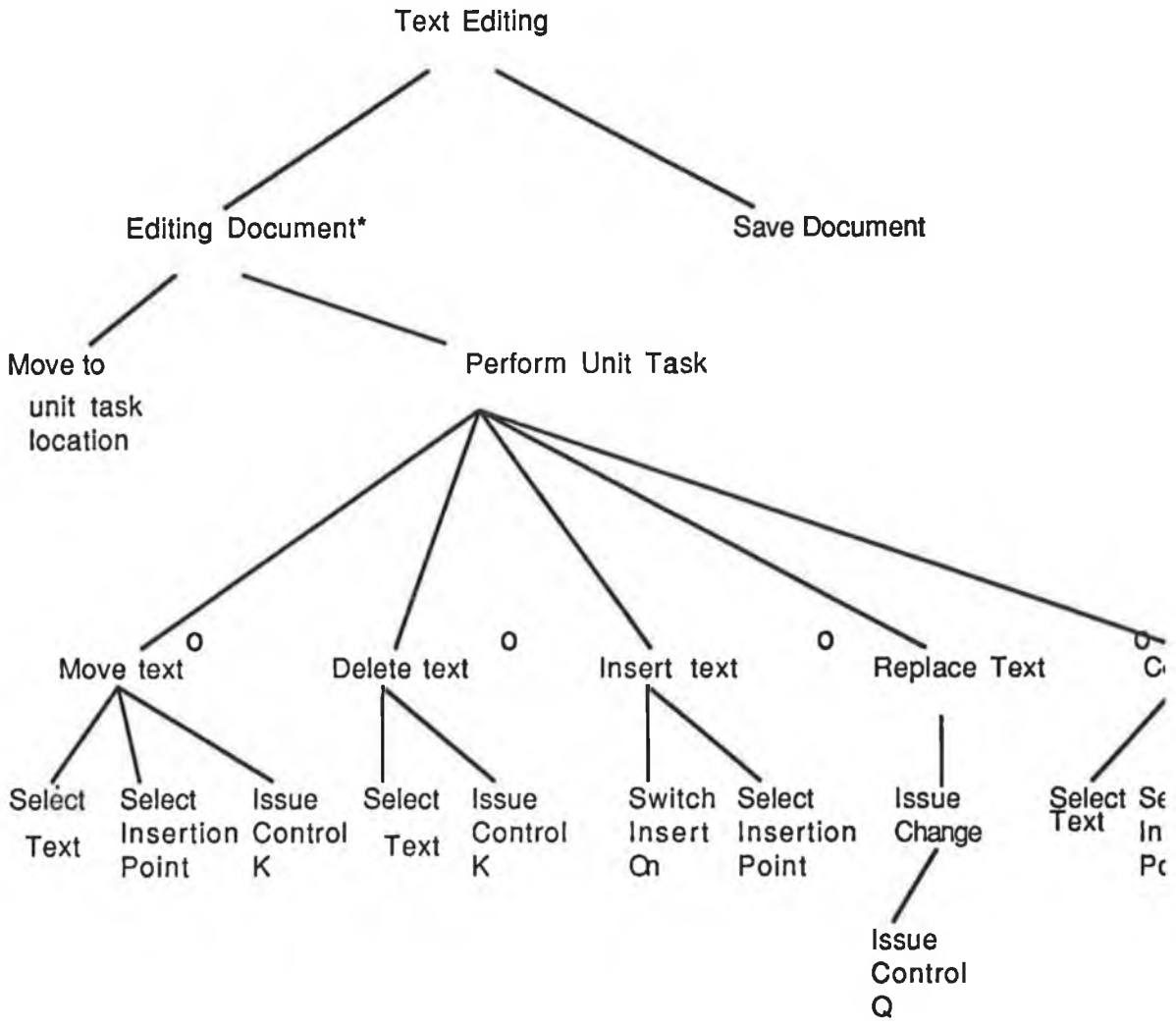

Fig :4.2 GOMS methods used for WP 2

4.2 Phase 2: Quality Evaluation & Learning time Prediction

This phase involves using the GOMS models which were drawn up in phase 1, to

a) Evaluate the quality of the design of each interface, and

b) Calculate the predicted learning time for each package.

Qualitative Evaluation :

This is done by general observations of the GOMS models of each of the packages. Each model is evaluated under the categories outlined in Section 3.2.1.

Naturalness:

W/P 1:

The terms and the jargon (command names etc) used in this package would make sense to any person who has never used a word processor or even a computer before. ie to save a document, click the mouse on the SAVE command.

W/P 2 :

This package doesn't convey natural features quite like the previous package. The users would have to learn the commands for some of the operations eg to save a document, type Control KD. What is the relationship between the word Save and the command Control KD ?

Because, I am evaluating existing packages , the other criteria of evaluation were found to be satisfactory. That is, each of the interfaces were found to be consistent, complete and exhibited cleanliness.

Predicted Learning Times :

This analysis postulates that the number of productions (rules needed to decompose goals into subgoals, to find methods to fit the subgoals and to execute the sequence of actions in a method) necessary to perform a task , is a good predictor of the time it takes to learn a system. Also, the number of productions that the two packages have in common, can be used to predict the transfer of learning criteria. That is, how easy is it to learn a second package, after learning one already.

W/P 1 : was found to have 135 NGOMSL statements :

PLT = (30-60) mins + 30 secs / no. of NGOMSL statements.
= (30-60) mins + 30 secs /135 statements
= -30 mins + 67.5 mins
= 37.5 minutes

W/P 2 : was found to have 88 NGOMSL statements :

PLT = (30-60) mins + 30 secs / no. of NGOMSL statements.
= (30-60) mins + 30 secs /88 statements
= -30 mins + 44 mins
= 14 minutes

4.3 Phase 3 : Experiments on novice users.

The task-based experiment was carried out by observing four novice subjects (A, B, C, D) using the two packages. Novice means that they had no previous experience with computers or word-processors. Four is the absolute minimum number of subjects needed to get some indication of individual user variation. They each had to learn and use two word processing packages. The two word processing packages were chosen specifically because one was mouse and menu-driven and the other was keyboard driven.

I used as much variety in this experiment as I could ,without the proper experimental resources and facilities. Normally many users with different experience and capabilities should have been involved, using many different types of word processors on different types of machines. The set of subjects should be selected to represent the diversity of the user community. [Martin 73] in his book categorises PC users as follows: frequent user, casual user, user with programming skills, intelligent user (high IQ), highly trained user, active user, passive user, and intermediary user. Maybe these types of users could be used as a basis for a proper experiment. This, and other improvements to the experiment are outlined in Section 5.3.

The subjects in this experiment were given the same task which was to modify the same marked-up manuscript on each of the packages. The modifications they had to make were the five text-editing operations, Move text, Delete text, Copy text, Insert text and Replace text. When these tasks had been done they had to save the modified document. The subjects were supplied with the marked up

document , a copy of the revised document and a list of the commands for each package and were shown how to use them.

The experiment was organised such that :

Users A & C	used	W/P 1	then W/P 2.
Users B & D	used	W/P 2	then W/P 1.

The subjects were instructed on how to use each package and terminology regarding the mouse, keyboard and commands was also explained to them. They were taught on a one-to-one basis which had the advantage that it is adaptable to the individual learner. On a one-to-one basis, I could respond to the particular difficulties of each learner by explaining things in a different way, by correcting misconceptions etc. They were allowed to practice as much as they required but once they started modifying the marked-up document supplied, their time was recorded.

When they were finished their tasks on each package, the subjects were asked to fill in a questionnaire. The purpose of this questionnaire (see Appendix C) was to evaluate their attitudes towards each package and it was used to compare with the results of the qualitative evaluation carried out earlier.

4.4 Phase 4 : Assessing the results of the Case Study.

Recall the values that the GOMS model predicted from Section 4.2 :

The predicted learning time for WP1 : 37.5 minutes and for WP2 : 14 minutes.

The actual learning times (in minutes) from the case study are :

Subject	WP1	WP2
A	21	16
B	21	15
C	25	14
D	27	19
Average	23.5	16
Prediction	37.5	14

How do the GOMS predictions compare with the actual results ?

The results for WP1 are not widely distributed and the GOMS prediction seems to be quite a bit higher than any of the actual results obtained. (See Fig 4.3) The predicted result for WP2 seems to be quite close to. the actual values. Fig 4.4

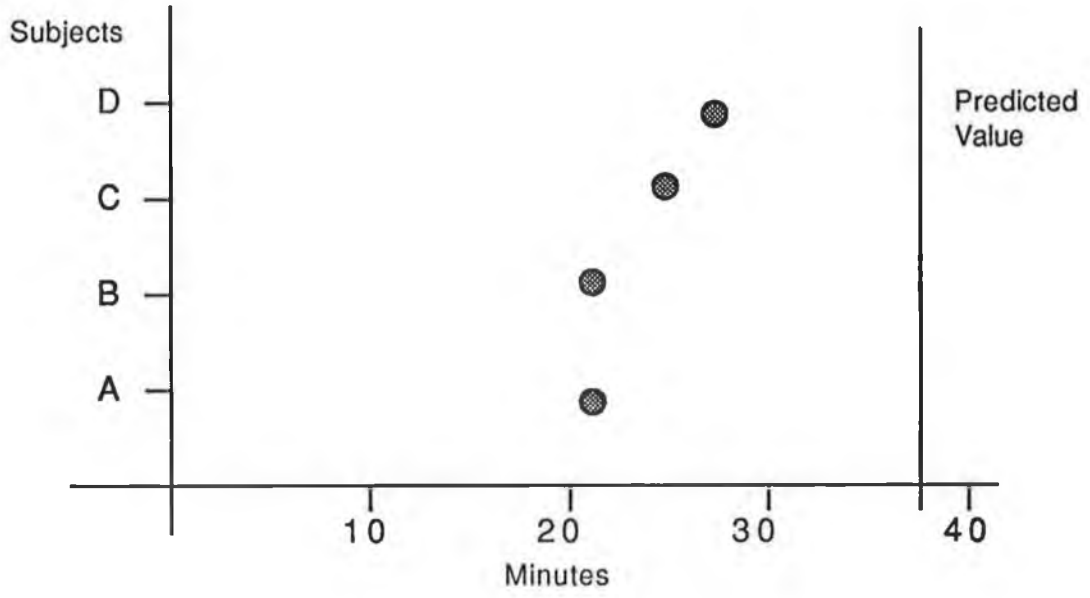

Fig 4.3 : Actual & Predicted Values for WP1

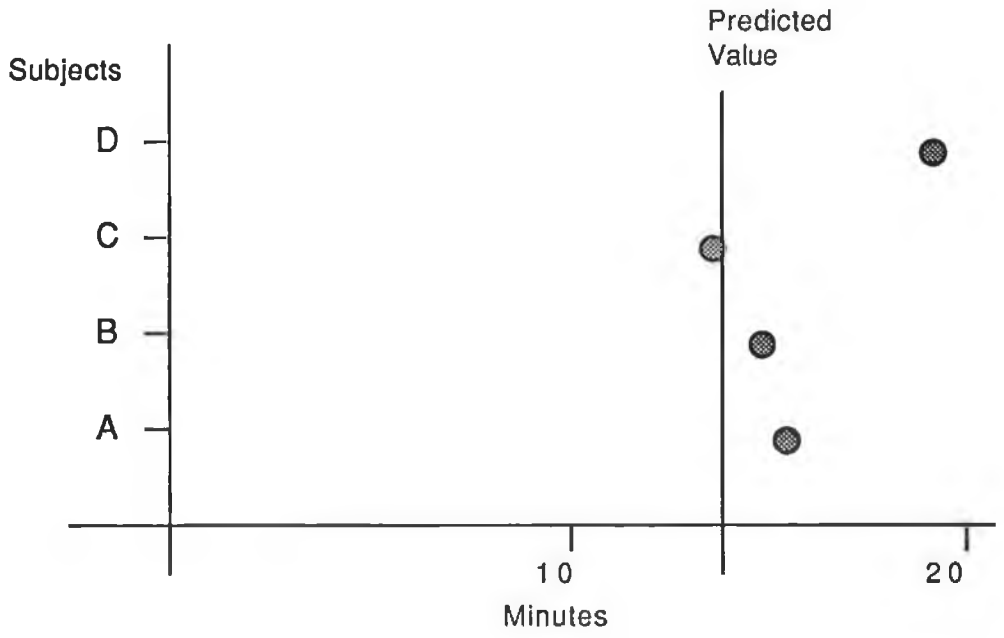

Fig 4.4 : Actual & Predicted Values for WP2

Does it matter in which order the subjects were given the word processors ?

Recall, Groups A&C were given WP1, then WP2

Groups B&D were given WP2, then WP1

Group	WP1	WP2
A	21	16
C	25	14
Aver	23	15
B	21	15
D	27	19
Aver	24	17

It is tempting to answer this question, but it isn't possible due to the lack of proper experimental conditions. Only speculative conclusions could be drawn from these results, because there seems to be a wide variation between the subjects.

There seems to be insufficient evidence of a transfer of learning effect. I would prefer to test this on more subjects and in a proper experiment environment.

Do the types of word processors make any difference ?

As in the previous question, only speculative conclusions can be drawn, to answer this qusetion. Recall, that WP1 was a mouse-driven word-processor whilst WP2 was keyboard driven. It seems

that if you can work a mouse-driven wp (word processor), you can perform better on another type of wp. But using another type firstly doesn't help with the mouse driven one. Thus, it seems to suggest that mouse driven word processors are difficult to master. Maybe it was because the subjects were novices and the idea of clicking a mouse, selecting text etc. took a while to become familiar with. These are all very speculative points but wouldn't it be interesting to find how expert users of both word-processors would perform in this experiment ?.

How did the user evaluation compare with the GOMS evaluation of the packages ?

These results were gathered from the questionnaires which the subjects had to complete. The majority of the users found that WP1 was the easiest to use given the limited amount of documentation and tuition supplied. Initially, they found the mouse difficult to understand and orientate but in the end found it faster and more meaningful than the keyboard. All the subjects deduced that WP1 was the better documented package, easiest to read and they found that it used terms which were familiar to them.

A paradox seems to have arisen !

Why was WP1 predicted to be (and actually was) the longest package to learn and yet the users indicate that they found it the easiest and most user friendly to use ? .

The answer to this seems to be in the method of presentation of the packages. WP1 is a menu-driven package, which is operated using a mouse. It uses different icons - small pictures - to visually represent documents, files, etc.. The user's work is spread out on the screen. Even the cursor takes on different shapes as it is used for different tasks. Specutively speaking, this layout seems to maintain the user's interest as they are using the package.

Because it communicates with metaphors, it seems to trigger the desired knowledge and experience in the minds of the users. Thus, because it was presented in an interesting manner, it required more statements to describe the commands etc.. fully. The longer learning time didn't seem to matter to the users, they found using this package to be the most interesting and useful.

Also, eventhough it takes longer to learn WP1, I would speculate that the commands etc.. learnt from WP1 would be retained longer in the user's mind than those of WP2. Maybe, this could be treated as another interesting extension to the experiment.

The major criticism for WP2 was that it was very easy to get lost in because it had large menus embedded in each other. Pull-down menus were found to be easier to understand and follow through. Also, as in the GOMS evaluation, the control key command was found to be very meaningless. They suggested that if the control key commands were made more meaningful, it might be the easier package to learn and remember.

Again, for this section it would be interesting as to how expert users would answer this questionnaire.

Conclusions :

The GOMS predictions for one of the packages.seemed to come quite close to the actual results obtained in the experiment. The other package which had the mouse driven menu was predicted to have a learning time greater than the actual values. This seems to suggest that the GOMS model might be too low level. Low level in the respect that it describes simple operator commands in very intricate detail. For example, selecting an item from a pull-down menu was described in 5 or 6 steps or productions. Once one had learnt this process once, it becomes more or less an immediate action thereafter.

Also, WP1 displayed and explained the submenus which are in it's package which WP2 didn't. WP1 had steps in the model to explain pull-down menus etc. whereas WP2 just said, for example issue Control QC command, not explaining that the Control Q section of the command brings one to a separate menu.

Finally, the GOMS model for WP1 had more productions than that of WP2, thus predicting a longer learning time for WP1. But this is due to the method of presentation of WP1, which was found to be the more interesting package.

Chapter 5

Conclusions & Summary

5.1 The Advantages of GOMS

As mentioned previously, the GOMS method can be carried out at any stage of the software lifecycle. In my experiment, I demonstrated how it could be used post-implementation, but if it was carried out at the development stage the following would be an advantage. Since the GOMS model is a complete description of the procedural knowledge that the user needs to know in order to perform tasks using the system, the procedure documentation could be written from the GOMS model directly, as a way to ensure accuracy and completeness from the beginning. Even though it is a big help in providing the content of the documentation, it provides little guidance concerning the form of the documentation, the organisation and presentation of procedures in the document. [Elkerton 88] says to make sure the index, table of contents and headings are organised by user's goals, rather than the function names, to allow the user to locate methods given that they often know their natural goals, but not the operators involved.

As defined earlier transfer of learning can be predicted for a specific interface based on the number of new rules to be learned in a task. If this predicted time is extreme and very high, appropriate training procedures could be implemented to simplify the learning

environment. Similar and dissimilar rules would dictate the presentation of interface methods. For example, if a text editor is difficult to learn, a GOMS analysis may suggest initial training for say, selecting text since this method is required for many other commands.

The GOMS model also permits analysis of working memory loads. Specifically, the memory loads experienced by the user can be estimated by counting the goals and subgoals activated during a simulation with the model. From this analysis, predictions for user error rates could be generated with the expectation that high memory loads would result in more errors than periods of low memory loads. High working memory loads also may decrease learning and performance times. Therefore, if periods of high working memory loads can be predicted, then additional prompts and cues could be provided by the software to support error-free and time efficient user performance.

5.2 The Limitations of GOMS

The GOMS approach is very low-level, in that it focuses on very detailed user tasks and user typing behaviour. It doesn't account for personality characteristics or problem solving behaviour of the user which results from completing a task that has no prescribed method. In the experiment the GOMS model had to describe the specifics of how to click the mouse button in graet detail. This is

why the predicted learning time value for the mouse driven word processor was so high.
The GOMS approach doesn't account for memory processes such as comprehension, forgetting or information reorganisation. The fact that these processes take place in the learning stage are not incorporated into the approach.

GOMS is limited to describing the error-free behaviour of a computer user. Making errors is a routine occurrence, thus the model is far from approximating typical user behaviour. Even skilled users spend at least one-quarter of their time making and recovering from errors. Errors in understanding or simple human errors in areas such as typing or pointing can cause complete failure in a users ability to communicate with the system. It would be impossible to consider all possible user responses to a given situation but the GOMS model should take into account time lost due to errors. [Robertson 83] has proposed a method as to how errors and error recovery could be incorporated into a GOMS like analysis.

Also the GOMS approach assumes a computer task can be described in terms of an over riding goal divided into independent, context-free subtasks - this isn't always the situation.

5.3 Improvements to the Pseudo-Experiment.

The definition I used earlier for usability is that it is ' the ease with which a system can be learned and used' [Bury et al 86], so to test these capabilities properly, I would improve on the following :

To test the ease of use :

Many different editing tasks would be embedded into a number of different types of documents eg. an office memo, paged report, chapter from book etc.. The tasks would appear anywhere in the documents and their complexity would be randomly distributed. The usability of the layout and formatting facilities would also be tested. The subjects would be expert users, some from a technical background and some from a non-technical background with no programming experience.

When they are using the system, the overall time to complete the tasks would be recorded. In addition to this, the amount of time that was spent in error-time would also be recorded. This is one limitation of the GOMS model in that it doesn't take into account users making errors. Thus, in recording the time spent in error, it will show whether it is a significant omission from the model.

To test the ease of learning :

The subjects would be novices. They would be provided with a marked-up document similar to the document that was supplied in the experiment I carried out and they would have to perform similiar types of tasks. But as an extension, once the user had learnt

some of the tasks, they would be quizzed on them. This is in order to examine what tasks they could do independently. Each user would be taught following a common syllabus. However, it would be up to the instructor to determine which specific editor commands and facilities to teach in order for the subject to accomplish the core tasks.

5.4 Suggested Improvements to GOMS

[Kieras & Polson 85] stated that the learning time is a linear function of the number of productions. But instead of weighting all the productions etc. with the value of 1 as they have proposed, why not put different weights on the types of operators in the method. Section 3.2.1 outlined the types of operators which can be present in the methods. There are both external operators and mental operators.

Maybe the external operators could have a weight of 0.5, (rather than 1) because they only require a glance at the screen or a touch of a key on the keyboard. These represent more or less immediate actions performed by the user, thus they only require half the original weight..

The mental operators like **Accomplish Goal of..., Report Goal Accomplished** etc. might retain the original weight of 1.

Also, if the mental operators that refer to the working memory, operators like **Retain, Recall & Forget it**, could have double the original weight to be worth the value of 2 because these represent stages when there is a high memory load on the user.

The value of the **Method** and the **Selection Rule** statements are left at their original value.

So, how would the GOMS models of the 2 word-processors score under this speculative weighting scheme ? .

The GOMS model for WP1, was found to have

24 - method & selection rule statements

53 - mental operators

58 - external operators

0 - working memory statements

Thus,

(24+53) * 1 + 58 * 0.5

= 77 + 29

= 106

Use this value in the Learning time equation :

(30-60) minutes + 30 seconds / number of statements

= -30 minutes + 30 secs * 106

= -30 mins + 53 mins

= 23 minutes

The GOMS model for WP2, was found to have

16 - method & selection rule statements

46 - mental operators

27 - external operators

8 - working memory statements

Thus,

(16+46) * 1 + 27 * 0.5 + 8 * 2

= 62 + 13.5 + 16

= 91.5

Use this value in the Learning time equation :

(30-60) minutes + 30 seconds / number of statements

= -30 minutes + 30 secs * 91.5

= -30 mins + 45.75 mins

= 15.75 minutes

Recall, that the average actual values achieved for these word-processors were :

23.5 for WP1 & 16 for WP2.

It is obvious that these predicted values come closer to the actual values than those obtained from the Kieras & Polson's approach. But to make a formal proposal as this being an improvement, this 'improved' method would have to be tested fully and properly. And, to really test this fully, as I have pointed out many times previously, proper experiment conditions and controls must be enforced.

5.5 Summary

In this dissertation I have described the GOMS model of human-computer interaction [Card, Moran & Newell83]. The author has become familiar with the GOMS approach, in sofar that she can apply it to particular off-the-shelf products. The GOMS model was then used to predict the usability of the products.

To test the accuracy and reliability of the GOMS model, a pseudo-experiment was carried out. Four in-experienced users were given a set of standard tasks to complete and their learning times were recorded, as they worked on the packages. The actual learning times recorded were compared to the predicted learning times.

The values of the GOMS model using Kieras & Polson's predicted learning equations seem to be inaccurate for some instances, to the experiment carried out. Maybe this is because of the limitations that were on the experiment or because of the means by which the statements of the GOMS model were added up.

A set of improvements are presented which could be made to the experiment and/or which could be made to GOMS model itself. The improvements to GOMS seem to be more reliable and accurate when compared to the actual values (those that were achieved in the experiment).

The proposal, which is an expansion of Kieras & Polson's approach, takes into account the complexity of the knowledge required to learn the system. Different weightings are used when adding up the

statements, depending on whether the knowledge is perceptual (looking), motor (doing) or mental.

As an expansion to this dissertation, it would be interesting to carry out a 'proper' experiment, using the proposed improvements in the experiemnt to get the actual learning values and the proposed improvements to GOMS to get the predicted values.

Appendix A

GOMS task analysis for text-editing in WP1

Method to accomplish goal of text-editing

1. Accomplish the goal of editing the document.
2. Accomplish the goal of Save Document
3. Report goal accomplished

Method to accomplish goal of editing the document

1. Get next unit task from marked-up document.
2. Decide : if no more unit tasks, then report goal accomplished
3. Accomplish the goal of moving to the unit task location.
4. Accomplish the goal of performing the unit task
5. goto 1.

Method to accomplish the goal of Save Document.

1. Move cursor to "File" on menu bar.
2. Press the mouse button down
3. Move cursor to "Save"
4. Verify that Save is selected
5. Release the mouse button
6. Report goal accomplished

Method to accomplish the goal of moving to the unit task location.

1. Get location of unit task from manuscript.
2. Decide : if unit task location on screen, then report goal accomplished.
3. Use scroll bar to advance text
4. goto 2

Selection rule set for the goal of performing the unit task.

if the task is moving text, then accomplish the goal of moving text
if the task is deletion, then accomplish the goal of deleting text
if the task is insertion, then accomplish the goal of inserting text
if the task is replace , then accomplish the goal of replacing text
if the task is copy , then accomplish the goal of copying text.

Report goal accomplished.

Method to accomplish the goal of moving text
1. Accomplish the goal of cutting text
2. Accomplish the goal of pasting text
3. Verify correct text moved
4. Report goal accomplished.

Method to accomplish the goal of deleting text
1. Accomplish the goal of cutting text
2. Verify correct text deleted
3. Report goal accomplished

Method to accomplish the goal of inserting text
1. Accomplish the goal of selecting insertion point.
2. Type in new text
3. Verify correct text inserted
4. Report goal accomplished

Method to accomplish the goal of replacing text
1. Accomplish the goal of issuing CHANGE command
2. Verify correct replacement done
3. Report goal accomplished.

Method to accomplish the goal of copying text
1. Accomplish the goal of copy/op
2. Accomplish the goal of pasting text
3. Verify correct text copied
4. Report goal accomplished

Method to accomplish the goal of cutting text
1. Accomplish the goal of selecting text
2. Accomplish the goal of issuing CUT command
3. Report goal accomplished.

Method to accomplish the goal of pasting text
1. Accomplish the goal of selecting insertion point
2. Accomplish the goal of issuing PASTE command
3. Report goal accomplished.

Method to accomplish goal of selecting insertion point
1. Determine position of insertion point
2. Move cursor to insertion point
3. Click mouse button.
4. Report goal accomplished

Method to accomplish the goal of issuing CHANGE command.
1. Move cursor to "Search" on Menu Bar.
2. Press mouse button down.

3. Move cursor to "Change"
4. Verify that CHANGE is selected.
5. Release mouse button
6. Accomplish goal of using CHANGE menu.
7.. Report goal accomplished.

Method to accomplish goal of copy/op

1. Accomplish the goal of selecting text..
2. Accomplish the goal of issuing COPY command
3. Report goal accomplished

Selection rule set for goal of selecting text.

if text-is word, then accomplish goal of selecting word
if text-is arbitrary, then accomplish goal of selecting arbitrary text.
Report goal accomplished

Method to accomplish goal of issuing CUT command

1. Move cursor to 'Edit' on menu bar.
2. Press mouse button down
3. Move cursor to 'CUT'.
4. Verify that 'CUT' is selected.
5. Release mouse button.
6. Report goal accomplished.

Method to accomplish goal of issuing PASTE command

1. Move cursor to 'Edit' on menu bar.
2. Press mouse button down
3. Move cursor to 'PASTE'.
4. Verify that 'PASTE' is selected.
5. Release mouse button.
6. Report goal accomplished.

Method to accomplish goal of using CHANGE menu.

1. Move cursor to 'Find What' Box .
2. Click mouse
3. Type in text to be replaced.
4. Move cursor to 'Change to' Box.
5. Click mouse
6. Type in text to replace with
7. Move cursor to 'Change All' box
8. Click mouse
9. Decide :If message-on-screen is 'Continue changing from beginning of document', then accomplish goal of answer-cont message.
10. Accomplish goal of closing CHANGE menu.
11. Report goal accomplished

Method to accomplish goal of closing CHANGE menu.

1. Move cursor to box on top left hand corner of CHANGE menu.
2. Click mouse
3. Report goal accomplished.

Method to accomplish the goal of issuing COPY command.

1. Move cursor to 'Edit' on menu bar.
2. Press mouse button down
3. Move cursor to 'COPY'.
4. Verify that 'COPY' is selected.
5. Release mouse button.
6. Report goal accomplished.

Method to accomplish goal of selecting word

1. Determine position of beginning of word.
2. Move cursor to beginning of word.
3. Double-click mouse button
4. Verify that correct text has been selected.
5. report goal accomplished

Method to accomplish goal of selecting arbitrary text

1. Determine position of beginning of text.
2. Move cursor to beginning of text.
3. Press mouse button down
4. Determine position of end of text
5. Move cursor to end of text
6. Verify that correct text has been selected.
7. Release mouse button
8. report goal accomplished

Method to accomplish goal of answer-cont message

1. Move cursor to 'YES' box.
2. Click mouse
3. Report goal accomplished.

Appendix B

GOMS task analysis for text-editing in WP2

Method to accomplish the goal of text editing

1. Accomplish the goal of editing the document
2. Accomplish the goal of Save Document
3. Report goal accomplished

Method to accomplish goal of editing the document

1. Get next unit task from marked-up document.
2. Decide : if no more unit tasks, then report goal accomplished
3. Accomplish the goal of moving to the unit task location.
4. Accomplish the goal of performing the unit task
5. Goto 1.

Method to accomplish the goal of Save Document

1. Retain that command letter is D, and accomplish the goal of issuing Control K command.
2. Report goal accomplished.

Method to accomplish the goal of moving to the unit task location.

1. Get location of unit task from manuscript.
2. Decide : if unit task location on screen, then report goal accomplished.
3. Use keys on right-hand-side keypad to advance text
4. Goto 2

Selection rule set for the goal of performing the unit task.

if the task is moving text, then accomplish the goal of moving text
if the task is deletion, then accomplish the goal of deleting text
if the task is insertion, then accomplish the goal of inserting text
if the task is replace , then accomplish the goal of replacing text
if the task is copy , then accomplish the goal of copying text.
Report goal accomplished.

Method to accomplish the goal of moving text

1. Accomplish the goal of selecting text
2. Accomplish the goal of selecting insertion point
3. Retain that the command letter is V, and accomplish the goal of issuing Control K command.
4. Verify correct text moved
5. Report goal accomplished.

Method to accomplish the goal of deleting text

1. Accomplish the goal of selecting text
2. Retain that the command letter is Y, and accomplish the goal of issuing Control K command.
3. Verify correct text deleted
4. Report goal accomplished

Method to accomplish the goal of inserting text

1. Decide : if message on top rhs of screen says 'Insert Off', then
   Accomplish the goal of Switch Insert On
2. Accomplish the goal of selecting insertion point.
3. Type in new text
4. Verify correct text inserted
5. Report goal accomplished

Method to accomplish the goal of replacing text

1. Accomplish the goal of issuing CHANGE command
2. Verify correct replacement done
3. Report goal accomplished.

Method to accomplish the goal of copying text

1. Accomplish the goal of selecting text
2. Accomplish the goal of selecting insertion point
3. Retain that the command letter is C, and accomplish the goal of issuing Control K command.
4. Verify correct text copied
5. Report goal accomplished

Method to accomplish goal of selecting text

1. Determine position of beginning of text.
2. Move cursor to beginning of text.
3. Retain that command letter is B, and accomplish the goal of issuing Control K command.
4. Determine position of end of text
5. Move cursor to end of text
6. Retain that command letter is K, and accomplish the goal of issuing Control K command.
7. Verify that correct text has been selected.
8. Report goal accomplished

Method to accomplish goal of selecting insertion point

1. Determine position of insertion point
2. Move cursor to insertion point
3. Report goal accomplished

Method to accomplish the goal of issuing Control K command

1. Press the CTRL key
2. Type the letter K
3. Recall the command letter, and type it.
4. Release the CTRL key
5. Report goal accomplished.

Method to accomplish the goal of Switch Insert On

1. Press the CTRL key
2. Type the letter V
3. Release the CTRL key
4. Report goal accomplished.

Method to accomplish the goal of issuing CHANGE command.

1. Retain that command letter is A, and accomplish the goal of issuing Control Q command.
2. Accomplish goal of using CHANGE menu
3. Decide : if 'Replace Y/N' message not on top rhs of screen, then Report goal accomplished.
4. Type Letter Y
5. Goto 3

Method to accomplish goal of using CHANGE menu.

1. Type in text to be replaced .
2. Press Return Key
3. Type in text to replace with.
4. Press Return key
5. Type letter G
6. Report goal accomplished.

Appendix C

Questionnaire

	WP1	WP2
1. Which was the easiest package to use ?		
2. Which offered the best guidance on how to use the system ?		
3. Which was the better documented ?		
4. Which used jargon & terminology which was most familiar to the user ? ie. command names etc.		
5. Which was the most clear display to read ?		
6. Which had the best colours , ie. easiest to read ?		
7. Which had the best help facility ?		
8. Which system was the easiest to get lost in ?		
9. Which had the most suitable system response time ?		
10. Which had the most meaningful error messages ?		
11. Which has the most consistent design ?		
12. Which of these word processors do you prefer ?		

13. What features were omitted from these packages, that you would like to see included.

W/P 1:

W/P 2 :

Bibliography

[Barnard et al 81]
Barnard, Hammond, Morton & Long : Consistency & Compatibility in human-computer dialogue. Int. J Man-Mach studies, 15,1 (July 81) pp 87-134

[Bennett 78]
Incorporating usability in system design: the opportunity for interactive computer graphics. In Proceedings of the International Conference on Cybernetics and Society Nov 1981 pp 1119-1124

[Bennett 84]
Managing to meet usability requirements. In Bennett, Case, Sandelin & Smith (editors) Visual Display Terminals : Usability issues and Health concerns pp 161-184

[Boies et al87]
Boies, Gould, Levy, Richards & Schoonard. The 1984 Olympic Message System- A case study in system design. Comm of the ACM 30,9, pp 249-260

[Brooke 86]
Usability engineering in Office Product Development. In People and Computers- Designing for usability, Harrison & Monk,Cambridge pp249-260

[Brooks 77]
The computer scientist as "toolsmith" -Studies in interactive graphics. In Information Processing 1977, B Gilcrist (ed) pp 625-634.

[Brown, Sharratt & Norman 86]
The formal Specification of Adaptive User Interfaces using Command Language Grammar. In Proc. Human factors in computing systems CHI'86 pp 256-260 New York ACM.

[Bury et al 86]
Usability Testing in the real world. In Proc. Human factors in computing systems CHI'86 pp 212-215 New York ACM.

[Butler 85]
Connecting Theory & Practice : A Case study of achieving Usability goals. In Proc. Human factors in computing systems CHI'85 pp 93-98 New York ACM.

[Card, Moran & Newell83]
The Psychology of Human-Computer Interaction. Laurence Earlbaum Assoc.

[Carroll, Mack & Kellogg 86]
Interface Metaphors and User Interface Design. In Handbook of Human-Computer interaction, M.Helander Elsevier Science Pub 1988. pp67-86

[Carroll & Rosson 85]
Usability Specifications as a tool in iterative development. In Advances in Human-Computer Interaction H.Rex Hartson Vol 1 pp 1-28 Ablex Pub.

[Davis 83]
User error or Computer error ? Observations on a statistics package. Int. J Man-Mach studies, 19,4 (Oct 83) pp 359-376

[Ehrich & Hartson 81]
DMS- An environment for dialogue management. In Proc. of COMPCON81 Sept Pg 121 IEEE

[Elkerton 88] On-line aiding for Human-Computer interfaces.In Handbook of Human-Computer interaction, M.Helander Elsevier Science Pub 1988. pp345-364

[Foley & VanDam 82]
Fundamentals of Interactive Computer Graphics. Addison-Wesley.

[Fried 82]
Nine principles for ergonomic software. Datamation 28,11 Nov 82 pp 163-166

[Gilb 77]
Software Metrics. Cambridge MA

[Gilb 84]
The 'impact analysis table' applied to human factors design. In Proc Interact' 84. First IFIP Conference on Human-Computer Interaction Vol 2 pp 97-101

[Good et al 84]Good, Whiteside, Wixon & Jones : Building a User-derived Interface. Comm of the ACM 27, pp1032-1043

[Good et al 86]
Good, Whiteside,Spine & George : User derived impact analysis as a tool for usability engineering. In Proc. Human factors in computing systems CHI'86 pp 241-246 New York ACM.

[Gould & Lewis 85]
Designing for usability: key principles and what designers think. Comm of the ACM, 28, pp 300-311

[Hayes 85]
Executable Interface Definitions using form-based interface abstractions. In Advances in Human-Computer Interaction Vol 1 pp 161-189 Ablex Pub.

[Heckel 82]
The Elements of Friendly Software Design. New York, Warner Books.

[Hewett & Meadow 86]
On designing for usability: An application of 4 key principles. In Proc. Human factors in computing systems CHI'86 pp 247-252 New York ACM.

[Kieras & Polson 85]
An approach to formal analysis of user complexity. Int. J man-Mach Studies 22,pp 365-394

[Kieras88]
Towards a practical GOMS model methodology for User Interface Design. In Handbook of Human-Computer interaction, M.Helander,pp 135-157 Elsevier Science Pub 1988.

[Kiss & Pinder 86]
The use of Complexity Theory in evaluating Interfaces. In People and Computers- Designing for usability, Harrison & Monk,Cambridge pp447-463

[Martin 73]
Design of Man-Computer Dialogues. Prentice-Hall

[Mehlmann81]
When people use computers: an approach to developing an interface. Prentice-Hall

[Moran 81]
The Command Language Grammar: A representation for the user interface of interactive computer systems.Int. J man-Mach Studies 15,pp 3-51

[LaLomia & Coovert 89]
Approaches to User Modelling. In Proc. of 21st annual Hawaii Inter. Conference on System Sciences. Vol 2. pp 470-476

[Newell & Simon 72]

Human Problem Solving, Englewood Cliffs, N J, Prentice Hall.

[Ravden & Johnson 89]
Evaluating Usability of Human-Computer interfaces: A practical method. Ellis Horwood.

[Reisner 82]
Further developments towards formal grammar as a design tool.In Proc. Human factors in computing systems pp 309-314 New York ACM.

[Robertson 83]
Goal, Plan & outcome Tracking in Computer Text-editing Performance. Cognitive Science Tech. Report 25, Yale Univ.

[Rubenstein & Hersh 84]
The Human Factor: Designing computer systems for people. Digital Press

[Rubin 88]
User interface design for Computer systems : Ellis Horwood.

[Shackel 86]
Ergonomics in Design for Usability. In People and Computers-Designing for usability, Harrison & Monk,Cambridge pp44-64

[Shneiderman 86]
Designing the User Interface . Addison Wesley.

[Tyldesley 88]
Employing usability engineering in the development of Office Products. In The Computer Journal 31,5 pp 431-436

[Wasserman 85]
Extending transition diagrams for the specification of Human-Computer interaction. IEEE trans. on Software Eng. 11,8 (Aug)