2012

Strategies for the intelligent selection of components

Valerie Maxville

Edith Cowan University

Copyright Warning

Strategies for the Intelligent Selection of Components

Thesis submitted by

Valerie Maxville, BSc (Hons)

for a Doctor of Philosophy Degree

in Software Engineering

EDITH COWAN UNIVERSITY

SCHOOL OF COMPUTER AND SECURITY SCIENCE

11th January, 2012

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

Declaration

DECLARATION

I certify that this thesis does not, to the best of my knowledge and belief:

(i) incorporate without acknowledgement any material previously submitted for a degree or diploma in any institution of higher education;

(ii) contain any material previously published or written by another person except where due reference is made in the text; or

(iii) contain any defamatory material.

I also grant permission for the Library at Edith Cowan University to make duplicate copies of my thesis as required.

Valerie Maxville

11th January 2012

List of Publications

Maxville V (2002) 'Intelligent Selection of Components', in 'Young Researchers Workshop, 7th International Conference on Software Reuse: Methods, Techniques, and Tools (ICSR-7)'

Maxville V, Armarego J and Lam C P (2003a) 'The CdCT Process for Component Selection and Evaluation', in 'Postgraduate Electrical Engineering and Computing Symposium (PEECS)'

Maxville V, Lam C P and Armarego J (2003b) 'Selecting Components: a Process for Context-Driven Evaluation', in 'Asia-Pacific Software Engineering Conference (APSEC)', IEEE Computer Society, pp 456-465

Maxville V, Armarego J and Lam C P (2004a) 'Assessment Methods for Component Selection', in 'Postgraduate Electrical Engineering and Computing Symposium (PEECS)'

Maxville V, Armarego J and Lam C P (2004b) 'Learning to Select Software Components', in Maurer F and Ruhe G (eds) 'International Conference on Software Engineering and Knowledge Engineering (SEKE)', pp 421-426

Maxville V, Lam C P and Armarego J (2004c) 'Intelligent Component Selection', in 'IEEE signature conference on Computer Software and Applications (COMPSAC)', IEEE Computer Society, pp 244-249

Maxville V (2005) 'Knowledge Representation for COTS Selection', in 'Postgraduate Electrical Engineering and Computing Symposium (PEECS)'

Maxville V, Lam C P and Armarego J (2008) 'Supporting component selection with a suite of classifiers', in 'IEEE Congress on Evolutionary Computation (CEC)', pp 3946-3953

Maxville V, Armarego J and Lam C P (2009) 'Applying a reusable framework for software selection', IET Software, 3(5), pp 369-380

Abstract

It is becoming common to build applications as component-intensive systems - a mixture of fresh code and existing components. For application developers the selection of components to incorporate is key to overall system quality - so they want the 'best'. For each selection task, the application developer will define requirements for the ideal component and use them to select the most suitable one. While many software selection processes exist there is a lack of repeatable, usable, flexible, automated processes with tool support. This investigation has focussed on finding and implementing strategies to enhance the selection of software components. The study was built around four research elements, targeting characterisation, process, strategies and evaluation.

A Post-positivist methodology was used with the Spiral Development Model structuring the investigation. Data for the study is generated using a range of qualitative and quantitative methods including a survey approach, a range of case studies and quasi-experiments to focus on the specific tuning of tools and techniques. Evaluation and review are integral to the SDM: a Goal-Question-Metric (GQM)-based approach was applied to every Spiral.

A range of contributions were made over seven Spirals. Spiral 1 delivered a component specification template, swvML, to support the selection process. There is no standard for specifying components - a known template is a prerequisite for automated component selection. In Spiral 2, the CdCE Process was formulated: created to be flexible, repeatable and suited to automation and tool support. The Process has been evaluated through case studies and is supported by tools and procedures developed as part of the research study. In addition, a pattern for software selection was derived from the Process, allowing the reuse of the generic approach, with an implementation tailored to the organisational environment.

Context was a central concept in the investigation implemented as non-functional selection criteria in the shortlisting and as context-based tests in the evaluation. With the specification template and process in place the focus turned to strategies for selection: shortlisting using classifiers; enhanced data representation; evaluation of components and interactive decision support. Each strategy delivered one or more novel contributions and addressed the research problem. The use of decision tree classifiers is novel in component selection and avoids aggregation of results while providing an understandable justification of decision. The metadata for a component repository has been integrated with a knowledge base to allow greater use of the semantic possibilities of the selection criteria. Candidate components are evaluated based on an automatically generated test suite which is adapted to each candidate. Metrics for evaluation target functionality, adaptation effort, test performance and context-based tests. The decision support tool, ClassifierSuite, provides a visualisation of the selection criteria to allow analysis of the criteria and shortlists. Real world data was used in case studies throughout the investigation.

The work has potential to impact professional practice in the way software is characterised, selected, and how evaluations are carried out. The decision support tool may be applied to other data and decision visualisation tasks. Future work to address additional repositories and wider trials will confirm the impact on component selection practice.

Acknowledgments

To my supervisors, Chiou Peng Lam and Jocelyn Armarego, for being so patient and supportive of my journey. Thanks for providing a true master/apprentice experience that went way beyond the research I was doing. Not just supervisors, you are mentors, role models, friends and really great people.

To all the staff at Edith Cowan University and Murdoch University - the support has been wonderful. Thanks to the students and colleagues who feigned interest and encouraged me.

To my boss, Andrew Rohl, for his good humour and acceptance of all those leave applications. To my friends at iVEC, the IEEE WA Section and the ACS WA Branch - the gentle prods and total absence of negativity were most appreciated.

And to my family and friends who never doubted and always offered to help. Especially to Chrissy, Elaine and Jackie. Wow.

For my three little birds - my sweet children who don't know a world where Mummy isn't doing a PhD. Yes, we can have a holiday, and a puppy, and sleepovers... a life!

And to my wonderful parents who have always given unconditional support to my education, my career and my happiness.

I dedicate this PhD to my sweet Mother, you put more into my PhD than anyone - by taking such loving care of my three kids for as long as your health allowed it. I wish you were here.

For Mum.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces the research problem, providing an introduction to component based software engineering (CBSE) and summarising the current issues. In brief, the problem to be investigated is: *can strategies be developed to support the selection of software components?* The approach taken explores four research elements using the Spiral Development Method. Due to the nature of the problem, and the scope of the research elements, a theoretical framework incorporating many domains has been used. This is illustrated and described along with an overview of the terminology used in this thesis, based on the relevant literature and working definitions for this research. The contributions of the work are the result of the development of a specification template, selection process and strategic tools to support the selection task.

1.1 Rationale

The driving force behind research in software engineering is to create more reliable systems in an estimable amount of time (IEEE, 1990). CBSE addresses this need by reusing code that has been developed and tested previously, reducing the amount to be written for a new system: it involves developing systems partially or completely from pre-existing components (Bachmann et al, 2000). This investigation focusses on the selection of third party components to be incorporated into component-intensive systems.

The potential benefits of CBSE are to reduce development and testing time, and create more robust and easily modified systems (Lucredio et al, 2004). However, the decision to use CBSE needs to be made early in the development of a system: it requires a fundamental change in process (Comella-Dorda et al, 2004) as it can create additional

constraints (e.g. choice of programming language or software architecture) and affects the decomposition of the problem at design time.

There are currently a number of issues for developers building systems from components. It may be difficult to find a component to match the exact needs of the system under development (Brereton and Budgen, 2000). In addition, a component itself, while having the required functionality, may not integrate well with the rest of the system (Oberndorf et al, 2000). This may be due to a difference in control models, or be the result of mismatches between the assumptions and provisions of the component and its environment (Garlan et al, 1995). In addressing some of these issues, standard environments (or frameworks) for components have been developed, and continue to evolve (Garlan et al, 2009) through COM/CORBA to Service Oriented Architectures (SOA) and web services. Broker web sites assist in the distribution and acquisition of components, providing facilities for discovering and purchasing a variety of components (Christiansson and Christiansson, 2004).

As availability of third party components becomes more widespread, developers need to manage the selection of the 'best' component to meet the requirements of the target system.

It is important to have structured and justifiable processes for the selection of externally produced software in order to maintain quality standards (Kotonya and Hutchinson, 2005). When selecting from a repository, this will be based on its supplied specification (often through metadata) and some testing. To ensure that required standards are attained and no weaknesses are added into the system, components need to be tested both separately and in context (Weyuker, 1998). The selection process requires a facility for comparison, where components are characterised using a common template (Cechich et al, 2006). These descriptions should provide a mechanism to make maximum use of the available data, including the semantic relationships between identifying elements of the template.

Selection is a manual task in most cases. To allow these processes to scale, some level of automation and tool support is required and strategies developed to add objectivity to the tasks within the process (Ruhe, 2002). For an automated process, it is necessary to both formalise the description of the components and to be able to describe the requirements to facilitate the searching of repositories.

Kotonya and Hutchinson (2005) state that the selection process must be documented as it includes key design decisions, while a repeatable process lets the application developer revisit the selection as the system evolves: vendors may update or withdraw support for their products adding to the maintenance effort required to choose updated or alternate software (Basili and Boehm, 2001).

There is evidence that component selection is hard and also holds potential for project risk. To improve quality and documentation, a process is needed. The discussion in the literature indicates a need for a repeatable, justifiable, structured approach to component selection which can be automated.

1.2 Statement of Problem

Although there are many published processes for component selection (described in Sassi et al (2003)), research indicates that they are not widely used in industry (Li et al, 2005). While selection of components has been shown to have the most impact on risk in component-based development (CBD) (Port and Chen, 2004), developers are still approaching the task in ad-hoc ways, with one difficulty being the acceptance of new processes into an organisation. Rifkin (2003) advises taking a staged approach and including flexibility within new processes to adapt to the local culture. In addition, there should be a net gain to the developers for using a process: faster, easier, better results. Thus processes must not only deliver these, they need to communicate the value to the users (Rifkin, 2003).

As noted previously, one of the main issues with using third party components is in integration, with mis-matches only becoming visible at execution time. Within the evaluation, it is important to test the candidate components in the target context, as any other testing does not show the potential problems (Weyuker, 1998). A complete evaluation needs to include the functional (executable) and non-functional (contextual) aspects of the component under test, since both are important in the choice of suitable components (Beus-Dukic, 2000).

Addressing the issues identified in the literature, a CBSE selection process should therefore be based on a standardised description of components available and a process to facilitate comparison. Some parts of the selection process can be automated or

accelerated, and others may benefit from offering decision support to the developer.

Component selection takes place in the context of a wider software development project. Decisions made in this global context affect the requirements used in the selection. For this investigation, the global context is abstracted to avoid unmanageable co-optimisations. The selection criteria are assumed to have been determined in readiness for the selection to begin.

This project develops a specification and process, then works with these to explore strategies to assist selection. To provide a quality CBSE process requires repeatability, transparency and metrics that allow valid comparisons to be made. To apply a CBSE selection process to real world tasks, it is necessary to have automation and tool support to allow scaling to large repositories and a range of candidates. Automation can also provide artefacts that will allow the repeatability and reuse for system evolution.

The problem being addressed in this research is:

> *What strategies and techniques can be developed to support the selection of third party software components?*

To explore the research problem, a general form of selection process is modelled to identify where strategies can be of benefit. At a conceptual level, the actors in the selection process are: component developers, brokers, application developers and quality assurance personnel. The first two actors provide components for development, while the application developer evaluates them and incorporates the selected components into the system under development. Quality assurance personnel are interested in the process: documentation, justification and repeatability.

Use cases have been developed to increase understanding of the problem under research (Figure 1.1). The initial use cases were used to identify key aspects of the context for software selection to aid the development of the research elements.

The key use case is **Select Component**, involving the application developer. Other use cases could be within or beyond the focus for the project depending on the aspect of the selection chosen. While it is possible that the application developer can influence the development of components (Morisio and Tsoukiàs, 1997), a disconnect between the marketplace and the individual developer is assumed in this work. The research problem is then to consider the application developer view of selecting components with

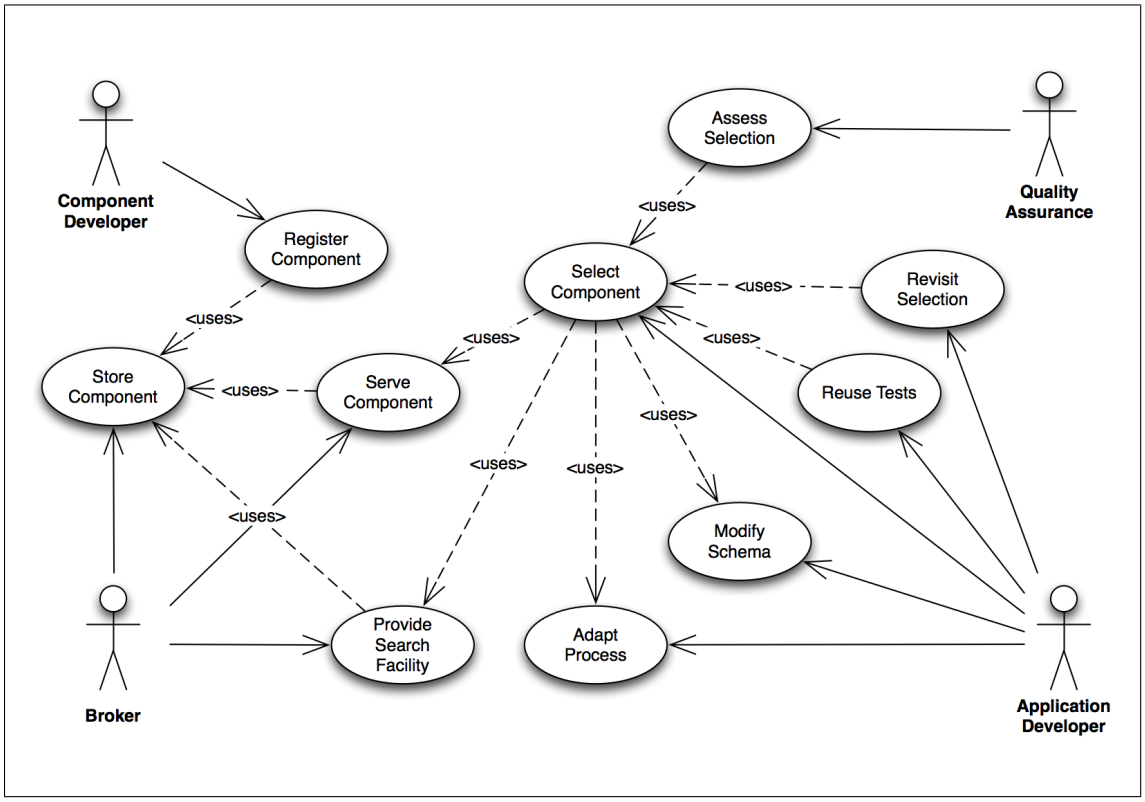


Figure 1.1: Use Case Diagram

an awareness of the importance of context and the reliance on discovery mechanisms.

The four research elements listed provide an overview of how the investigation has been approached. Each is also further described below:

Research Element 1 *Development or extension of a template for the specification of components (**RE1**: Development of template)*

Research Element 2 *Development of a process for the selection of software components (**RE2**: Development of selection process)*

Research Element 3 *Investigation of and implementation of strategies for the short-listing and evaluation of suitable software components (**RE3**: Strategies)*

Research Element 4 *Evaluation of the effectiveness of the template, process and strategies via case studies (**RE4**: Evaluation).*

The Spiral Development Model (SDM) (Boehm, 1988) has been applied for the development of strategies, their implementation, trial and evaluation. The Research Elements

drive successive iterations (Spirals) of the overall investigation - the focus of each Spiral moves through the four Research Elements. A mapping of Research Elements to Spirals is provided in Table 1.1.

Spiral	**Topic**	**Research Elements**
1	Specification	1 (4)
2	CdCE Process	2 (4)
3	Strategies I - Classification and iteration	3 (4)
4	Strategies II - Data representation and ontologies	3 (4)
5	Strategies III - Testing and evaluation	3 (4)
6	Strategies IV - Classifier suite	3 (4)
7	Case Study	4

Table 1.1: Mapping of Research Elements to Spirals

RE1 : Development of template. This element requires the consideration of templates used in industry and the literature to provide a realistic and usable specification of a component. Such a template needs to fit in with practice in Software Engineering and CBSE. It also needs to consider the description of components as electronic resources, drawing on standards used in information management. While much of the recorded information can be considered metadata, this project is also concerned with component behaviour. This requires the ability to describe the required functionality, clearly and formally, ready for automated use in tools.

RE2 : Development of selection process. There are a growing number of component and commercial off the shelf (COTS) selection processes in the literature. These are considered with respect to the goals of this research: to develop a process that is intuitive, iterative, repeatable and suited to automation. Third party software components are written for a specific context, and assessing their suitability must take the target context into account.

RE3 : Strategies. The main aim of this project is the development and evaluation of strategies for component selection. The specification and process developed in **RE1** and **RE2** allow the exploration of strategies. When the project was proposed, the expectation was that the strategy would include artificial intelligence and testing.

RE4 : Evaluation. Although evaluation is part of any research project, in this case, evaluation throughout the investigation is given additional focus. As each Spiral of the

project builds upon the previous, it is important to evaluate each Spiral and associated strategy before beginning the next iteration. Goals and win conditions follow key themes throughout the evaluation, binding the outcomes into a coherent research element. A final Spiral evaluates all strategies and solutions for the project as a whole to round out RE4.

As the aim of this project is to develop strategies and techniques to support the selection of third party software components, the research elements progressively focus on aspects of these strategies and, through the application of SDM, are evaluated and refined.

1.3 Delimitation and Limitations

This study developed strategies for working with commercial/public software components. The limitations of the study relate to the information available about these. On the level of individual items of software, inconsistent information may result in software being deemed unacceptable on the basis of omissions in the documentation, rather than suitability to the purpose. This is a real world issue, and illustrates the difficulties involved in working with third party software documentation. It was also difficult to gain access to data sources describing large collections of software.

The use cases for the overall selection process are a useful way to indicate the scope of the project (see Figure 1.1). The first six use cases are in scope directly or indirectly (see Table 1.2). Actors involved in these use cases are application developers and quality assurance personnel. Use cases of interest, but outside of the scope of this project, are those for developing and brokering components.

Delimitations were identified in an iterative manner, as the project progressed. To allow focus on the component selection process itself, the global context has been generalised. Thus the software project, development team and organisational aspects are not taken into specific consideration – they are not the target of the strategies investigated. The selection process expects the requirements for a specific context to have been defined and available as an input to the selection process.

The target audience for the tools and strategies are expert users who are developing CBSE applications. In view of the above limitations on data availability, Spirals 1 and

Use Case	Actor(s)	Relevance
Select Component	Application Developer	In scope
Revisit Selection	Application Developer	In scope
Reuse Tests	Application Developer	In scope
Adapt Process	Application Developer	In scope
Modify Schema	Application Developer	In scope
Assess Selection	Quality Assurance	Supported
Create Component	Component Developer	Out of scope
Register Component	Component Developer	Out of scope
Serve Component	Broker	Out of scope
Provide Search Facility	Broker	Out of scope

Table 1.2: Use Cases for Component-based development

2 worked with five repositories and a manual collation of data. As noted, the tools and strategies that have been developed are aimed at software developers, and require more than a novice understanding of the problem, software testing, eXtensible Markup Language (XML) and the Z notation. XML is applied as it has become the data interchange standard and is widely used for documenting electronic resources. Automation of test generation required a description of the required component behaviour. Z notation was selected on the basis of standardisation efforts, appropriate tool support, suitability to the task and relevant knowledge within the project.

As the research scope is third party software, source code is not expected to be available. The software testing was therefore restricted to black-box testing techniques.

After the completion of Spirals 1 and 2, further delimitations were identified. To avoid manual compilation of data, repository owners were approached to provide datasets. The commercial component broker sites used in the first spirals were not able to release their data. freshmeat provides an interface to the metadata for over 41,000 projects which suited the needs of the project's case studies. This describes open-source software applications, not specifically software components. As the selection approach can be generalised to applications as well as third party components, the dataset was considered adequate to show the validity of the strategies used.

1.4 Theoretical Framework

A range of research areas are used to understand and derive solutions to this research problem. Understanding the problem and justifying the research is based on Software

Engineering literature and industry trends in reuse and CBSE. Of primary relevance are the Software Engineering areas of CBSE, verification and validation of software systems, document standards and formal methods. CBSE itself draws on software reuse within Software Engineering and shares much with COTS software use.

The topic map (Figure 1.2) shows the relationships between the topics addressed by this project. In the diagram, topics below the line are directly applied in the thesis, and those above the line provide historical background. Connections between parent and sub-topics are indicated as arrows. The fields of study relevant to the thesis are grouped in vertical swim lanes: software engineering, information management and artificial intelligence. Where more than one field includes a sub-topic, both are linked to the sub-topic. These topics are now considered in terms of the research elements.

RE1 draws on existing component characterisations, knowledge management and documentation standards to develop the component specification template. The intended use of the template required consideration of testing and formal specifications to feed into the test generation activity. The implementation of the template follows standards for documentation (XML) and treats the specification of the component as a special type of electronic resource.

With the template available, **RE2** involves the development of a process that uses the template. Existing processes for component evaluation and selection were considered, along with their limitations and issues. Wider techniques from software engineering and COTS selection are also relevant to the development of a usable process. For maximum reuse through abstraction, pattern concepts can be applied to the selection process.

RE3 draws on a wide range of areas to develop strategies to assist component selection. The component specification template makes use of formal methods for the behavioural description of components (Z notation). Software testing techniques for automated test generation and the inclusion of context (environment and usage) are applied in the testing strategy. These directly address CBSE issues. The CdCE[1] Process and tools are a knowledge based system where data representations, classification and ontologies are used to support shortlisting and evaluation. The selection is assisted by artificial intelligence techniques, which is extended to a classifier suite in later Spirals of the project. Evaluation of components utilises verification and validation techniques to

[1]Context-driven Component Evaluation

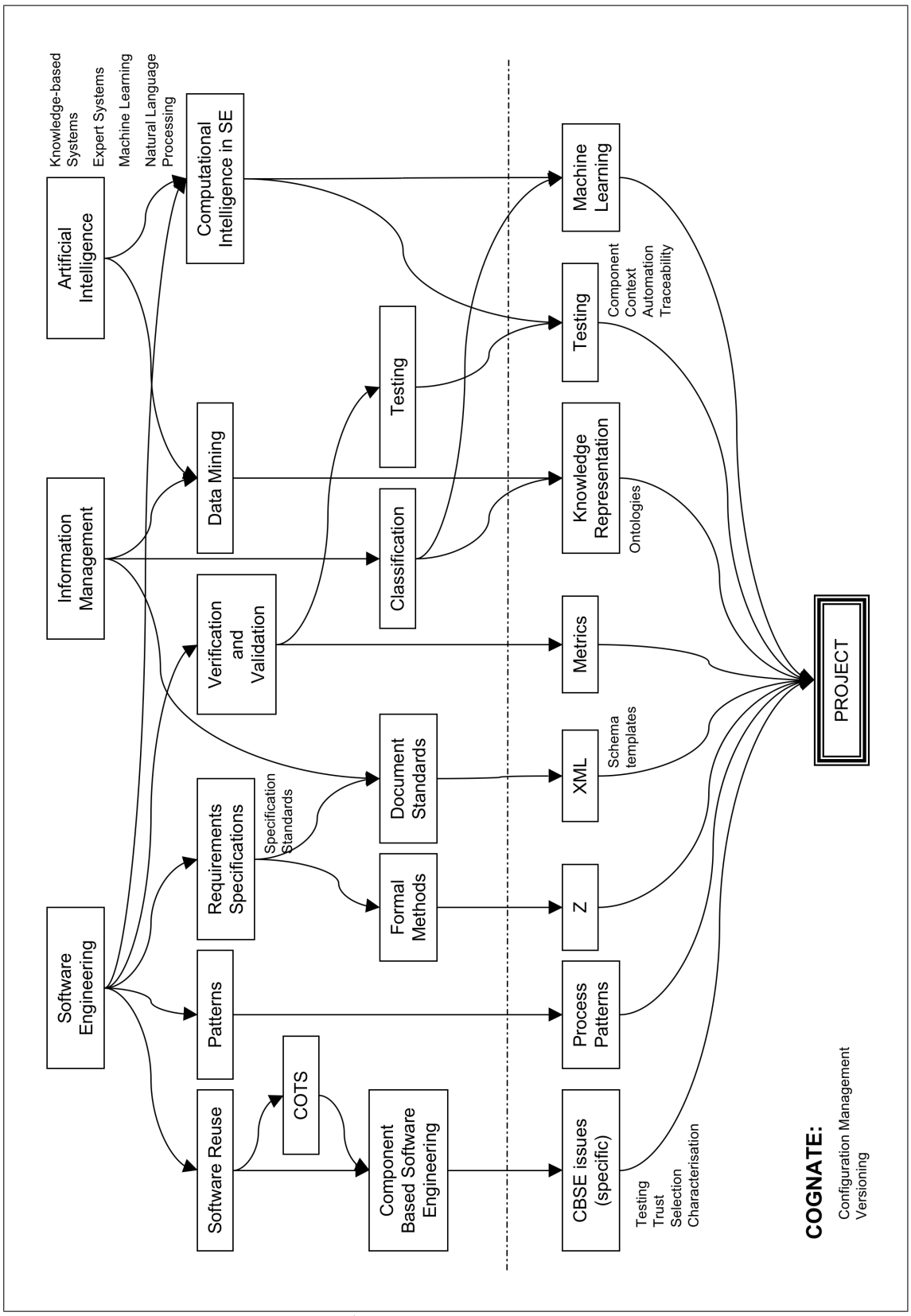


Figure 1.2: Topic map

develop appropriate metrics.

The evaluation for **RE4** is provided throughout the project by means of the review sector of the SDM, addressing a consistent set of goals and win conditions. The case study (Chapter 10) provides a means of exercising (and evaluating) all aspects of the process, compiling the assessment knowledge from **RE1**, **RE2** and **RE3**, combined with new learning from the application of the full Process to a realistic problem.

The theoretical basis of the work is described in greater depth in the literature review (Chapter 2).

1.5 Definition of Terms

This investigation centres on the selection of software components. Many definitions exist for the term **component**. These may imply a certain level of documentation (e.g. formal specification for in-house) or certification to provide a level of trust. To maximise the applicability of this research, the definition used is intentionally broad:

Component: 'A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties' (Szyperski, 1998).

This work addresses third party components, those developed by external organisations as opposed to in-house components. Third party software is often referred to as COTS software:

COTS: Commercial Off The Shelf - software that is developed externally and is available to be acquired.

Many variations of ...OTS exist, and the terms COTS and components are often interchanged. In this thesis open source software (OSS) is included when referring to COTS, although they are not usually commercial products.

To enable discovery of components, they are registered in a repository. This may be a list of software made available by the vendor, or an independent repository hosting software from a range of developers. Component brokers are responsible for listing software components in a repository. In the scope of this study, third party applications are

treated as components[2].

The strategies for software selection are the high level ideas for approaching the research problem. These can be implemented as processes, procedures, applications and specifications.

Two categories of people are considered in this work: actors and stakeholders. Actors are defined as those involved in the selection process and are identified in the use case diagram: component developer, component broker, application developer and quality assurance. While stakeholders have a significant overlap with actors, they include additional people who have an interest in the outcomes of this study. The key stakeholders considered for this work are:

Component Developer: Develops or modifies components for third party use.

Application Developer: Incorporates a component into a system, also referred to as the **User**.

Component Broker: Provides access to components either commercially or as a service.

Quality Assurance: Assesses the quality of the application development.

Academia: Concerned with the merits, methods and outcomes of the research project.

This work requires the characterisation of a component in terms of a component specification. The characterisation is based on a set of attributes, with selection criteria being the required values on those attributes. These are used for shortlisting candidates and for specifying the required component. The requirements for the component are described through the ideal component specification. The context of a component includes information about the target environment (e.g. frameworks and standards). Key terms in this selection activity are:

Attribute: An element in the characterisation of a component.

Selection criteria: Attributes and required values for suitability.

Shortlist: The set of candidate components matching the selection criteria.

Specification template: Definition of the attributes and related types in a specification.

[2]Due to the use of the freshmeat repository

Component specification: Documentation that is included with a component to assist the developer's selection and integration of a component.

Ideal component: An application developer's **abstract** requirements for a third party component.

Ideal component specification: A **concrete** description of the ideal component encoded using the template developed during this project.

Candidate component: A component that has been shortlisted for evaluation against an application developer's requirements (via the ideal specification).

Target system: System into which a component is to be integrated.

Context: The environment of the component, described in the ideal component specification. Includes the target system, interfacing components and applications and dependencies. Also includes development languages, frameworks and usage/risk information for testing purposes.

The CdCE Process has been developed for component selection, defining eight Steps, their inputs, outputs and internal procedures, and related techniques and tools. The strategies are developed within and across the process steps. Components identified during the filtering or shortlisting phase are candidate components. They are then evaluated against criteria through static or dynamic methods. Given the results of the evaluation, a ranking can take place to inform the final selection.

Further definitions are listed in the Glossary in Appendix A and stakeholders are described in more detail in Table 3.3.

1.6 Assumptions

Assumptions made in this work relate primarily to the component marketplace. To use the process, access to component repository data is required. Although there was difficulty with gaining access to commercial data, it is likely that, in the long term, repositories for third party components will make their data available. The repository data used in this work is from an open source project site. Although it would not seem to match the intentions of the work, the data made available was of a similar structure to that used in commercial repositories. The dataset was also larger (by a factor of 3) than the largest commercial repository.

The experimentation and case studies in this thesis were carried out by the researcher. It is assumed that the experience level of the researcher is on par with the expert users defined as the audience to the work.

1.7 Research Approach

The research methodology for the investigation is the Spiral Development Model (SDM) (Boehm, 1988). The SDM is an iterative approach to developing solutions and is usually applied to software development. Each Spiral includes four phases: determine objectives; plan the next iteration; development and test and identify and resolve risks.

This investigation involved seven Spirals, each focussing on different elements of the research problem and providing evaluation information for the project. Spiral 1 develops a component specification to address **RE1**. In Spiral 2, the selection process is developed, as required by **RE2**. Spirals 3-6 develop and implement strategies for **RE3**. Spiral 7 is in the form of a case study to evaluate the process, strategies and techniques for **RE4**.

Data for the study is generated using a range of qualitative and quantitative methods. For **RE1**, a survey approach is taken to developing the template. For **RE2**, a survey of processes is used to develop the base CdCE Process, which is then applied to a manual case study. A range of case studies were used in **RE3**, with quasi-experiments to focus on the specific tuning of tools and techniques. A comprehensive case study is used in Spiral 7 to give data for **RE4**. Evaluation and review are integral to the SDM and are approached using a Goal-Question-Metric (GQM)-based approach (Basili, 1992), applied to every Spiral. A more detailed description of the methodology is provided in Chapter 3.

1.8 Contributions

Specific issues in component selection were targeted across the study by a range of strategies. These were outlined in the four research elements and their investigation has resulted in the following key contributions of the study:

- Component specification template (C1)
- Repeatable, semi-automated process for component selection (C2)

- Support for context (C3)
- Use of classifiers for selection (C4)
- Data representation enhancements (C5)
- Testing and evaluation approach (C6)
- Classifier suite for decision support (C7)
- Pattern for software selection (C8)
- SDM as a research methodology (C9).

C1: swvML is a component specification template including functional, non-functional and context information, enabling the automation of parts of the selection process.

The CdCE specification template draws together aspects of characterisations from the literature and those used in existing repositories. The swvML template includes functional and non-functional attributes, including context-related information for the components. A transformer from freshmeat to CdCE was developed to illustrate the conversion process from other repositories, including thesaurus functionality for converting terms. The ideal specification extends this component specification to include behavioural details defined using Z notation. The specification thus has the information required to describe a repository of software, to define the requirements for the ideal component, and to give the basis for tests to be generated. The CdCE template and supporting tools form the foundation for development of the CdCE Process, enabling the automation of selection tasks.

Developed in Spiral 1, documented in Chapter 4.

C2: The CdCE Process is a repeatable, semi-automatic process with artefacts to support documentation, reuse and quality assurance.

The CdCE Process again draws on the literature to provide an original, straightforward approach to selection and evaluation of components. Tools have been developed to automate activities within the process to enable the consideration of larger repositories of software, optimising the use of developer's time. There is an emphasis on context-based assessment, using the ideal specification to drive the Process. The Process includes strategies for refining the ideal specification, and a decision-support tool, helping the user to get the best results from the available data. Through the automation of the Process, artefacts are created that can be retained and reused in future searches.

This provides repeatability, supports revisitation of the selection for system evolution or quality assurance and can be used as part of the rationale for selecting a particular component.

Developed in Spiral 2, documented in Chapter 5.

C3: Context has been supported throughout the CdCE Process through the attributes included in the specification and context-based testing.

Literature related to component testing highlighted the importance of context. To support selection, non-functional attributes were included to allow matching of components to the target context. In addition, context schemas in the formal specification support the representation of directives for usage, performance, stress and reliability testing. Four metrics have been developed to carry the results of the context-based tests.

Documented in Chapter 4 and Chapter 9.

C4: Machine learning classifiers are used for shortlisting in a novel application of computational intelligence.

After consideration of the current techniques for comparing components, a different approach was developed to avoid some of the issues with Weighted Sum Method (WSM) and Analytical Hierarchy Process (AHP). Initially an expert system (Jess), classifier (C4.5) and artificial neural network were trialled, with C4.5 chosen as it provides a decision tree to justify the reasoning for the selections made. In a novel approach, the training data is generated from the ideal specification and allows for supervised learning. Training the system creates a predictive model which captures the 'requirements' of the component and is a reusable artefact of the selection process. A key benefit of the classifier approach is that each attribute is taken on its own merit and is not obscured through aggregation. Applying classifiers to component selection is a novel approach and this project indicates it can assist with filtering large number of components into a manageable shortlist.

Developed in Spiral 3, documented in Chapter 6.

C5: Attributes in the specification are represented by five data-types, an ontology, a distance matrix, and a range of matching transformations.

The tools and supporting files for the process have been developed to maximise the use of given information. Five datatypes allow comparisons to be more attuned to the data, including the semantic closeness of some terms as represented by an ontology. Five

transformations are provided to differentiate handling and give a stronger understanding of the particular datatype. The variation within the transformations include:

- match/no_match (boolean/DB query approach)
- loosening restrictions to include near misses
- distance measures for the closeness of terms
- keyword handling for long text fields
- abstraction via ontology.

This improves on existing techniques for searching repositories as most search interfaces will do a text-matching search on all or some of the information they hold. This approach allows searching for a specific term against a specific attribute. The CdCE-Transformer conversion process makes it possible to look at information from a variety of brokers using one tool/interface with the data in a consistent format. The conversion process includes a thesaurus to standardise terminology (e.g. the classifier will not miss a match because the description uses 'Win98' instead of 'Windows 98').

The handling of missing data was also enhanced to provide predictable behaviour based on user settings. Data representation enhancements through datatypes and ontologies, transformations and comparison operations have improved the performance of the classifier (percentage of correct classifications). This has enhanced the relevance and recall of shortlists and further demonstrated the applicability of the classifier approach.

Developed in Spiral 4, documented in Chapter 7.

C6: Evaluation is driven by nine metrics, the values coming from evaluating adaptation and functional and context-based test results.

The approach to testing and evaluation requires the generation of tests that can provide a valid comparison - using the same tests on each component. Equivalence partitioning, based on the Z notation specification within the ideal specification, is used to generate the test cases and substitute supplied test data into the abstract test cases. These abstract tests are adapted to the candidates, which also produces input for calculating metrics for functional fit and adaptation effort. These are the first four metrics - functional fit (`FFIT`), functional excess (`FEXS`), adaptation effort (`AEFT`) and testing fit (`TFIT`). The tests are provided as abstract XML and need to be ported to the local harness/environment, then executed.

The results of the tests are evaluated to calculate the remaining five metrics where appropriate - test result (TRES) along with the context metrics for performance, reliability, stress and usage tests (CX_P, CX_R, CX_S, and CX_U respectively). The developer's preferences for these metrics are gathered from the ideal specification. Evaluation again uses an alternative approach to aggregation and leads into the reporting of all results based on the XML artefacts created in the CdCE Process. In the event the results are not satisfactory, iteration is accommodated to refine the behavioural specification and/or adjust expectations on the evaluation metrics. The automation within the Process, and the ability to adapt to the available software, reduce the additional effort required for iteration.

Using a single source of abstract tests is a novel approach to the dynamic evaluation of components: it standardises the assessment and ensures it is firmly grounded in user requirements.

Developed in Spiral 5, documented in Chapter 8.

C7: The ClassifierSuite is an interactive visualisation tool to support shortlisting and enhance decision-making.

As the shortlisting process has been automated, it is possible to create a graph of the combinations of criteria by dropping one non-mandatory criterion at a time and the shortlist that results from each set. This graph can be traversed to find an optimal criteria set to identify a shortlist of components. The approach condenses the iteration, removes some of the heuristics and subjectivity, and provides a better overall view of the impact of including or excluding criteria.

The ClassifierSuite is an effective decision support tool which allows the application developer to explore and analyse potential shortlists and make use of its features to support decisions. This can make the shortlisting faster, more accurate and assists the user in dealing with more complex and larger criteria sets.

Developed in Spiral 6, documented in Chapter 9.

C8: The CdCE Process is also a process pattern for software selection, with flexibility for a range of instantiations.

The design of the CdCE Process was intended to provide flexibility and reuse. While this is clearly possible at the task and system evolution level, the Process can also be viewed as an implementation of a more abstract framework for selection. As a framework

it describes a three-phase process of filtering, evaluating and ranking items, which can then be populated with tools and procedures for selecting software or other resources. This pattern can then be used to provide consistency and repeatability in many tasks that need to extract and select items from repositories.

Documented in Chapter 5.

C9: The investigation has applied the SDM as the research methodology.

As this project required the building of theory, strategy and tools, all of these could be addressed within the SDM, rather than just the software development. This is a novel application of the SDM, according to Barry Boehm[3]. The SDM has proven to be very useful in staging the investigation, controlling risk and providing review and planning for subsequent iterations.

Documented in Chapter 3.

1.9 Thesis Structure

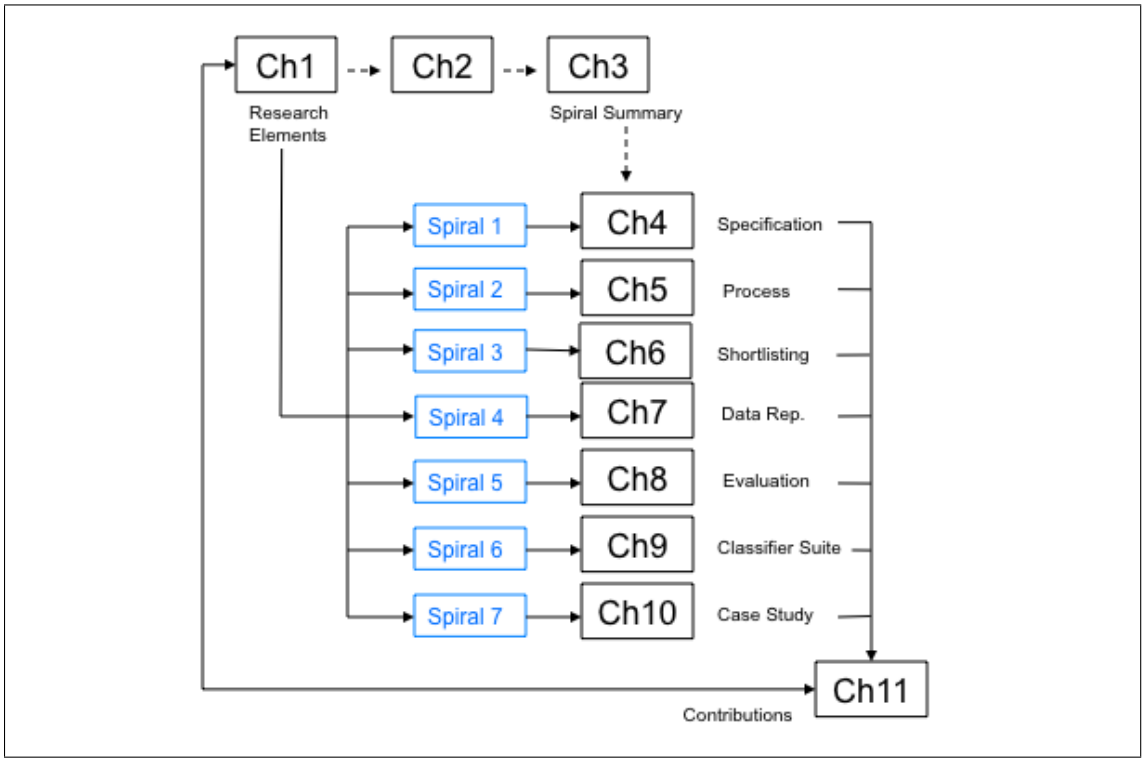


Figure 1.3: Map of thesis showing relationship between Chapters and Spirals

[3]Personal communication at ICSE 2009, Vancouver, Canada

Figure 1.3 provides an overview of the investigation and the structure of this document. Chapter 2 is the review of literature. In Chapter 3, the research procedures are discussed, closing with a summary of the Spirals of development throughout the investigation. The project includes seven Spirals, six for development of strategies and a final Spiral for a case study and evaluation. These Spirals map into seven 'results' chapters. Each of the Spiral chapters includes a discussion of relevant theory and context, and ends with a review and evaluation. All of these results are considered in Chapter 11 with respect to the overall project goals.

1.10 Summary

This Chapter has provided the rationale for the investigation of the stated problem:

> *What strategies and techniques can be developed to support the selection of third party software components?*

Four research elements guide the project direction: template, process, strategies and evaluation. The approach and techniques span the theory of software engineering, information management and artificial intelligence. The overall theoretical basis for the work is provided in Chapter 2.

The investigation has been structured using the Spiral Development Model, providing an iterative, adaptive and risk managed methodology. The novel use of the SDM is one of the contributions of the project and is described in Chapter 3. In addition, the component specification template, CdCE Process and process pattern provide a framework for software selection and evaluation. A range of strategies have been successfully implemented: classifiers for selection; enhanced data representation; support for context; testing and evaluation techniques; and the ClassifierSuite for decision support. Each of these represent a novel contribution of the investigation.

Chapter 2

Review of Literature

This investigation explores strategies for selecting software for systems incorporating third party components. The use of artificial intelligence and knowledge management to assist software engineers is a key thread through this review as it may provide guidance in identifying potential strategies for the investigation.

In Section 2.1 the background and history of component based software engineering (CBSE) is described, including its roots in software reuse. Following from this, Section 2.2 considers the selection and evaluation of components, drawing out the issues that exist for component and application developers. Section 2.3 provides a critique of the literature with respect to the issues raised and considering industry use of components. Based on the critique, Section 2.4 describes the approach selected for the research project described in this thesis. Although many processes for selection exist, uptake has been low, which may impact on the quality of component based systems. This may be due to a need for more intuitive, flexible approaches, with automation and tool support for application developers.

2.1 Background

Software Engineering grew alongside hardware engineering for the early computers of the 1950s (Boehm, 2006): the differences between software and hardware, particularly maintenance and modification, became apparent soon afterward. The 1960s saw the emergence of large, complex software systems that could not be approached by simply scaling up existing methods of development. The subsequent problems with late delivery of projects, and over budget, difficult to maintain, buggy systems were referred to as the

software crisis (Buxton et al, 1968). The NATO Software Engineering Conference in 1968 brought international experts together to discuss issues, and to consider techniques and methods to solve this crisis. Dijkstra (1972) refers to the software crisis' major cause as the increase in power of the machines, 'when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem'.

These early discussions drew a link between the software crisis and an increasing demand for software, with the number of computers increasing 25-50% per year around 1968 (Buxton et al, 1968), eventually 'slowing' to just below doubling every five years (Petska-Juliussen and Egil-Juliussen, 2009). However, this does not reduce demand for software, as a variety of platforms have emerged including games consoles, mobile devices, virtual machines, cloud computing and websites. Wirth (2008) states that mobile phones have 100 times the power of the biggest computers of twenty years ago.

The increasing demands are not fully expressed by considering only the number of computers. A jump in complexity occurred in the transition from batch to time-sharing systems (Wirth, 2008). Scale has also increased in terms of the numbers of developers - in 1958 a general purpose computer manufacturer employed 50 programmers, increasing to 1000-2000 in 1968 (Buxton et al, 1968). In current day figures, companies such as IBM have 398,455[1] employees worldwide, 49,185 of whom are software developers in the US (in 2008). Beyond the number of computers and size of companies, we have an increase in the scale of software in multiple dimensions - including number of end-users, variety of platforms, code size, parallelism and threading, developer team sizes and geographic distribution of development teams (Maxville, 2009). These factors compound the demands and complexity inherent in software development and indicate a continued need to deal with the on-going software crisis through application and improvement of software engineering. Dijsktra's comment on the cause of the software crisis is just as applicable today, keeping solutions out of the reach of the humble programmer. With the crisis now passing forty years, it may be better to call it the software 'reality' or 'challenge'.

The IEEE Computer Society defines software engineering as

[1] personal correspondence, 2010

1. The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
2. The study of approaches as in (1) (IEEE, 1990).

The useful results of forty years of software engineering experience and research are summarised as the Software Engineering Body of Knowledge (SWEBOK) (Abran et al, 2004). A key contribution of SWEBOK is the clarification of language to assist communication, as many of the difficulties in Software Engineering are social (Weinberg, 1971). SWEBOK language will be the convention throughout this discussion.

In reviewing what has taken place in the history of software engineering, it is important to bear the context in mind. The concepts of Information Hiding (Parnas, 1972) and Abstract Data Types (Liskov and Zilles, 1974) were vital to the construction of systems by large groups of people, and came at a time when data was 'common' and goto statements were rife (Dijkstra, 1968). The changing computing environment impacted on software, in the new challenges of shared systems over batch, in new languages, tools, operating systems and hardware.

Brooks (1987) states that 'the complexity of software is an essential property, not an accidental one'. A solution to this essential complexity put forward by Brooks is the 'Build versus Buy' option where the economies of scale can allow vendors to increase the quality of their software by spreading the cost among multiple users. The software reuse approach also enables quick prototyping and assembly of systems on a platform of middleware/API, making fast iteration and rework easier than with a fully developed system. Reuse is discussed in more detail in Section 2.1.2.

2.1.1 Software Development Process

From the IEEE definition, the key to Software Engineering is a systematic approach, or process, for software development, operation and maintenance. A process provides a structure to which to attach techniques such as those in the SWEBOK Knowledge Areas, as each project or organisation sees appropriate. Curtis provides a list of the goals and benefits of modelling software processes (Curtis et al, 1992):

1. Ease of understanding and communication: requiring a process model containing

enough information for its representation. It formalises the process, thus providing a basis for training.

2. Process management support and control: requiring a project-specific software process and monitoring, management and co-ordination.
3. Provision for automated orientations for process performance: requiring an effective software development environment, providing user orientations, instructions and reference material.
4. Provision for automated execution support: requiring automated process parts, co-operative work support, a compilation of metrics and process integrity assurance.
5. Process improvement support: requiring the reuse of well-defined and effective software processes, the comparison of alternative processes and process development support.

All of these are important to this study, with item 4 of particular interest.

Royce (1970) published one of the earliest process models to coordinate the development of complex systems, commonly referred to as the Waterfall Model, which provides an idealised progression of the development of a system. Royce's paper indicated a difference with other fields of engineering by including iteration and feedback in his model. More recent processes and methodologies have iteration as central, including the Spiral Development Model of the 1980s and agile methodologies of the 2000s (Larman and Basili, 2003). Iteration is recognition that the requirements for projects are not known in advance, and that there can be a need to make changes based on the understanding that emerges during a project. These adjustments help to match the system to the user needs. However, iteration must be managed to converge towards a solution, which can be assisted by making key decisions and recording any externalities early to reduce the solution space. Although methodologies and development frameworks have value, Brooks (1987) states that fostering great software designers will assist in addressing system complexity. However, processes often indicate documents and artefacts which aid the development and maintenance teams in understanding the project and decisions that have been made.

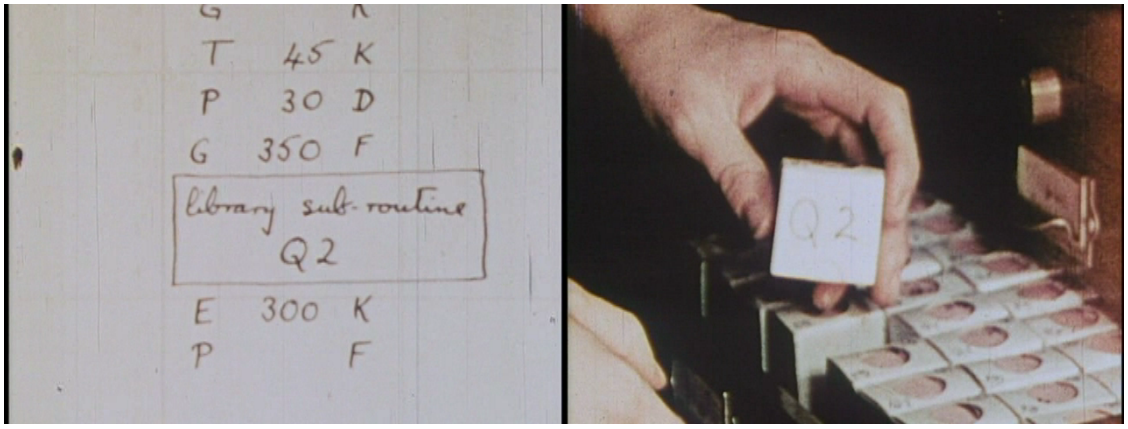
2.1.2 Software Reuse

One means of making the most of the precious resource of great designers is to reuse all, or parts, of their work. Reuse can be opportunistic, or it can be planned for and carried out with intent. In the early 1970s, Parnas wrote of the importance of modularisation and information hiding, enabling design for software reuse when developing software systems (Parnas, 1972). His paper provides examples of how two approaches to decomposing a problem can make a significant difference to the ease with which the resulting system can be updated, understood and reused.

Frakes and Terry (1996) surveys the metric and models for reuse, including cost/benefit, maturity, amount, library metrics, failure models and reuse estimates, while Mili et al (1999) propose a discipline of reuse. Developing code for reuse has been reported as requiring 63% more time than one-off code development (Galorath and Evans, 2006). Lampson (2004) states 1/2 to 2 times the effort for developing code with clear interfaces, increasing to 3-5 times the effort for a reusable component.

The reuse of source or executable code is trivial and was accepted practice from the earliest programmable computers. Grace Hopper reflected that the notebooks she and her colleagues shared when working on the Harvard Mark 1 in 1944 were the beginning of subroutines (Williams, 2004) and hence code reuse. Issues of errors in copying and in the adjustment of values in the target context were noted by Hopper. Later, work on the EDSAC (1949+) reused libraries of subroutines via punched tapes, boxed up and filed in cabinets (Computer History Museum, 2009). These were mechanically copied into the program at the required point (Subroutine Q2 in Figure 2.1) and a 'comparator' was used to ensure the correctness of the copy. A library catalogue was used by the programming committee to match the available subroutines to the requirements of the scientist.

Programming languages, such as C, have been created to include minimal functionality, to be extended through reusable libraries of code (Frakes and Kang, 2005). Although an obvious form of reuse, code reuse now applies to billions of software installations, from operating systems and mobile phones to embedded systems in cars and washing machines. The ubiquitous systems of today are reusing Million Lines of Code (MLOC), with Windows Vista around 50 MLOC (Manes, 2007), Max OS X 10.4 84 MLOC (Jobs, 2006) and Debian 4.0 (full Linux distribution) 283 MLOC (Robles, 2005). Thus the granularity of reuse can be from one to millions of lines of code.



(a) Specification for program including subroutine Q2

(b) Box containing tape for subroutine Q2

Figure 2.1: Subroutine library use for EDSAC in 1949 (Computer History Museum, 2009) [@28 minutes]

Another view on code reuse is the risk of defects being spread across many machines. With the estimated 1,338 million computers in the world in 2010 and a conservative 50 MLOC per machine, there are 67,000,000,000,000,000 reused lines of code in operating systems alone in 2010, with approximately 60% of those being Windows XP[2]. Dijkstra foresaw these risks in 1968: 'The dissemination of knowledge is of obvious value - the massive dissemination of error-loaded software is frightening' (Buxton et al, 1968). The more visible of these defects are vulnerabilities that may be exploited by hackers. An early example is the CERT Advisory CA-1996-26 Denial-of-Service Attack via ping[3] defect, affecting most operating systems. An advisory was issued on December 18, 1996, requiring patches to operating systems worldwide to correct the problem.

Reusing Artefacts

Dimensions when considering reuse include types of artefact, audience, intent and developing for or with reuse. Trust and context are two of the key challenges in the reuse of artefacts.

The concept of reuse is not restricted to code: it can also be applied through architecture, design and processes. Although this can be at an informal level, the discussion here will be in terms of formalisation of reuse, in particular through patterns. A pattern

[2]Source: W3schools http://www.w3schools.com/browsers/browsers_os.asp
[3]http://www.cert.org/advisories/CA-1996-26.html

'describes a problem which occurs over and over in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice' (Alexander, 1977; p. x). This definition comes from architecture and has found strong application in software engineering. In Gamma et al (1995) the Gang of Four (GoF) provided software design patterns as solutions to problems in context. 'A design pattern systematically names, motivates, and explains a general design that addresses a recurring design problem in object-oriented systems. It describes the problem, the solution, when to apply the solution, and its consequences' (Gamma et al, 1995; p. 360).

Alexander's approach to patterns was to allow the 'users' to find the patterns, given some basic building blocks. In a process, the blocks would be the key tasks, which can then be arranged into a generic pattern, the implementation of which can vary. This separates the pattern from its instantiation and allows for adaptation for all or part to match the target environment or application. The patterns allow abstraction of processes and the potential for reuse across application areas.

The documentation of patterns is described in Gamma et al (1995) and adherence to this, or similar templates allows easier communication and comparison. Thus the concept of patterns assists reuse beyond the artefact itself as it helps with communication, describing the context to allow the assessment of the suitability of an artefact and in defining requirements for artefacts. It also assists a pervasive reuse approach, consistent throughout lifecycle, levels of abstraction and team members.

Architectural patterns in software engineering allow for reuse of the structure of systems, including the widely applied layered, client-server and pipe and filter patterns (Shaw and Garlan, 1996). Similarly, the use of patterns enables the reuse of designs at a lower level of abstraction. Many design patterns have been published (Gamma et al, 1995, Cunningham, 2010b) and provide a common language for developers and a shorthand for specifications and evaluation of the resultant code.

Coplien et al (2005) defines process patterns as 'the patterns of activity in an organisation', and are therefore closely related to organisational patterns. In all patterns, a key aspect of the description is the context. An appreciation of context is required for successful reuse of any artefact from processes to architecture, to designs and code - 'at the core to both [all] kinds of patterns is a solution to a problem in context' (Gamma

et al, 1995; p. 3).

Discovery

Reuse can also be viewed in terms of who the artefacts will be provided for or by. This correlates with the intent of the reuse. Individual programmers, or small groups, may apply opportunistic reuse. However, with support, an organisation may move to intentional reuse where libraries and repositories are provided along with procedures and templates for describing and specifying reusable artefacts (Prieto-Diaz, 1991). When developing with reuse, a mechanism must be available to discover existing artefacts. These may be previous design documents, architectural designs, test cases or test data, as well as code. The initial focus of learning software organisations (LSO) was the development of content management systems and other repositories to be searched or mined (Holz and Melnik, 2004). The current focus is in maintenance and flexibility, which must be considered to avoid 'experience cemeteries' (Holz and Melnik, 2004) - a challenge shared by all repositories. Prieto-Diaz (1991) explores the issues of describing reusable software to allow searching of repositories. This and other repository literature is usually confined to in-house reuse where standards can be set for documentation and source code is available for analysis (Fidge, 2002, Rao and Sarma, 2003, Nakkrasae et al, 2004, Stylianou and Andreou, 2007).

Searching through source code is unlikely to provide a match, particularly at the design stage. There is a need to characterise or classify artefacts at a domain or problem level to assist discovery. In Prieto-Diaz (1991), the faceted classification approach allows a variety of dimensions to be represented. More recently, the common approach for repositories is to record metadata in a standardised form, usually XML documents based on DTDs or Schemas. This is particularly helpful as it is compatible with HTML, used for web development, and XML provides the structure for descriptive text. Public repositories available for access and download of software include freshmeat[4], ComponentSource[5] and TuCows[6], and more recently the iPhone Appstore[7]. ComponentSource quotes over 1,000,000 developers using its site worldwide[8].

[4] http://freshmeat.net/
[5] http://www.componentsource.com/
[6] http://www.tucows.com/
[7] http://www.apple.com/iphone/iphone-3gs/app-store.html
[8] http://www.componentsource.com/

Searching for software can utilise a generic search engine (e.g. Google[9]), or may take advantage of an understanding of the fields describing aspects of the artefacts in the repository. Faceted classification, SQL queries and ontologies are applied in repositories to make greater use of the structure and semantics of the metadata (Prieto-Diaz, 1991, Cechich et al, 2006). In Hemer (2003) the approach is to include a formal specification for each artefact, to be able to search an in-house software repository at an abstract level at design time. Repositories are part of a marketplace which is an additional level of separation from in-house reuse. As a market it raises new issues as the code is often executable-only, has licensing restrictions and may not be a perfect match for the target system.

Reuse Marketplace

The market for reusable software was predicted by McIlroy (1968), at which time he envisioned a software component subindustry. This would allow developers access to product lines of black box component to assemble into software systems. Lampson (2004) presented a keynote at ICSE[10] putting forward an argument on the failure of McIlroy's vision for software components. Lampson states three reasons for the failure:

- There's no business model
- Cost to understand and use components
- Components have a conflicting world view.

In the decade since his presentation, the marketplace, platforms and standards have evolved through many advances including web services, grid computing, Service Oriented Architecture (SOA), and open source software (OSS). This has led to a more conducive environment for components and the following discussion may challenge the premise of his talk - 'Only the Giants Survive'.

As demand for new software escalates, software reuse is increasing productivity and reducing cost, with over 99% of code executed at the US Department of Defense coming from Commercial-Off-the-Shelf (COTS) products by the end of the 1990s (Basili and Boehm, 2001). Those who had previously developed complete systems in-house began

[9] http://www.google.com/
[10] Presented at ICSE in 1999 and published in 2004

moving towards COTS-based development: as one example, at the University of Southern California (USC-CSSE) projects were 28% COTS in 1997, but had increased to 60% by 2001 (Boehm et al, 2003). The primary reason given for this change was 'the large increase in the number of COTS products providing application functions' (Boehm et al, 2003). The flipside of this observation is that there is a viable market not only for standalone COTS but also for COTS to build into systems. While the change is not perhaps the realisation of the 'generic' components of McIlroy, there is evidence of the success of components.

With around 90%[11] of the estimated 15 billion computers Windows-based, it can be argued that the software marketplace began with the IBM-compatible PC. This provided a hardware platform that could be cloned, to which a 'standard' operating system (MS-DOS) could be loaded, opening up IBM compatible machines as a target for third party software developers. Software development has been boosted across all platforms by the provision of libraries, APIs and middleware, allowing developers to reuse common functionality and concentrate on building the unique part of their application. The distribution of applications built upon these APIs etc is an example of code reuse at multiple levels, and often forces developers to align to patterns and architectures - thus reusing more artefacts of previous software development.

Widespread reuse is made possible through well defined interfaces and standard operating environments (SOE) including Microsoft Windows, Apple Mac OSX and Linux, who combine to an estimated 99.5% of the market[12]. Virtual machines and browser-based software also contribute to the standardisation of target platforms for developers. The Internet and the World Wide Web have brought standards and protocols that can be targeted, with approaches such as RESTful interfaces becoming popular. Integrated Development Environments (IDEs) enable easy use of reusable libraries and classes, as predicted by Brooks (1987). Large, enterprise level software systems have been developed, with site-specific tailoring taking place after purchase (e.g. SAP) (Comella-Dorda et al, 2004). Another recent area of growth is the availability of portals and repositories, particularly in the OSS space. Distribution via the Internet, rather than through shrink-wrapped products also increases the spread and reuse of software.

[11] http://en.atinternet.com/Resources/Surveys/Internet-user-equipment/Operating-systems-April-2011/index-1-2-7-235.aspx.

[12] atinternet.com survey, based on browser visits to sites, thus doesn't include servers.

Users will always have a new and more complex set of problems ready to continue the software challenge. Examples exist in the large scientific instruments being planned and constructed which rely on Moore's law to assume the availability of faster processors, networks and larger data storage by the time they are commissioned. An example is the Square Kilometer Array (SKA) radio telescope, which will have a processing requirement of 30 petaflops/sec and data storage requirements of 18 petabytes per year (Quinn, 2009). At the front end it will total 7 petabytes of data per sec requiring over an exaflop of processing power per second. The SKA project is designed to use COTS hardware and software, and much of the work will be in developing communications, processing and visualisation software to meet the scale of data involved. COTS is considered to be suitable as it is cheaper and faster than the custom solutions that radioastronomy has used in the past, and much of the science research is moving to collaborative models which require the interoperation of multiple systems around the world. Benholt (2007) discusses the challenges of petascale computing and the opinion that the solution will be based in the Common Component Architecture and promotion of standards for the development of components.

An active and influential movement in the software market is the OSS community, which advocates sharing code for wider reuse and has developed licences to allow for copyright and to define how code can be extended and reused. Although OSS had been considered to be quite different to COTS, it has been shown that developers working with OSS software components generally do not modify the source code, effectively treating them as COTS (Li et al, 2005). Some differences exist, particularly in licensing, but the other commonly raised issues: the money factor, lack of support and reduced security, are considered myths (Di Giacomo, 2005). OSS developers often target common software needs, for example, the development of the LAMP stack, providing Linux, Apache, MySQL and PHP (or Perl/Python) for a very common server platform (Lee and Ware, 2002). Systems are in place to prioritise the next developments in these and related standards groups. Examples include the Internet Engineering Task Force which has a Request For Comments (RFC) system to create new standards for the Internet[13] and the Bazaar[14] approach described in Raymond (1997). In this investigation, we include OSS and Government Off the Shelf (GOTS) software, and will be referring to all of these

[13] http://tools.ietf.org/html/rfc4677

[14] 'Bazaar' as an alternative to the more traditional 'Cathedral' approach.

when discussing COTS - that is, any software developed by a third party (as opposed to in-house).

As discussed, software is increasingly acquired, rather than custom made for in-house use. In some cases, the choice of third party software is straight-forward; however, if there are a range of options, a selection process will be needed.

COTS Selection

The software selection process can range from very information, to an elaborate, intensive process, involving multiple assessors. Bakås et al (2007) surveyed processes used to select enterprise resource planning (ERP) systems and found that analytic network process (ANP), analytic hierarchy process (AHP), data envelopment analysis (DEA), multi attribute decision making (MADM), and utility ranking methods (URM) were applied, along with critical success factors and Frameworks. These approaches are discussed further in Section 2.2.2 since they also apply to Component Based Development (CBD).

However, the literature may not be representative of the way COTS software is selected in industry settings. Robb and Susser (2000) surveyed foreign language instructors on how they chose courseware and found that, although checklists were available in the literature, only 25% of the software was reviewed against a checklist before purchase. For those where the software was still in use at the time of the survey, 78% of respondent indicated they were most influenced by 'Recommendations of colleagues' and only 50% had made the primary decision based on a checklist. The paper indicates issues with context and selector preferences when using checklists. The authors suggest that selection combines recommendations, testing and checklists in a thorough evaluation. Chau (1995) focussed on software selection by owners and managers and found that owners take a more strategic view, considering more factors, giving more weight to technical factors and consider all factors more seriously than managers. The authors suggest different selection frameworks may be appropriate for different roles.

Two remaining issues that affect the uptake and success of reuse are trust and context. The social aspect of trusting another person or organisation's code can be a barrier to reuse. The choice of software for email services or websites is an example where security is of high concern and trust becomes a key factor in selection. A new option of organisational email being provided in the cloud is challenge to trust, as it is not only the software, but

also the hardware and all data which is put into the care of a third party.

At a more technical level, the context targeted for the reuse of an artefact must be similar enough to avoid mismatches between the expected and provided environments. This context could include the operating system, virtual machine, libraries, communication/network availability, hardware and may be version-specific on any of these. Many of the items in the context match those that would be expected in the characterisation of software. The metadata for COTS software is what an initial decision will be based on and thus anything required for the successful running of the code should be represented in the metadata for the artefacts and required by the repository. In the description of a design pattern, this information would be listed under Problem, where the context indicates when a pattern is applicable (Gamma et al, 1995).

Using COTS requires trust, particularly if not able to access the source code through certification and testing and needs to overcome potential issues of mismatches in context. Therefore, trust and context become more critical in Component-based Software Engineering and will be discussed further in Section 2.2.6.

COTS and Components

The terms COTS and components are often interchanged. However it is important to disambiguate the two terms. COTS-solutions differ from COTS-intensive systems in that the former are often stand-alone systems which integrate with business processes and can be loosely coupled to other software (Comella-Dorda et al, 2004). In COTS-intensive systems, the COTS and in-house components of a system are assembled at build or run-time to create a system. Component-based development of COTS-intensive systems impacts on the software development process and we now discuss the differences that need to be considered.

Design for reuse and design for CBSE require a different approach where, after the analysis, the designer has to identify commonality across various applications. To create reusable components or modules, each element needs to have a high level of independence from the rest of the system and have a clear interface for collaboration (Schmid, 1999).

COTS systems may be bought as complete, standalone systems (e.g. a word processor) or may be COTS components incorporated into larger systems to accommodate part of the required functionality. COTS development can involve customisation, and

many packages have facilities for extending functionality and interfacing to other COTS products. Carney and Long (2000) places COTS and MOTS[15] as two axes of a matrix: source and modifiability (Figure 2.2). The sources may include in-house, legacy, contracted and commercial items. These may or may not be able to be modified internally via customisation, parameterisation or not modifiable. Examples include SAP[16] and Talent2[17], large, generic finance and payroll systems which are customised for each site. In-house developers then work on glue-code to interface with other information systems (COTS or in-house).

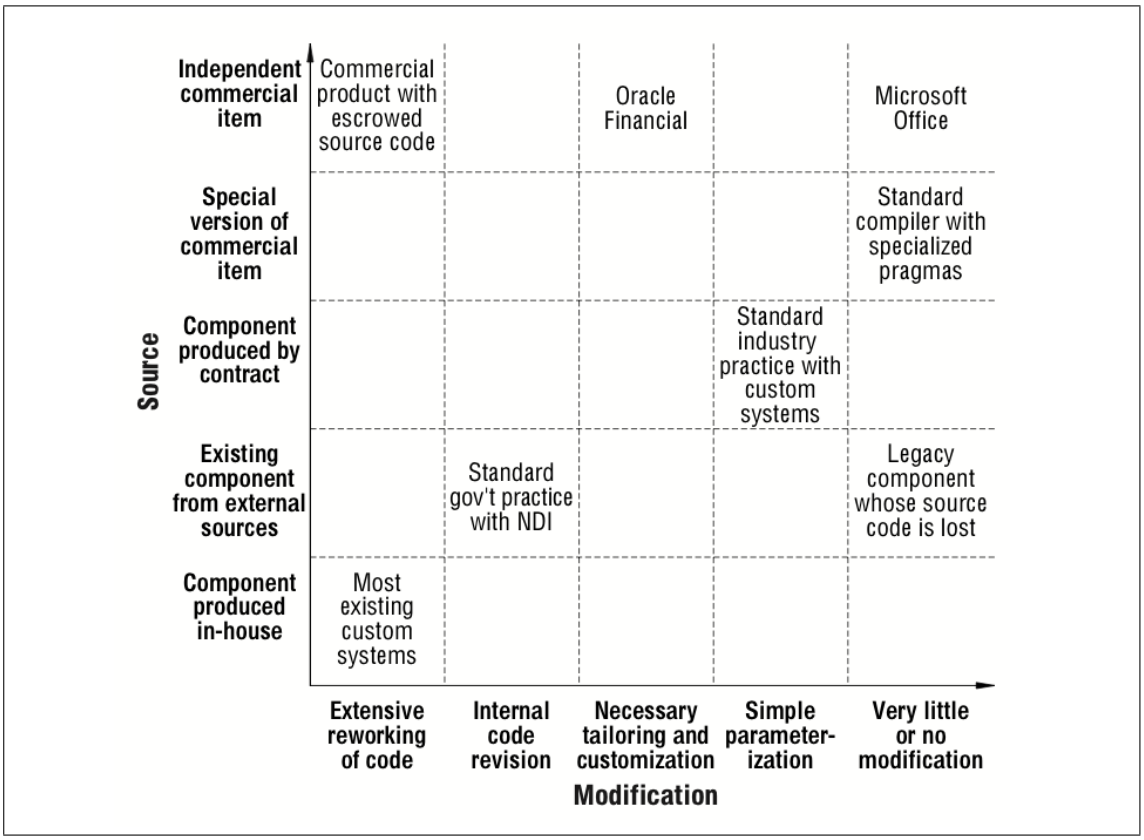


Figure 2.2: The source and modification axes, with sample items located for comparison (Carney and Long, 2000)

2.1.3 Component Based Software Engineering

Developing systems with a focus on composing existing parts is referred to as Component-Based Software Engineering (CBSE) (Bachmann et al, 2000) - it is the application of best

[15] Modifiable Off the Shelf products
[16] http://www.sap.com/
[17] http://www.talent2.com/Products–Services/Payroll-solutions.html

practices to developing Component-based Systems (CBS). The aim, when developing these systems, is the economic construction of reliable software systems from trusted components (Morris et al, 2001).

Some see reusable components as being any piece of code or design that can be reused - at all stages of software development (Yacoub et al, 1999). In other cases, a minimum component would refer to routines or classes provided in libraries that can be included in new systems, and are integrated at compile or link time (Meyer, 1999). This type of reuse has been available (and required) in languages such as C, C++ and Java for many years. Meyer (1999) lists five levels of abstraction for components: functional, casual grouping, data, cluster (framework) and system (e.g. COM/CORBA). At the other end of the spectrum, a component may be defined as a stand-alone module which can be hooked into a system by referencing the module's public or external interface (Schmid, 1999). These components are integrated into the system at run-time. The developers may or may not have access to the source code for each component they use (Bergner et al, 1999, Udell, 1994), or may be able to pay extra to the vendor for the component source[18].

Independent of integration time and availability of source code for the component, most definitions of the term agree that a component must be a replaceable unit and serve a clear distinct function (Yacoub et al, 1999). Szyperski (1998) states that integral to the specification of a component is the framework it is developed for, referring to frameworks that provide communication and other services including CORBA and .NET.

Szyperski and Messerschmitt (2003) gives an update on his 1998 definition, stating five criteria for components:

- Multiple-use
- Non-context-specific
- Composable with other components
- Encapsulated
- A unit of independent deployment and versioning.

Szyperski (1998) lists examples of early components including Netscape plugins,

[18]Offered by http://www.componentplanet.com/

Quicktime plugins, Terminate and Stay Resident (TSR) code and Visual Basic components, along with many parts of operating systems (e.g. pipe/filter). There are many definitions of components: an example of compiled definitions and related discussion is at Cunningham (2010a). New technologies and philosophies for software development may be included in the component space - including OSS, web services, cloud appliances and specific programming languages. Challenges to the binary component definition come from popular languages such as Python, where a scripting language can be used to develop complete systems, and a range of components to extend them (e.g. Zope and the Plone learning management system). Python developers are sharing code as packages[19], and as well as downloadable plugins for Plone[20], Zope[21] and other systems.

Other related developments are Grid Computing and SOA. Grid Computing has primarily come from the eScience and eResearch communities to facilitate the sharing of resources including supercomputers, data storage and large, shared scientific instruments (Bote-Lorenzo et al, 2004). To facilitate this, a framework for communicating components has been developed for interoperability across heterogenous environments (Malawski et al, 2007). With the addition of security and administration functionality, the Grid is moving to Cloud Computing in commercial applications, with much of it built upon components. The Amazon Elastic Compute Cloud (Amazon EC2) is one example[22].

The SOA solution has grown on the Internet as an eCommerce solution, often aligned to BPEL (Business Process Execution Language). In this case, the reused software sits in a virtual component model, made up of black box components with well defined interfaces and protocols. SOA differs from traditional CBSE as the third party components are integrated at execution time through orchestration of web services, not as part of an internally controlled assembly. The affinity between SOA and CBSE is strong and there are groups working to bridge the two approaches, for example, bridging the Fractal component model with the Service Component Architecture (SCA) (Collet et al, 2007). Silaghi and Strohmeier (2003) propose a process that integrates CBSE, Separation of Concerns, Model Driven Architecture and Aspect-oriented Programming, regarding each as a layer of abstraction to pass through when developing a system.

[19] http://pypi.python.org/pypi
[20] http://plone.org/products
[21] http://old.zope.org/Products/all_products
[22] http://aws.amazon.com/ec2/

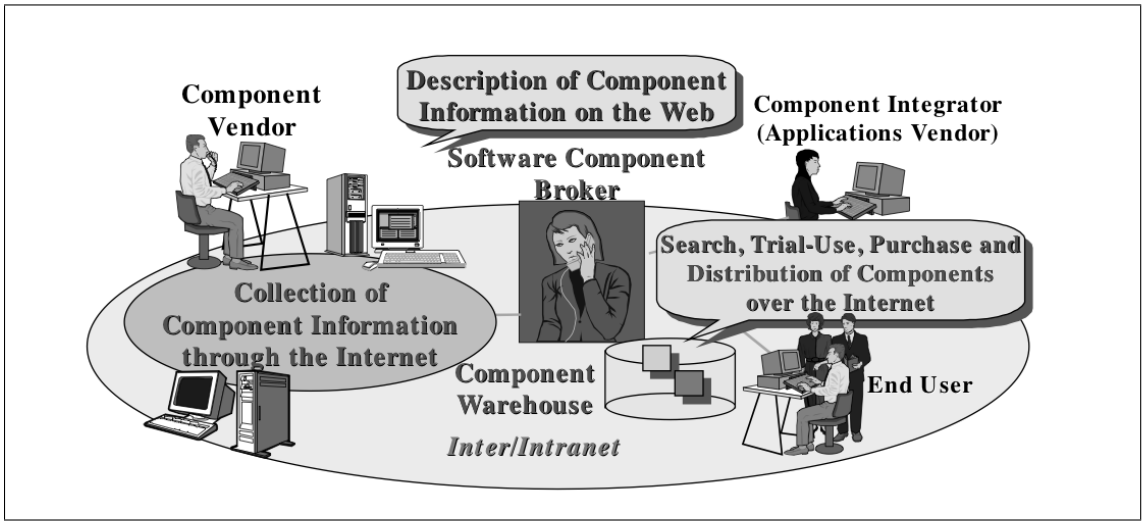


Figure 2.3: Vendor-Broker-Integrator model (Aoyama, 1998)

Aoyama (1998) wrote of a future of developers purchasing components from third party vendors, modifying the code or interfacing the modules to create a complete system. In that model (Figure 2.3) vendors, brokers and integrators are all part of the process to develop systems for end users. Since then, component repositories such as ComponentSource have appeared, while other software repositories, such as freshmeat and SourceForge track software and projects which can include component development. Some of the new architectures that support CBSE are CORBA, COM+, EJB, .NET, Fractal and Web Services, with many providing the facility to replace a component while the system is running. In this document, and throughout this project, we have used a broad definition of component, as given by Szyperski (above, and explicitly stated in Section 1.5) and including, but not targeting, Grid and SOA.

2.1.4 CBSE in the Software Development Lifecycle

Although many software processes exist, there is a fundamental change required to engineer a system from COTS components in the 'simultaneous exploration of system context architecture and design and available software in the market' (Comella-Dorda et al, 2004). A COTS-based approach entails a carefully reasoned selection of potential components through a comparison of options, features and tradeoffs, in the context of the system under development. This is illustrated as the four stages in Figure 2.4 through which a custom path is chosen, based on feedback on design, architecture and component availability.

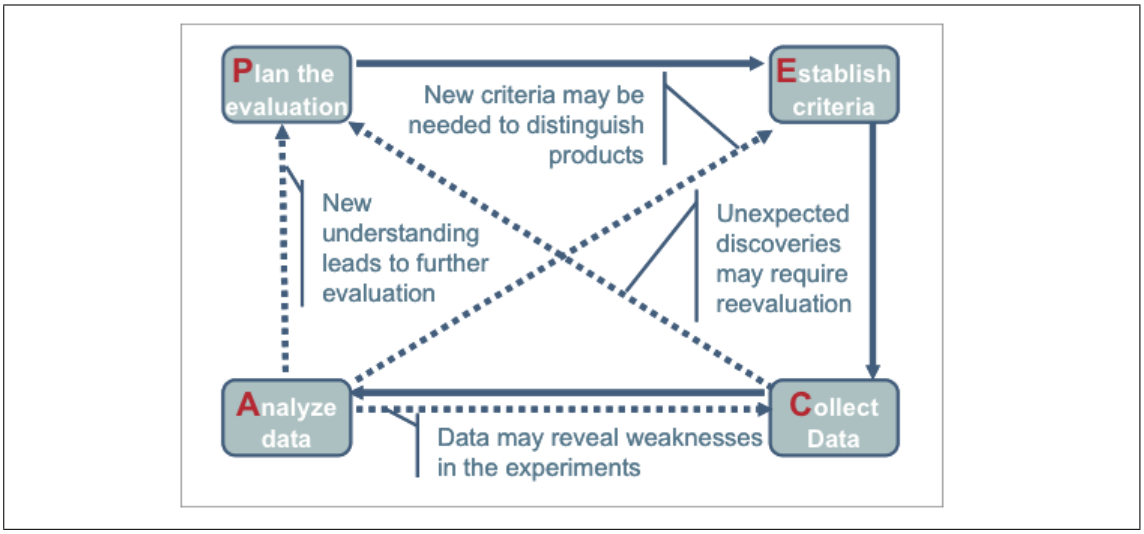


Figure 2.4: PECA process (Comella-Dorda et al, 2004)

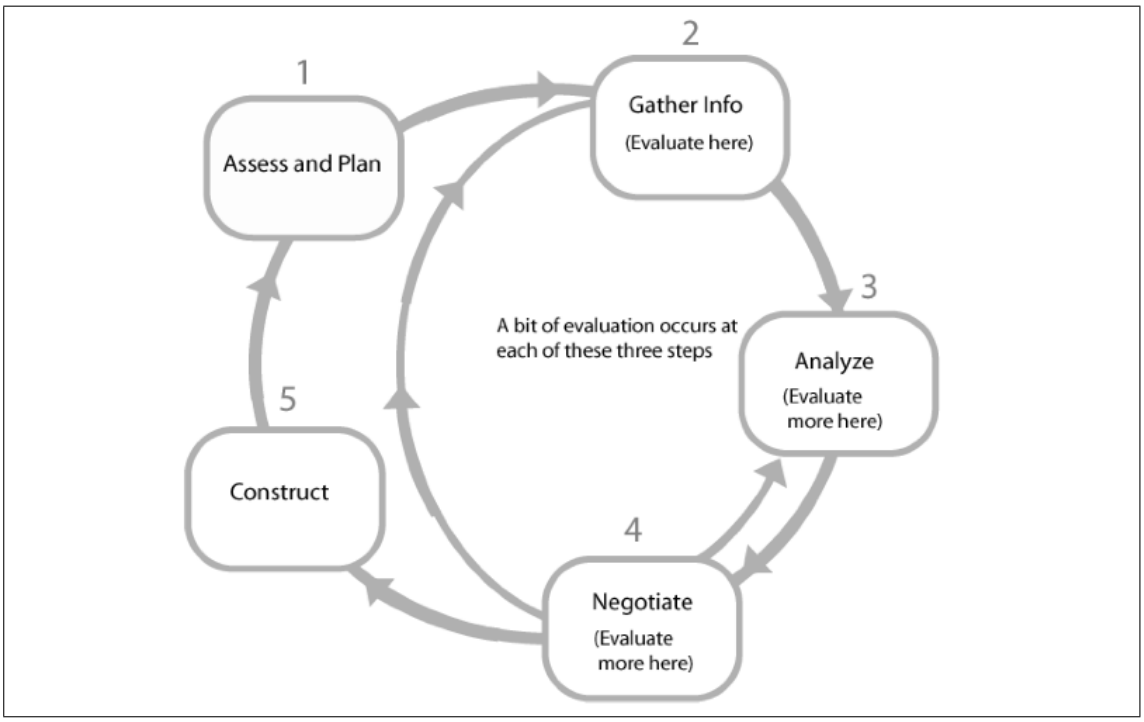


Figure 2.5: A good CBS Process (Comella-Dorda et al, 2004)

As Comella-Dorda et al (2004) discuss, a conceptual CBS process has five steps:

1. Assess and Plan
2. Gather Information
3. Analyse
4. Negotiate
5. Construct.

The related figure (Figure 2.5) shows how component selection fits into the wider software development process with the inner loop of COTS product evaluation and the outer loop is the wider system development. CBD involves iteration and refinement which eventually converge towards a solution. The developers will be creating glue code to integrate the components, and core code to drive the system or add functionality not provided by the components.

A number of processes have been published for CBD, particularly in the selection of components. One benefit of a formalised process is that it can provide a framework for documenting decisions: Kotonya and Hutchinson (2005) states this documentation is necessary in preparation for the evolution of the system. The seminal paper in the area is the description of OTSO (Off-The-Shelf-Option) (Kontio, 1995). OTSO was not specific to components, and set the trend for early processes in the use of GQM and AHP to define criteria, weightings and calculate the comparisons between components.

In Morisio and Tsoukiàs (1997), a process is described for COTS-based development in the Software Engineering Laboratory (SEL) at NASA. From an analysis of previous projects, a strawman process was defined which included four phases:

1. Requirements Analysis and COTS Identification
2. Design and COTS Selection
3. System Integration and Test
4. Technology Update and System Maintenance.

At the process level, the SEL process has parallel activities for COTS-specific and non-COTS aspects of the development. Figure 2.6 shows an actual COTS development process, showing the vendor's continued involvement in the system development. More commonly, a third party component is taken as-is, independent of the vendor.

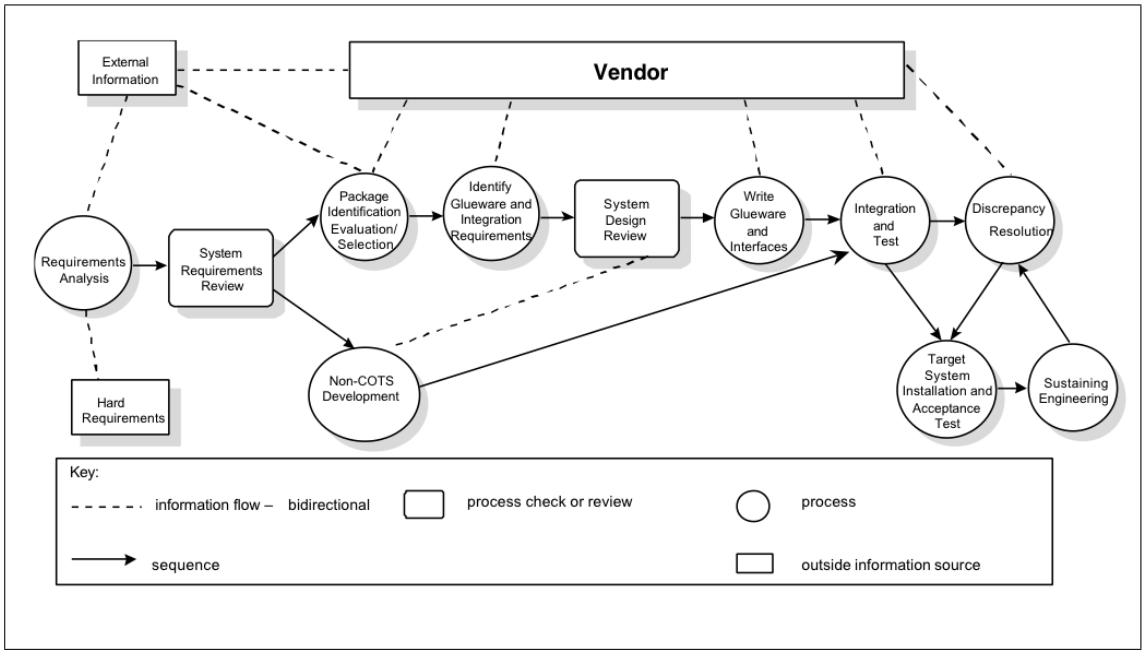


Figure 2.6: The actual COTS process (Morisio and Tsoukiàs, 1997)

Kunda and Brooks (1999) stated four main issues in selection and evaluation of components: lack of a well-defined process; problems with the definition of evaluation criteria; the black box nature of components and rapid changes in the marketplace. They propose STACE (Social Technical Approach to COTS Software Evaluation) as a systematic approach that can evaluate both COTS products and the underlying technology, using socio-technical techniques to improve the software selection process (Figure 2.7). The search for alternatives is conducted through market surveys, Internet search, product publications and sales promotion and computer fairs and shows, with an implied screening process. The evaluation uses AHP and a process flowchart shows the main steps involved (Figure 2.8).

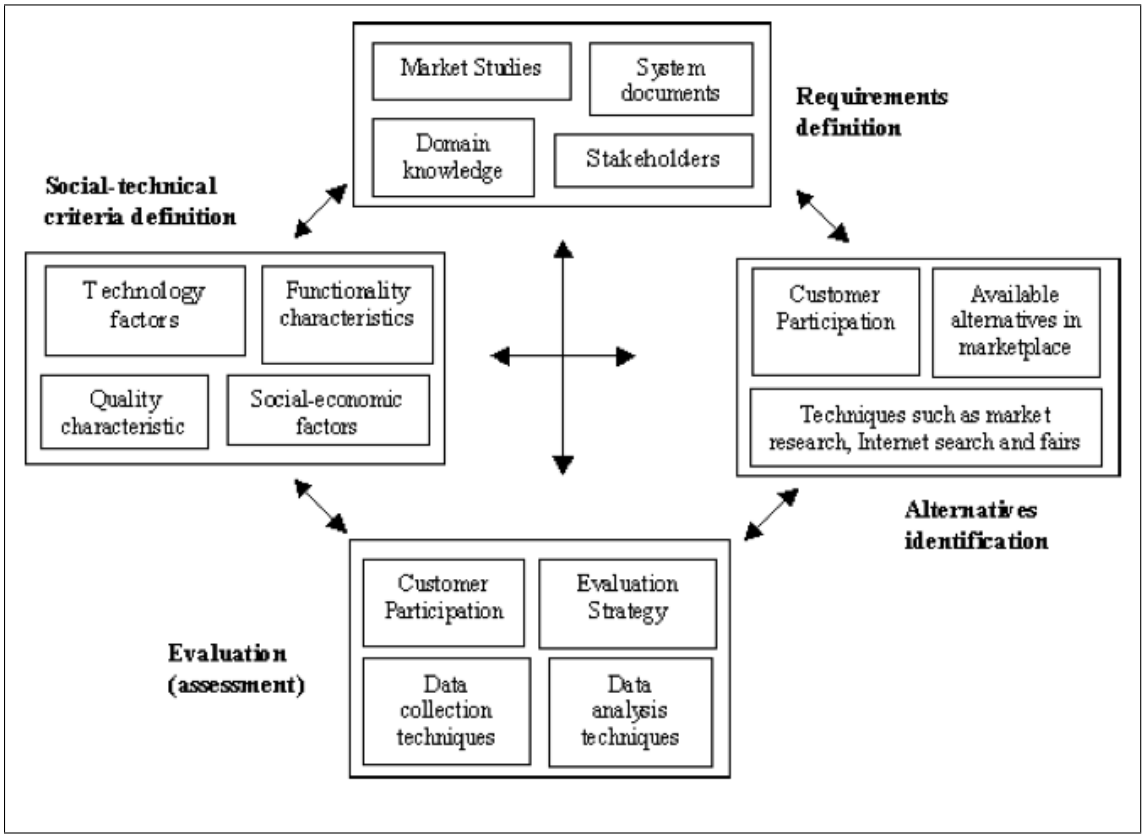


Figure 2.7: STACE framework (Kunda and Brooks, 1999)

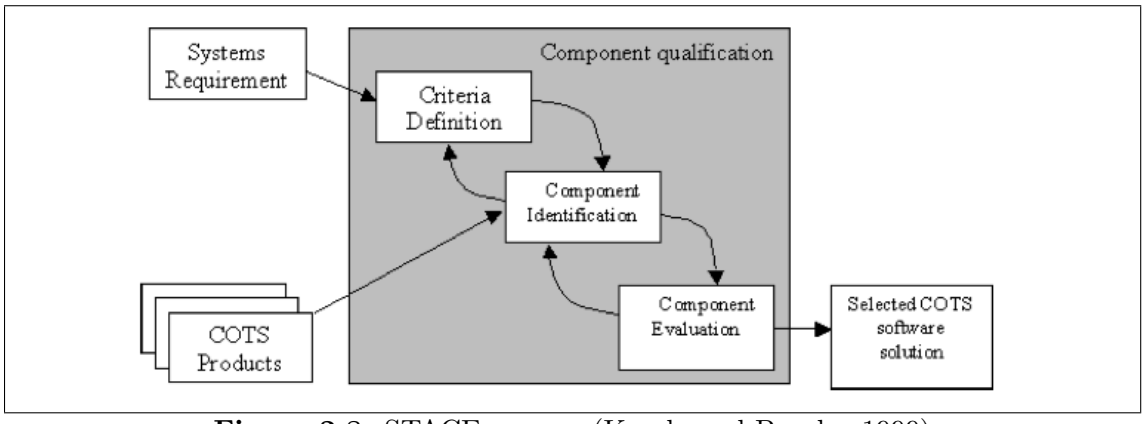


Figure 2.8: STACE process (Kunda and Brooks, 1999)

Many processes have included iteration, including Comella-Dorda et al (2002), Kotonya and Hutchinson (2005), to allow for adjustments to the requirements based on the available components. An iterative approach results in an increase in acquired requirements while reducing the pool of potential components to a set of candidates. Ncube and Maiden (1999) illustrate this refinement in Figure 2.9. As the selection process proceeds, the template has more requirements and the number of products under consideration decreases. The types and depth of tradeoffs depends on the number of potential components. Where there are few to choose between, the developer may need to concede more, whereas if there is a good range of components close to specification, the developer may even need to tighten the requirements.

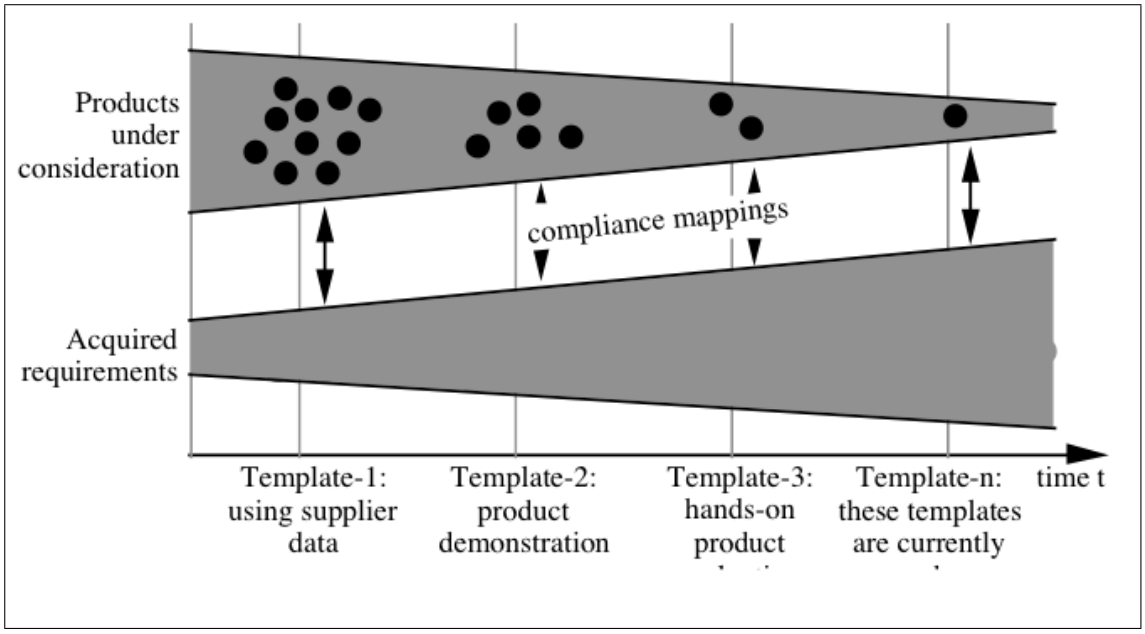


Figure 2.9: PORE templates for iterative selection (Ncube and Maiden, 1999)

Although there are a range of processes in the literature, a study in Norway has found that most companies do not follow a formalised process for COTS selection. This echoes the study on COTS selection, where only 25% used checklists (a basic formalisation of software selection) (Robb and Susser, 2000) - indicating users/developers may not be seeing the need for structure and traceability, or that it may not fit their workflow. Navarrete et al (2005) considers a number of published CBD processes in terms of agility, however finds that the suggestions to 'improve agility in general, [but] may collide a bit with particular [agile] principles'.

The discussion will now focus on the issues raised in CBSE, throughout the software

development lifecycle (SDLC). These include: lack of uptake; lack of process; black box challenges; and rapid change in the marketplace. There are five generic activity areas which are seen in all software lifecycles: Requirements, Design, Construction, Testing and Maintenance (Abran et al, 2004). Testing, or more broadly, verification and validation, span the other activities. There are two overlapping lifecycles - one for the component (component developer view) and one for the system under development (application developer view). The focus is on the latter.

Requirements

The first phase in the SDLC includes high level decisions and requirements definition. In Morisio and Tsoukiàs (1997), it is suggested that a level of COTS selection should take place in this first phase. The developers need to consider the 'make or buy' decision, which will require flexibility in requirements, and willingness to depend on vendor(s) (Morisio and Tsoukiàs, 1997). In addition to the system requirements, there are some COTS-specific requirements that need to be defined to facilitate component acquisition (Comella-Dorda et al, 2004):

1. Architecture and interface constraints: e.g. middleware
2. Programmatic constraints: technical skill-base of team
3. Operational environments: maintenance of the system after launch
4. Stakeholder expectation: (e.g. users may see a different interface from external components).

The requirements will need flexibility to allow for trade-offs (Morisio and Tsoukiàs, 1997). Comella-Dorda et al (2004) suggest denoting requirements as 'hard' or 'negotiable'. PORE (Ncube and Maiden, 1999) concentrates on the iteration and trade-offs that progress a COTS project towards implementation, defined through templates (Figure 2.9). In COMPOSE (Kotonya and Hutchinson, 2005) the requirements are split into functional and non-functional with functional becoming 'services' and the non-functional becoming 'constraints'. The non-functional constraints include what may be called 'ilities' (Voas, 2001). Kotonya and Hutchinson (2005) note the possible interdependence of these requirements, which may impact on selection.

Clark and Clark (2007) highlight cost and licensing as issues that affect the cost of COTS-intensive development. Decisions are needed setting limits on costs and defining the types of licences that will be acceptable. With some software the costs may not be visible (e.g. advertising; vendor lock-in for pro versions; use of data by vendor) (Clark and Clark, 2007). Licensing may be the trickiest part as the non commercial, no derivatives, mix with open source, is an area of complex legalese for all users. Creative Commons[23] (CC) licensing has been ratified in countries around the world - providing more stability when incorporating CC software into a system. Developers need to be mindful of inclusion of components that have licences restricting sales or profit of derivative code.

Tran and Lin (1999) discusses the difficulties of applying CBSE to projects that lack stable requirements. In their experiment, they were open to using COTS for up to ten building blocks within their system. They found issues with the tightly coupled circular relationship between the product selection and system requirements definitions process. The product sets changed three times in nine months, rippling through the rest of the system. They provide six strategies to deal with the Cyclic requirements-product dependencies problems: early evaluation and selection of COTS; capture product information in each iteration to enable easy re-evaluation; separate product limitations from technical limitations; prioritise system requirements; subcontract vendors for key infrastructure components; and document dependencies between requirements and components.

A number of approaches have been put forward to deal with the trade-off of requirements: loosening or tightening requirements in response to component availability and wider system development. Ruhe et al (2003) apply the Quantitative Win-Win method, while Kotonya and Hutchinson (2004) use viewpoints. Alves and Finkelstein (2002) has a goal-driven approach, using two attributes: desirability and modifiability. Ncube and Maiden (1999) uses parallel requirements and market research for a tightly scheduled COTS selection. The PORE approach uses iteration, knowledge engineering, feature analysis, MCDM and design rationale.

Key points: When working with components, decisions need to be taken early - some that may typically be design decisions will happen during requirements. This is when the component selection begins and must integrate with the wider system development. Functional and non-functional criteria must be considered and there will need to be

[23] http://creativecommons.org/

flexibility, to allow iteration as tradeoffs are considered.

Design

The Design phase adds detail to the requirements and is the main area of iteration in a Component-based process, meeting back with the non-component development of the rest of the system for possible adjustments. To prepare for acquisition, the requirements must be transformed into measurable selection criteria. The characterisation of the desired and the candidate components is key to this process to allow comparison, but currently there is no standard for specifying a component (Christiansson and Christiansson, 2004). In some processes the selection criteria are standardised (Yamamoto and Saeki, 2007), whereas others will have custom requirements set up for each selection (e.g. elicited via GQM for OTSO (Kontio, 1995)).

With a specification of the required component, the developer can search for potential matches. In Mili et al (1992) and Mittermeir et al (2007) this is using a formal specification on an in-house repository. For externally supplied components, repositories such as ComponentSource[24] can be searched. Each repository has its own schema for holding the metadata for listings, most commonly in XML and/or accessible via a web portal (Christiansson and Christiansson, 2004).

Evaluation of components can be static or dynamic, with static evaluation based on specifications and documentation. Dynamic evaluation may require a test harness to provide an environment for the component to be tested. This is testing for evaluation purposes, and is separate to the integration testing done during implementation/integration. Section 2.2 considers component selection in greater depth.

Selection of components may also have to consider more than whether the component does what is required. For critical systems, developers would need to be assured that there is on-going support from the vendor to patch problems with the software. Developers would also need to decide whether the vendor is 'safe' in terms of being in existence to support the product. Information technology is a volatile market, with regular closures, mergers and buyouts of businesses. This may mean that management feels safer to purchase from and depend on 'stable' vendors (e.g. Oracle and Microsoft), than smaller companies who may be more innovative, but also more susceptible to market forces.

[24] http://www.componentsource.com/

Along with the focus on selection and integration of individual components, some research is looking at assemblies of components and prediction of their performance. Stafford and Wallnau (2001) suggests three interlocking questions: What system quality attributes are developers interested in predicting?; What analysis techniques exist to support reasoning about these quality attributes, and what component properties do they require?; How are these component properties specified, measured, and certified?

Compositional reasoning requires information about the components to match the properties that you want to predict. Predictable Assembly from Certifiable Components (PACC) aims to demonstrate compositional reasoning without the need for full formal specification and extending architectural analysis and component certification (Ivers and Moreno, 2008). The work has two premises: system quality attributes are emergent properties that adhere to patterns of interaction among components; and component technology has mechanism for enforcing interaction patterns. Attribute-Based Architectural Styles (ABAS) are used as the basis for defining the interaction patterns within the architecture (Klein and Kazman, 1999).

Burgues et al (2002) proposes a process model for the combined selection of components. For a vertical application, they consider four levels of requirements - external, corporate, interaction and local. These are simplified in to local and global (not local) scenarios to coordinate a cyclic (from PORE) evolution of the selection process (Figure 2.10). The authors are considering UML to provide the required formalism.

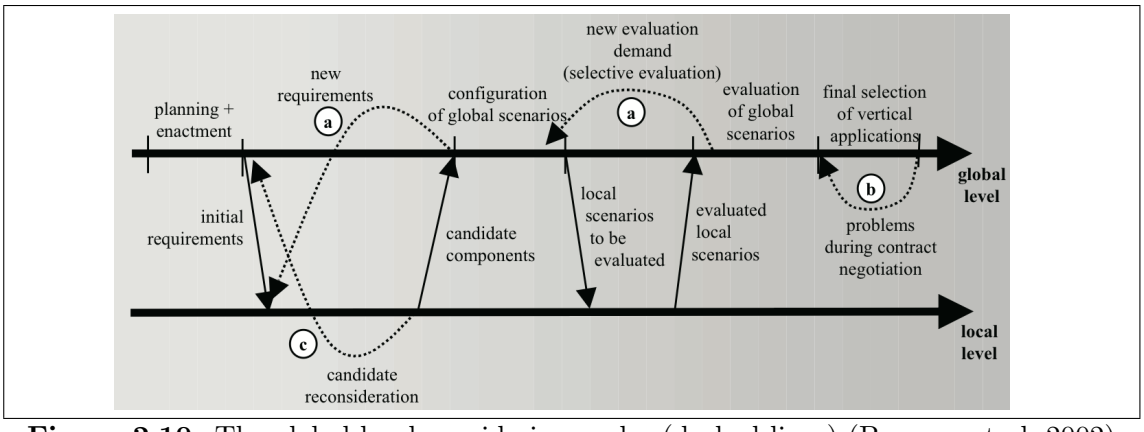


Figure 2.10: The global level considering cycles (dashed lines) (Burgues et al, 2002)

Key points: The shortlisting of candidates takes place during design. Evaluation may be static, or may be dynamic and hands-on. More recent work is considering the selection of assemblies of components.

Construction

Once selection has taken place and the rest of the system has been built in parallel, the components can be integrated into the system. Key to construction is the interoperability of the involved components: the ability of two or more entities to communicate and cooperate despite differences in their implementation language, their execution environment, or their model abstraction (Wegner, 1996). Vallecillo et al (2000) describes three types of interoperability: signature, semantic and protocol. One of the issues they raise is difficulty in checking the implementation against the specification due to the lack of access to source code.

Rose (2000) provides a checklist for adapting and integrating COTS components to ensure that all critical activities and decision are carried out in an effective manner. Architecture has been mentioned as an early decision and the developers need to understand the architecture, all included interfaces and the middleware involved (e.g. CORBA, .NET) (Szyperski, 1998). Architectural mismatch is considered to be caused by different assumptions components make about their environment (Garlan et al, 1995). The paper advised that research was needed to better understand mismatch and the adaptation that may address the problem.

One approach to dealing with this runtime mismatch is to add contracts to the interactions between components (Watkins, 1998). The component itself is then empowered to check that other components are calling it correctly, and the calling component can verify that the results of the call are as expected. Eiffel (programming language) provides the facility for programming by contract in the source code itself (Meyer, 1997), while Watkins implements the contracts in the (Interactive Data Language) IDL for the components (Watkins, 1998). Component-oriented platforms typically use IDLs to specify required and offered functionality (Bracciali et al, 2002). The IDL approach has advantages, especially in that it does not alter the source code, which is unlikely to be available.

Development using CBSE encounters rigidity in the components to be integrated into the target system. In situations where a component is known to differ from its requirements, a compromise can take place using glue code. This may result in changes to the surrounding (calling) system, or modifications to the way the component interacts with the system. A common way of resolving these issues is to use wrappers, bridges or

mediators as intermediaries (Bass et al, 1998). Figure 2.11 illustrates these techniques. This approach saves on resources by not having to customise or rewrite the components. However, it is best to minimise the use of these devices due to the overhead they may add to the system. There is a tradeoff between the cost of customisation, against the cost of developing the interfacing software and its effects on the system performance.

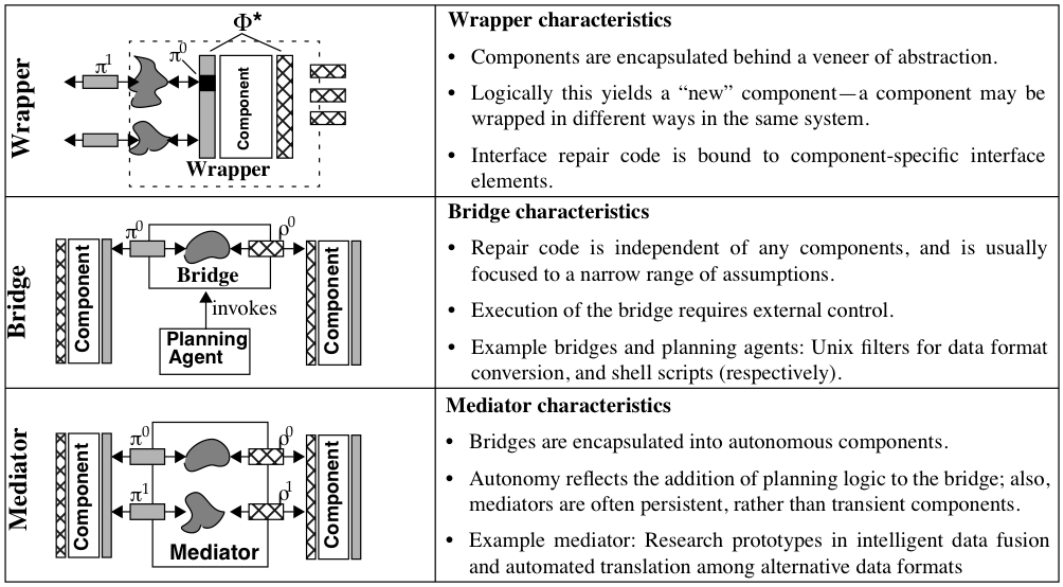


Figure 2.11: Techniques for Repairing Interface Mismatch (Wallnau et al, 1997)

Formalisation of approaches to customising code can improve consistency and maintainability. Xie and Zhang (2007) provides a framework for adaption of software components and discusses two types of adaptation - component signature and component function. Signature mismatch may be in the names or the types of the parameters - this framework formally describes the replacement function to address these mismatches. For functional mismatches, the strategy is to excise excess functionality, and for missing functionality, to write or source additional code and compose the components to provide complete functionality. The composition may be sequential, parallel or alternative (parallel with decision making).

Cubo et al (2007) refer to four levels of mismatch: signature, behavioural (message order), service (non-functional) and semantic (functional), and indicate that most issues occur at the behavioural level. Similarly, Min and Kim (2004) put forward four types of smart adaptors: value range transformer; interface adaptor; functional transformer; and workflow handlers. Brogi et al (2006) considers software adaptation as deploying a

software component (adaptor) capable of acting as a component in the middle between respective components to support successful interoperation. They provide a formalisation of adaptor specifications and show how it can be applied to contract agreements, soft adaptation and hard requirements. Formalisations that have been applied in adaptation include lambda calculus (Bracciali et al, 2002), Calculus of Communicating Systems (CCS) (Cubo et al, 2007) and Finite Automatons (Xie and Zhang, 2007).

Key points: When constructing component-based systems, mismatches between components and the target system are common due to differences in context. Responses to architectural and functional mismatch include glue code, formal adaptation models and contracts.

Testing

Verification and validation of a project should be carried out over all stages. The component developer will test the component during and after implementation - checking to see that it matches its specification and that it executes correctly in at least one context. The client must also test the component - firstly in isolation, then when integrated with the target system.

With components, the testing process is made more complex by the variety of environments that a component may be expected to execute. Weyuker (1998) discusses the problem of testing software components to take their context, or target operating environment, into account. When testing a component for reuse in a new or changed environment, it is important to prioritise testing based on expected usage. Research into predictability of assemblies of components also highlights the increased importance being given to the target context of the component (Crnkovic et al, 2003). Beyond unit, integration and system testing, organisations can carry out feature testing and load testing (including performance, stability, stress and reliability testing). These more specialised tests take the context of the component into account and will vary between applications of the same component.

Third party components pose special difficulties in testing. Developers of components and certification bodies may make use of white box testing techniques as they have access to the component's source code (Morris et al, 2001). However, application developers can only expect an executable and associated documentation to work from. This dictates that

Black-box test techniques	White box test techniques
Equivalence partitioning	Branch/decision
Boundary value analysis	Data flow
State transition	Branch condition (combination)
Cause effect graphing	Modified condition decision
Syntax testing	Linear Code Sequence and Jump
Statement testing	Random
Random	

Table 2.1: Types of test design techniques (BCS, 2001)

component testing (from the user perspective) is limited to black-box techniques (Myers, 1979). An issue for component users is the amount of documentation available. Given a specification of the required functionality and/or interfaces the testing can be based on metadata (Orso et al, 2000), Unified Modelling Language (UML) (Yoon et al, 1999) or Assertion Definition Language (ADL) (Sankar and Hayes, 1994). These can form the basis for specification-based testing.

Test generation options are limited if only provided with interface descriptions. Partition information for input variables can aid in the selection of test data to exercise various scenarios. Behavioural information can provide partitioning information and also assist in developing meaningful sequences of method calls and some oracle functionality.

The British Computer Society provides a component testing standard, using component to refer to the lowest level of independently testable software (BCS, 2001). Table 2.1 lists types of test design techniques and indicates whether they are black- or white-box. BCS metrics for testing follow a general pattern of: `number of successful tests / number of possible tests x 100%`.

COTS components are delivered as black boxes, with no expectation for access to source code. Bertolino (2007) lists the new challenges raised by CBD for testing: components are generic to allow deployment, but must be re-tested in context; and the lack of information provided to allow for testing by a third party. There are three views in component testing and certification: component developers, application developers and third party testing certification organisations (Councill, 1999). When considering the SWEBOK recognised testing areas, the component developer is responsible for unit, basic integration and acceptance testing. Methods used by component developers can include

white-box techniques as they have full access to the source code and design documents to allow structural testing. For application developers and certification organisations, the testing techniques are limited to black-box as it does not require knowledge of the structure of the program (Cechich et al, 2003).

Application developers have two levels of testing to consider - evaluation of candidates can involve testing, and the integration of components needs to be tested, including any glue code or adaptations (Weyuker, 1998). Cortellessa et al (2008) focus on assemblies of components as, even though individual components may be correct, the assemblies need further testing as the combination may be incorrect. Briand et al (2006) list four difficulties raised when carrying out Verification and Validation (V&V) on components: heterogeneity of deployment platforms; limited test model; difficult definition of testing adequacy criteria; and, generality of components.

Testing by application developers and third party assessors is dependent on the documentation shipped with the component. This may include a user manual, user reference manual, interface specification and may include informal, formal or model-based descriptions of the components (Gao et al, 2002). Components may also include mechanisms to support checking, such as that provided through design by contract (Meyer, 1992).

Strategies for black-box testing include equivalence classes, error guessing and random tests (Cechich et al, 2003). Gao et al (2002) adds fault-based testing to this list, while Korel (1999) describes an interface probing approach. Requirements-based approaches work from a model of the system which can be used to generate tests (Tahat, 2001). Metadata is used by Cechich and Polo (2002) and Harrold et al (2001) to provide usage data about a component to assist testing. Built-in-tests is another approach where the component includes interfaces specifically for testing.

In the COMPOSE process, (Kotonya and Hutchinson, 2005), different types of testing are used across development. These are indicated in Figure 2.12: COTS testing in the Design phase, and subsystem assembly, regression testing and non-functional testing at the Compose stage.

A key issue throughout component testing is that the context for development is almost always different to the target context (Weyuker, 1998). This drew attention to the need for testing in the target context. Bertolino (2007) lists composition testing as a key challenge across all of software engineering. An aspect of this is addressed in

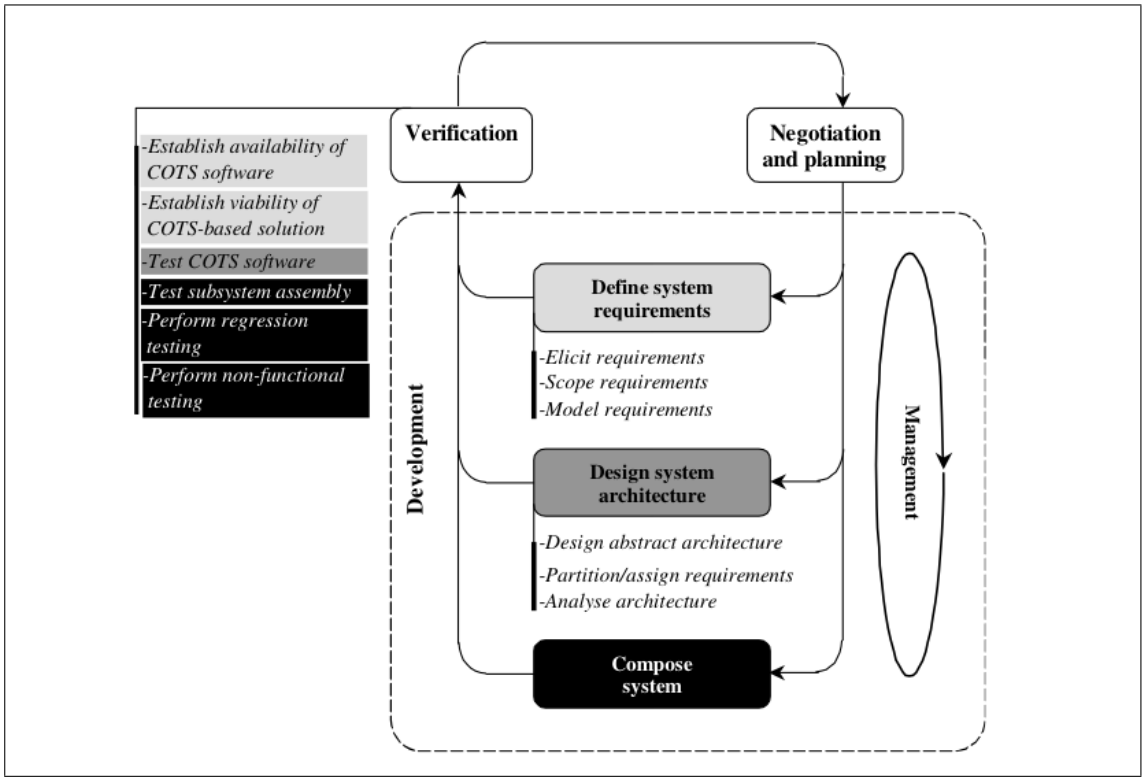


Figure 2.12: COMPOSE process (Kotonya and Hutchinson, 2005)

testing assemblies, however more is required for confirming that a component will behave correctly. Voas (1998a) outlines these issues as expected and unexpected behaviour of software.

Gao et al (2002) discusses component testability in terms of observability, traceability, controllability and understandability. User understanding relies on the provided documentation, in turn affecting testability. Freedman (1991) refers to controllability as how easy it is to control a program on its I/O, operations and behaviours. For components, controllability has three aspects: behaviour control; feature customisation; and installation and deployment (Gao et al, 2002). Observability indicates how easy it is to observe a program and the traceability is how the component allows tracking of the status of component attributes and behaviour. All of these testability measures are impacted in a COTS situation.

Another approach to making the components more usable and trustable is to provide verification and certification of their functioning. This certification can be taken to various levels, giving the component user some assurance that the component will work as expected. Third parties may certify the software (Morris et al, 2001) or the certification

may be of the vendor who developed the software (e.g. CMM and ISO).

Key points: Testing without source is difficult, with added complexity of context and integration. Certification and testability have potential for increasing trust. More detail on testing components is in Section 2.2.5 and Trust is discussed further in 2.2.6.

Maintenance

As software enters the maintenance phase, there needs to be a response to its evolution. In CBSE, the composition of software from different vendors creates dependencies and compounds the traditional maintenance challenge. Maintenance can be classified into four types: corrective (patches); adaptive (porting); perfective (enhancements); and preventative (IEEE, 1990). A study conducted at Statoil found that maintenance effort was 59% perfective, 26% adaptive and 14% preventative (Gupta et al, 2008). It is also recognised that software evolves and is not just being maintained. Lehman's Laws[25] recognised this in 1974 - Law 1: continuing change, and Law 2: increasing complexity (Lehman, 1979-1980).

According to the CeBASE initiative, the top 12 lessons on COTS-based systems maintenance are as listed in Table 2.2. Clark and Clark (2007) list major sources of added costs with COTS: evaluation of new releases; on-going vendor support; ripple effect from upgrades; hardware upgrades; disabling new features; early maintenance; market watch; continuous funding; and increase in costs as a function of the number of third party components. Recognising the risks associated with maintaining COTS-based systems, some options for risk mitigation are to request source code, categorise critical and non-critical components (then focus effort on critical) and to design for change (Clark and Clark, 2007).

Vigder (2006) suggests that system maintenance can be addressed at design time by encapsulating product collaborations, controlling interfaces, controlling dependencies, minimising coupling, using consistent failure handling, having a high level of visibility (of behaviour) and minimising build and deployment effort. Larsson and Crnkovic (2001) use dependency graphs to provide software configuration management for components. The graphs facilitate maintenance by identifying differences between configurations. Nguyen (2008) describes a formal model of the process of software update for CBS and a tool to

[25]These laws noted in 1974, related publication with 5 laws is in 1979, eventually there were 8 laws.

Lesson 1	The refresh and renewal process for COTS-based systems (CBS) needs to be defined a priori and managed so COTS package updates can be synchronized
Lesson 2	COTS software capability and quality evaluation needs to be managed as a continuing task during the maintenance phase.
Lesson 3	The cost to maintain CBS often equals or exceeds that of developing custom software.
Lesson 4	The most significant variables that influence the cost of CBS maintenance include the following (in priority order): Number of COTS packages that need to be synchronized within a release; Technology refresh and renewal cycle times; Maintenance workload for glue code and wrapper updates...
Lesson 5	Maintenance complexity (and costs) will increase exponentially as the number of independent COTS packages integrated into a system increases.
Lesson 6	Significant time and effort must be spent up-front analyzing the impact of version updates and new releases.
Lesson 7	Flexible CBS software licensing practices lead to improved performance, reliability and expandability.
Lesson 8	Wrappers can be effectively used to protect a CBS from unintended negative impacts of version upgrades.
Lesson 9	You may have to re-tailor COTS components with new releases to accommodate new features and functionality.
Lesson 10	The Achilles' heel of most COTS projects is the interface to legacy systems. They fail over and over again.
Lesson 11	Out-sourced CBS applications that don't require refreshed COTS components in their contracts for delivered applications often have to live with obsolete COTS products.
Lesson 12	When the error rates with COTS packages equal to or exceed those being experienced with other software, it is time to consider replacing the package.

Table 2.2: Maintenance lessons for COTS-based systems (Reifer et al, 2004)

keep track of their evolution.

In Kotonya and Hutchinson (2005), the focus in on the impact of change. COMPOSE is architecture-centric CBD process and uses Component Architecture Description Language (CADL) (Kotonya et al, 2001) to model components and architectures. The mapping of requirements to services and constraints, and in turn to components, allows the indirect impact of change to be revealed along with the direct impact of change to connecting components and connectors.

Maintenance and the evolution of CBS are a challenge, requiring ongoing configuration management and management of external dependencies. Selection of components needs to tale the reliability of the vendors into consideration and manage the risks associated with third party software.

Key points: There is a need to consider change and evolution and how they impact on cost and the on-going need for revisiting component selection. Some are developing approaches for estimating and minimising the impact of change.

2.1.5 Issues in CBSE

Issues exist when reusing software in general, with additional issues arising when composing systems from reusable components. Reuse issues stem from: managing development with reuse; approaching analysis and design to maximise reuse of artefacts; trust and discovery. CBSE issues relate to the fact that the developer cannot assume any knowledge of implementation for a third party component, which has an impact on testing. A systematic approach is required to select for the available components. They may also face difficulties integrating the component to fit smoothly with the rest of the system, and responding as the system evolves.

From the literature discussed in the previous sections, a list of key issues in CBSE can be compiled:

1. **Availability of and sourcing components**
2. Extra time required for development
3. **Selecting the 'best' components**
4. Trusting components
5. **Testing a component**
6. Integration problems
7. Vendor reliability & risk management
8. Configuration management & interdependencies.

In this study, the research explores those issues relating to item 3: selecting the best components. Of secondary interest are sourcing and testing, items 1 and 5. In most published processes, filtering for candidates is manual using sources such as: 'market surveys, Internet search, product publications and sales promotions, and computer fairs and shows', (Kontio, 1995, Kunda and Brooks, 2000). Over the decade since research began in the area of component selection, the current practice is to undertake an Internet search or ask an expert (Li et al, 2005) and evidence indicates that the use of published processes is low, with only 19% of COTS projects using a formal decision-making method (Li et al, 2005). While selection of components has been shown to have the most impact on risk in CBD (Port and Chen, 2004), developers are still approaching it in ad-hoc ways.

In addition, the issues relating to component selection are considered, including availability and sourcing, and testing. The next Section works from these issues to identify

opportunities and positions.

2.2 Component Selection

Although components were promoted as simplifying and speeding up software development, there are also intrinsic characteristics of CBSE which make component selection difficult. Among them is the lack of a well-defined process and the lack of a specification standard. Assessment for selection needs to include functional and non-functional criteria, which rely on the specification standard, along with more hand-ons evaluation. The black-box nature of the software restricts evaluation and testing to what can be inferred from the published interfaces.

Some processes concentrate on the issue of how to assess ensembles of components and how the group will interact as the complexity of systems increases. Another issue is the selection is usually a time-consuming, manual process which restricts the number of components evaluated (scalability) and may result in suitable software being missed. Questions have also been raised about the suitability of techniques used for comparison (Ncube and Dean, 2002). The manual selection process does not lend itself to repetition as the system evolves and marketplaces change. It may be that the choice of components impacts on other aspects of the system under development. This can result in Cyclic-Requirement-Component Dependency (CRCD) where selection does not converge (Tran and Lin, 1999).

When considering improvements to component selection, there are a range of potential focus areas[26]. The characterisation and specification is needed for requirements definition and matching, and for the discovery of candidates. In the pursuit of repeatability and quality, and potential for automation, a defined process is required. Selection of the best component is a combination of being able to discover potential candidates and the evaluation techniques used within the selection process. Evaluation may be static, based on specifications, or dynamic, via testing. The dynamic approach gives more information on the suitability, particularly as context and mismatches may negatively impact on expected performance in the target environment. In any reuse, there can be issues of trust in the quality of the code built by others. By looking at supporting quality in

[26]In terms of the Topic Map (Figure 1.2), the discussion is within "CBSE Issues", and draws on the wider theory of software engineering, information management and artificial intelligence

CBSE, it may be possible to increase trust and uptake. Finally, the use of automation to support selection is an additional way to encourage uptake. This is considered in Section 2.2.7, spanning AI, knowledge based approaches and tool support.

The focus in the next Section is the various aspects of component selection.

2.2.1 Characterisation and Specification

Descriptions of components are required to facilitate discovery and allow a shortlist of possible matches to be created. This information can contribute to the evaluation of the candidates; however further information is likely to be needed. In similar applications, the description of resources are standardised through schemas to hold metadata. There is a long history of cataloguing information through standardised systems such as MARC[27] and Dublin Core[28]. However, there is no standard for the documentation of a component. This creates difficulties when trying to compare components based on the vendor's 'shipping information'. Vendor and repository information is based on an internally developed data model - complicating comparisons across repositories if the models do not match.

A common discovery mechanism is the use of search engines on free text, such as Google[29]. Cechich et al (2006) indicate that current shortlisting is based on unstructured information on the web, which does not align well with complex selection criteria. If given a list of components, this may mean the ones higher on a list are chosen. It is not a trivial task to select the best component as there will always be differences between the resultant components based on the programmer/designer's view of the problem space. Repositories such as Component Planet[30] provides a search mechanism which sorts the components in order of previous sales, and includes a customer rating system to assist in selection of components. However, these orderings are based on the requirements of other organisations and may not be useful or valid in the target context.

Early work in software reuse applied faceted classification to the characterisation of software, to aid in-house retrieval (Prieto-Diaz, 1991). Although work continues on this

[27] MAchine-Readable Cataloging - http://www.loc.gov/marc
[28] http://dublincore.org/
[29] http://google.com/
[30] Site has since closed - http://www.componentplanet.com/

approach (Kaur and Goel, 2011), Cechich et al (2006) survey component characterisation and asserts that there is a move from faceted classification and taxonomies, towards ontologies. They propose the need for standardisation of component identification, notation and frameworks. Since the early work of Frakes and Pole (1994), Sassi et al (2003) include a survey of eight models, noting missing elements in each, including information on vendor, functionality, non-functional attributes, cost, domain, architectural features. Across all characterisations there was no information on dates for first and most recent release, and change frequency.

Key points: From this discussion, gaps remain in the characterisation of components, particularly in the lack of a standardised specification. To allow discovery to make more complex, structured queries on knowledge bases, ontologies are being applied. On a basic level, however, the data model - basic recorded information - is yet to be standardised.

2.2.2 Selection Processes

A defined process for selection is required for repeatability and is particularly important if building strategies to improve selection. Mohamed et al (2007a) provides a diagram (see Figure 2.13) of the evolution of selection practice, many of which will be described in this Section.

One of the earliest formalisations of software selection was Off-The-Shelf-Option (OTSO) (Kontio, 1995). This introduced the use of hierarchies of selection criteria elicited using Goal/Question/Metric (GQM) (Basili et al, 1994) and evaluated using the Analytic Hierarchy Process (AHP) (Saaty, 1990), which was adopted in many subsequent processes including STACE (Kunda and Brooks, 1999) and CAP (COTS Acquisition Process) (Ochs et al, 2001). The AHP was developed by Saaty and is based on hierarchical decomposition, pairwise comparison and priority vector generation and synthesis (Mohamadali and Garibaldi, 2009). In the Procurement-Oriented Requirements Engineering (PORE) approach (Ncube and Maiden, 1999), an alternative evaluation method, outranking, was described, with the authors later publishing a paper outlining the limitations of the commonly used approaches, including AHP and Weighted Score Method (WSM) (Ncube and Dean, 2002).

The PORE approach provides an overview in Figure 2.14 for a process developed

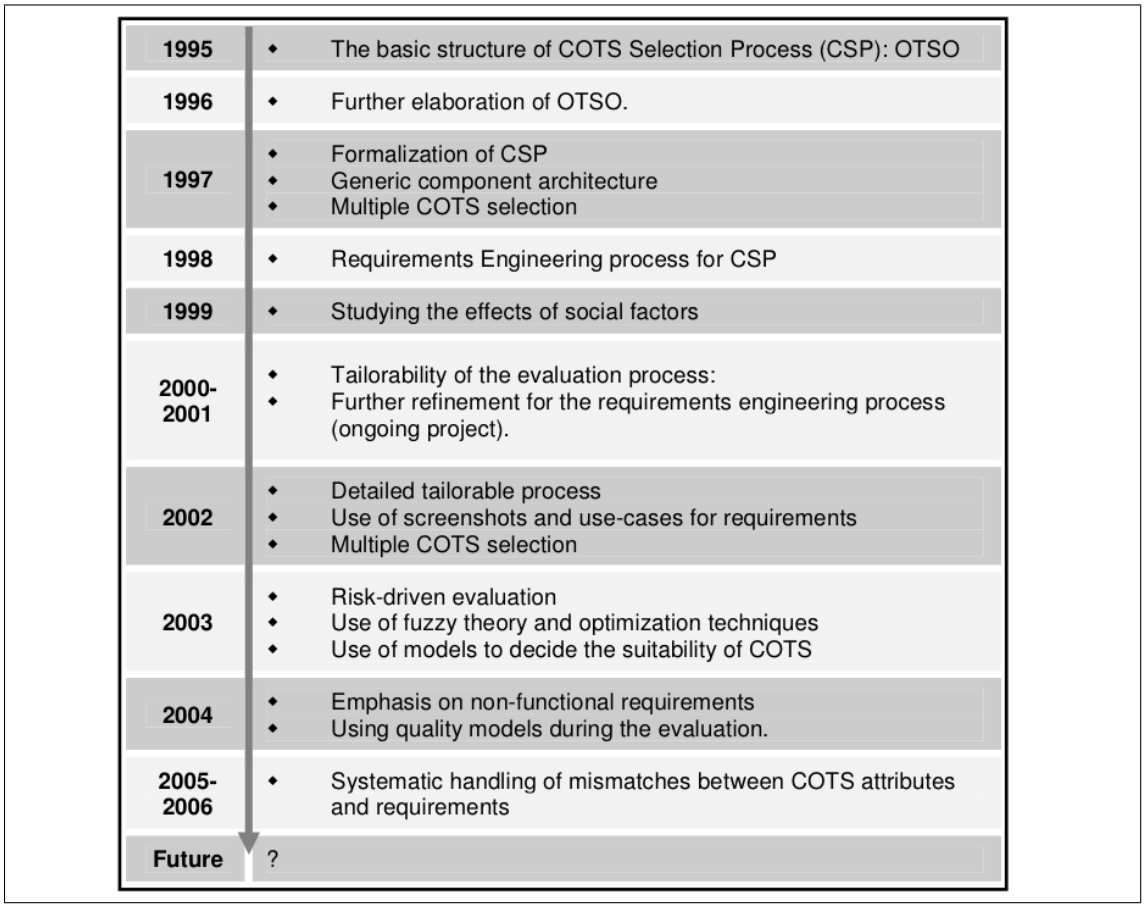


Figure 2.13: Evolution of COTS selection practices (Mohamed et al, 2007a)

to respond to lessons learned in a UK Ministry of Defence project (Ncube and Maiden, 1999). Key attributes are: knowledge engineering techniques; feature analysis; MCDM techniques; and design rationale techniques to record and aid the decision process. In PORE, selection is an iterative process of rejection, reducing the list of candidates as seen in Figure 2.9. COTS-Based Requirements Engineering (CRE) (Alves and Castro, 2001) focusses on systematic, repeatable requirements-driven COTS software selection process. The method provides templates and guidelines for iterative rejection of candidates (as in PORE). The CRE provides criteria targeted at non-functional requirements and situation rules deal with conflicting requirements. Scores are generated by WSM, with AHP suggested for more complex tasks.

The Evolutionary Process for Integrating COTS-Based Systems (EPIC) uses a risk-based spiral development process as four spheres of influence are adjusted based on available components (Albert and Brownsword, 2002). EPIC has four phases: inception, elaboration, construction and translation. There are three process components in the

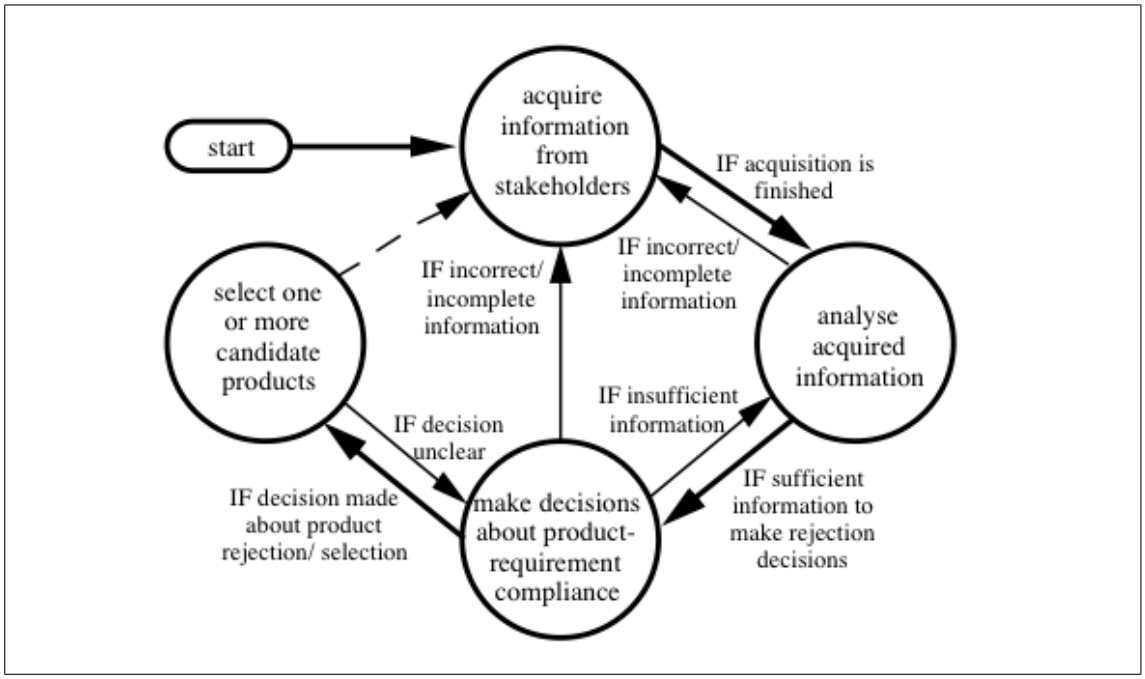


Figure 2.14: PORE route map (Ncube and Maiden, 1999)

COTS Acquisition Process (CAP) (Ochs et al, 2009): CAP Initialisation Component (CAP-IC), CAP Execution Component (CAP-IC) and CAP Reuse Component (CAP-IC) (Figure 2.15). This method uses over sixty metrics based on the ISO 9126 standard for product quality, and is supported by expert interviews, literature reviews and applied research activities. The developers of PECA list five evaluation mistakes affecting quality and traceability: 'inadequate level of effort; neglecting to re-evaluate new versions or releases; use of 'best of breed' lists that do not reflect the characteristics of the system; limited stakeholder involvement; and, no hands-on experimentation' (Comella-Dorda et al, 2002; p. 86). The four basic elements are: Planning the evaluation; Establishing the criteria; Collecting the data; and, Analyzing the data (see Figure 2.4) which is a means for evaluation and is not always executed in sequence. The process is open to respond to new criteria, unexpected discoveries or the need for better data. PECA has three main outputs: product dossier, evaluation record and summary/recommendation.

Kotonya and Hutchinson (2005) have developed a method with a cyclic process (Figure 2.12) which includes verification and negotiation in each cycle, allowing iteration until a match is found. The shortest use of the Kotonya process would unroll to three iterations: requirements, design and composition, but it is possible to have multiple visits to each of the development tasks and revert to the previous task if needed.

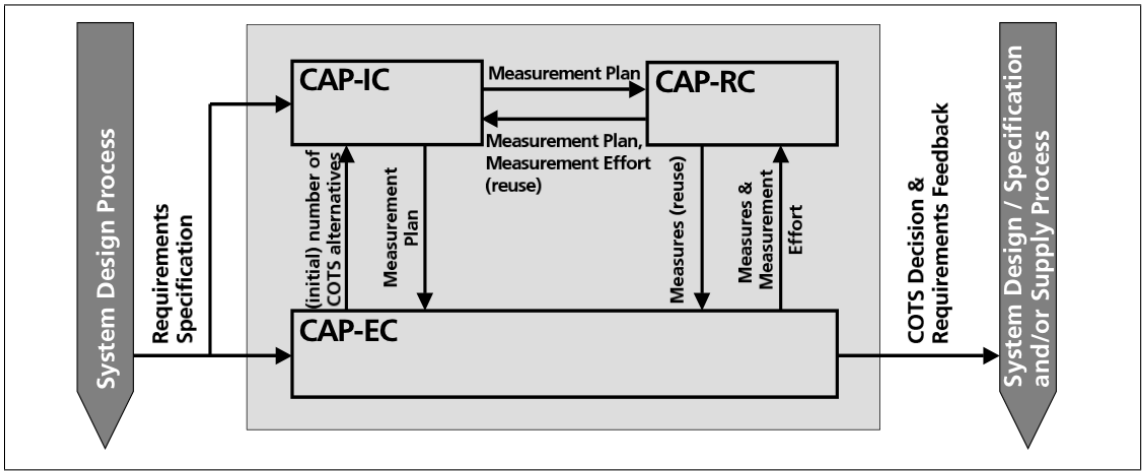


Figure 2.15: CAP components, internal and external information-flow (Ochs et al, 2009)

Once there is a range of components to match a given requirement, it then becomes an issue to select the 'best set' of components for the task at hand. This may be optimised for efficiency, security, ease of integration etc. Early work in this area included K-BACEE - a system for selecting the best ensemble of components based on a weighting of their performance against certain criteria (Seacord et al, 2001).

In Sassi et al (2004), the authors give an abstracted view of a number of CBSE processes representing the tasks common to many processes to facilitate comparison. In the diagram (Figure 2.16) the sources for discovering COTS are listed as the input to 'Identify potential COTS', which iterates to refine the search criteria. From there, arcs representing selection methods move to the 'Select on COTS' action. From this state, the component can be integrated by architectural framework or direct composition, or the task may be abandoned. There may also be a need to return to select a different component. Once integrated, checks can be conducted for errors, side effects or interface propagation analysis. With this lens, it becomes clear that there is a common basic process across the literature and that the differences are in the choices of arc to transition between states.

Martinez (2008) carried out an analysis of COTS selection approaches, forming a summary table (Figure 2.17). The comparison factors include whether the methodologies incorporate various dimensions: searching (SEARCH), characterisation (IDENT), evaluation (EVAL), selecting single or multiple components (SNG/MLT), provide for reuse (REUSE), can be tailored (TAILOR), have tool support (TS) and the roles that

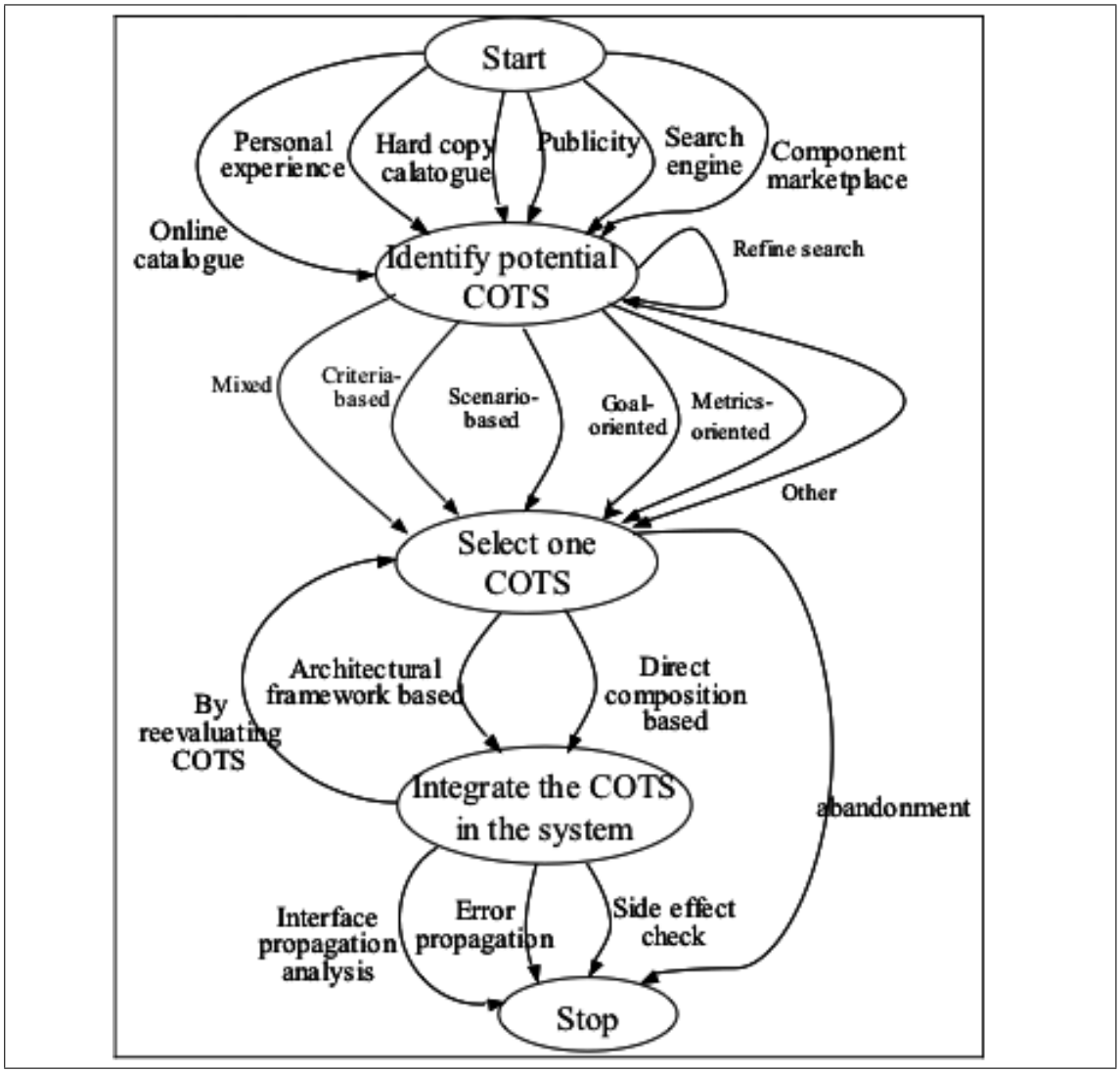


Figure 2.16: CBD process modeled with the MAP (Sassi et al, 2004)

are considered (CVR: MW, QE, S, KK[31]). The assessment is that there is still a need for: approaches with full coverage of dimensions; mechanisms for documenting COTS; mechanisms for search and identification; and a mechanism for knowledge management and reuse.

Key points: While there are a wide range of selection processes published since 1995, few have automation or support for reuse or for searching for components. Many processes use the AHP or similar approaches. Most are complex to understand, and it is difficult to see how they integrate with the wider development process, and the organisation.

[31]MW = market watcher, QE = quality engineer, S = selector, KK = knowledge keeper

APPROACH		COMPARISON FACTORS											
Name	Year	SEARCH	IDENT	EVAL	SNG	MLT	REUSE	TAILOR	TS	CVR			
										MW	QE	S	KK
OTSO	95/96	-	-	√	√	-	-	-	-	-	*	√	-
IusWare	1997	-	-	√	√	-	-	-	-	-	√	*	-
PORE	1998	-	-	*	√	-	-	*	√	-	√	*	-
STACE	1999	-	*	√	√	-	-			-	√	√	-
CAP	2001	-	-	√	√	-	-	√	-	-	√	√	-
CRE	2001	-	-	√	√	*	-	-	-	-	√	√	-
CEP	2002	-	*	√	√	-	*	-	-	-	√	√	*
CARE	2001	-	*	*	√	-	*	-	√	-	√	√	*
PECA	2002	-	*	√	√	-	-	√	-	*	√	√	-
StoryBoard	2002	-	-	*	*	√	-	-	-	-	*	*	-
Combined Selection	2002	-	*	-	*	√	-	-	-	-	*	*	-
SCARLET	2002	-	-	*	√	√	-	*	√	-	√	*	-
DBCS	2003	-	-	*	√	-	*	*	-	-	*	√	-
WinWin	2003	-	-	√	√	*	-	-	-	-	*	√	-
COSTUME	2004	-	-	*	√	√	√	√	√	-	√	√	√
MiHOS	2007	-	-	√	√	-	-	-	√	-	√	√	-

Figure 2.17: Comparing some representative methodologies dealing with COTS selection (Martinez, 2008)

2.2.3 Metrics for Evaluation

Evaluation is driven by the requirements, with some processes including requirements elicitation (e.g. PORE). It is closely linked to the characterisation of the components, which comes from the component developers and brokers, whereas the requirements are defined by the application developer. Requirements can be divided into functional and non-functional, referring to the behaviour and implementation in functional requirements and attributes including trust, reliability, vendor history and security on the non-functional side. The metrics used in selection processes may be unique to each selection task, or there may be a template to use as part of the process. A common approach to moving from requirements to metrics is to use the GQM approach (Basili et al, 1994). This gives a customised hierarchy of criteria and metrics for each selection task.

Reusability metrics for black box components has similarities to those from object-orientation, however they require a considerably different approach (Washizaki, 2003). They define five metrics targeting enterprise Java Beans (EJB) which are combined into and overall metric. (Gui and Scott, 2006) considers static measures of coupling targeted at a Java component search engine, formalised in OCL (Object Constraint Language). (Gill, 2004) presents an interface complexity metric based on interface signature, interface constraints, interface packaging and configurations. Serban and Vesca (2007) provide

metrics for CBSD where the component assembly is represented as a directed graph of dependencies. The CBC metric is the number of components with which a component is coupled. From this, chains for depth dependence and breadth dependence can be developed. Unfortunately, many of these metrics require source code for their calculation. Thus, unless the code is provided by the developer or vendor, they cannot be assumed for third party components.

Gill and Grover (2004) state that many traditional software metrics are inappropriate for CBS as the focus should be on granularity and interoperability aspects. They put forward granularity, adaptability, interoperability and interface complexity as more useful metrics. Some metrics are limited in applicability due to the black box nature of components and the variation in the structural complexity that is dependent on the target context. Suggested metrics are: interface complexity, size, portability, integration complexity, test coverage, semantic complexity, reliability, functionality, customer satisfaction, resource utilisation, cost, time-to-market and incremental delivery.

Sedigh-Ali et al (2001) provide some SE metrics for COTS-based systems, aiming to quantify component quality. In addition to the traditional software metrics (Cost, Quality, Reusability and Risk) COTS require integration complexity and performance to be considered. Third party software creates new risks due to unpredictable quality and this may affect performance, reliability, adaptability and ROI (Return on Investment) (Sedigh-Ali et al, 2001). System-level metrics are in three categories: management; requirements; and quality, while costs of resources to improve quality include appraisal, prevention, internal failure and external failure costs.

ISO/IEC 9126 provides a framework for the evaluation of software quality, defining six quality characteristics: functionality; reliability; usability; efficiency; maintainability; and portability (InternationalStandardOrganization, June, 2001). The standard clarifies quality terms to improve communication and reduce misunderstandings between producer and supplier. Quality models for components have been put forward by Alvaro et al (2005b) and by Andreou and Tziakouris (2007), both based on a modified ISO/IEC 9126.

The standard is also the basis for the Carvallo et al (2004) quality framework to support COTS evaluation. They reuse existing quality models, have patterns for reuse and tool support as part of the COSTUME method. In their later work (Carvallo and

Franch, 2006) they propose extensions to the ISO/IEC 9126-1 quality model with non-technical factors for COTS selection including supplier, cost and product categories, subdivided into over 200 non-technical quality attributes (e.g. Product > Stability > Time in Market). The metrics are arranged into a hierarchical tree-like structure which can be used for identification of mismatches, potential risks, estimating budget and schedule and analysing the viability of the project. Sharma et al (2008) surveyed quality models for component and non-component systems and put forward a hierarchical approach (demonstrated using AHP). In an industry application, Choi et al (2008) show the use of a quality model for embedded software at Samsung.

Another focus is taken by Ding and Napier (2006) who provide a measurement framework for risk in CBSD considering two stakeholders - the vendors and the customers. They provide risk measurement tables for the vendor including marketing, management and development categories, while the customer has application use, application management and competitive advantage risks.

McGregor et al (2003) put forward the Component Reliability measurement method (CoRe) for empirical reliability measurements. The definition of CoRe includes probability (% of correct executions), operational profile/context and duration. Each component in the system plays a 'role' in the overall system tasks, which makes up the operational profile. They create and execute a test suite for each role, then analyse it. CoRe is a part of PECT.

Serban and Vesca (2007) models the component based system as a dependency graph, then applies measures for coupling, depth dependence and breadth dependence.

Key points: There is no standard set of metrics related to evaluating components. Some existing (pre-CBSE) metrics are not applicable, while more focus on vendor, reliability, granularity, adaptability and performance are advised. In component selection, some of these values would be supplied by the repository metadata, and some would be generated through the results of the evaluation. Some gaps in vitality measures have been shown.

2.2.4 Evaluation Methods

Given a set of attributes, the requirements and the values presented by a particular component, the screening or shortlisting process needs to combine the data to create

a ranking or recommendation. Much of the literature uses the Weighted Sum Method (WSM) to aggregate a value by summing attribute weights multiplied by their respective values (Solberg and Dahl, 2001, Alves and Castro, 2001). Criticism of the WSM includes the summing of differing types of data (e.g. cost plus memory plus quality), lack of process for determining attribute weights and the inherent problem with the formula losing dependency information between attributes (e.g. conflicts and co-requisites).

A commonly used alternative is the AHP, which includes a method for determining weights and component scores against attributes (Kunda, 2003, Kontio, 1995, Ncube and Maiden, 1999). These scores are based on pairwise comparisons, and thus use the same 'units', even when combining qualitative and quantitative data. Features of the AHP are that it organises the criteria into a hierarchy (e.g. group quality attributes as subnodes of the quality node) and that scores can be consistency checked. Disadvantages are the number of pairwise comparisons (and therefore time) required and that the interplay between the attributes is lost as the final aggregation is essentially the WSM formula. A technical criticism of the AHP is the rank-reversal problem, which can be addressed by using a multiplicative formula for aggregation (Triantaphyllou, 2001).

The common problems with WSM and AHP stem from the assumption that attributes are independent, resulting in compensations in scores and 'passing' unworkable combinations of values (e.g. .NET with Linux). Another option for COTS and component selection is the outranking approach using the ELECTRE family of methods (Roy, 1991). These methods rank each candidate on each attribute and determine an outranking relationship to categorise attributes into those preferred and those non-preferred. As with the AHP, comparisons are made between candidates on each attribute, removing the issue of units and attribute types. Although it has been successfully used for software evaluation (Anderson, 1989, Morisio and Tsoukiàs, 1997), there are issues with explaining the reasoning for decisions and that a complete ranking may not be possible (Kunda, 2003).

Key points: A range of approaches have been considered in evaluation, many of which are based on WSM and AHP. Other ranking mechanisms have been used, while there are also options in computational intelligence to be explored.

2.2.5 Testing

In Section 2.1.4, the approaches to component testing, and some of the challenges were discussed. In this section the focus is on test generation for the evaluation of the component. Regression testing is also discussed as it is relevant for evolution, and perhaps when testing alternatives.

There are many approaches to test generation for components, depending on the information available to the tester. Bertolino (2007) lists three approaches: model-based, random and search-based, while Gao et al (2002) puts forward usage, error and fault-based testing. Cechich and Polo (2002) identifies equivalence-based, error guessing and random testing as appropriate, Briand et al (2006) adds category partitioning, CSPE (Constraints on Succeeding and Preceding Events) constraint-based, testing logical expressions, statechart and metadata. Within testing logical expressions, approaches include predicate coverage, combinatorial coverage (exhaustive), implicant coverage and Prime Implicant Coverage (PIC). Briand notes that there can be issues with the level of detail in statecharts and scaling techniques when they become large. In some cases, a model or specification of the component can add to the testing options.

From a testing perspective, components can be characterised by their interfaces, operations, events (external), context dependency relationships and content dependency relationships (Wu et al, 2003). The relationships may be derived from UML diagrams with context coming from collaboration, sequence and statechart diagrams and content from collaboration and statechart diagrams. Cechich and Polo (2002) applies reverse engineering of code and interface probing as two methods of enhancing black box understanding.

Aspects may be used to correlate to the services that sub-groups of components may require from each other, for example user interface aspect (Grundy, 1999). In Cechich and Polo (2002) test selection is based on mapping methods to one or more aspects to provide a systematic approach for coverage of methods. This allows aspect specific information, such as pre/post conditions to be considered across all methods in an aspect. They note that once aspects are identified, traditional techniques for black-box testing can be applied. Their example uses category-partition and metadata to produce test frames. The approach can also be applied to integration testing.

Briand et al (2006) focus on the contractual specification of component interfaces.

The approach is to combine constraint types and predicate criteria into fourteen CSPE-based test adequacy criteria. The user view is supported via required methods and required constraints. They automate the generation of tests by representing constraints in a graph, with the costs associated with each arc, then traverse the graph (using DRPP) and compare cost based on size of the test suite in terms of number of methods executed.

Regression testing is 'the selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements.' (IEEE 610.12). Zheng et al (2006) utilise impact analysis for regression testing. The premise is to safely reduce the amount of testing required when components change. I-BACCI takes old/new binaries and source and tests for an application and generates the impact analysis. Firewall analysis (White and Leung, 1992) helps limit the regression testing by only testing within the firewall and running integration tests across the firewall. Their approach identifies changes, then propagates them along the call graphs to identify affected glue code. Test cases are then chosen for glue code. In Gao et al (2006), a systematic re-test method for components is proposed. Four step process given for assessing change impact, using a firewall approach. The approach provides automated black box selection for reuse and test suite refreshment, based on extensive metadata to support models. Orso et al (2001) have techniques to use metacontents for regression test selection in two ways - code-based or specification-based. Control flow graphs are used with the new and old being compared via synchronous traversal.

Bergner et al (1999) uses formal models for specification-based test generation for component systems. Instances in the model include components, interfaces, connections, relations and properties. The work is based on UML diagrams including state transition diagrams, instance diagrams, sequence diagrams. Another useful activity is to support test stub generation to represent the clients and servers for the component under test (Rocha and Martins, 2008). The driver coordinates test executive in terms of inputs, outputs and results. The researchers provide a static, model-based approach to stub generation. They propose the use of activity diagrams which are: turned into a graph; paths are selected; test cases specified; inputs identified; and test cases are implemented.

Hamlet (2007) discusses the potential value of a theory of composition of components based on descriptions in a catalogue, then static assessments can be done. This work

sets up foundation for testing theory and notes the difference between what a component should do and what it does do.

Key points: Testing techniques for third party components are limited to black-box techniques, often with little more than interface information. Some documentation may be provided with each component and may give more options (e.g. specification-based testing). Techniques for isolating which part of the system to test (e.g. firewall) may also inform testing for component evaluation.

2.2.6 Quality

An underlying barrier to using third party components is a question of quality and trust. Ratings are a mechanism for giving confidence. These may be informal from individuals, or be the result of a more formal process. One path to a rating is through quality-related metrics, such as those by Alvaro et al (2005b), Andreou and Tziakouris (2007), Sharma et al (2008) and McGregor et al (2003). Other approaches are through certification, contracts, self-testing components and availability of source code, to give a level of guarantee of quality. Most of these rely on dynamic testing of components by the developer, user or a third party organisation.

Software certification has many flavours. For some, it should be done by certifying the organisation and its processes, implying a high quality product results from high quality processes. This is advocated by users of the Capability Maturity Model (CMM) developed by the SEI and in a more general sense, the ISO 9000 standard. Others see this as a flawed argument and advocate that the product should still undergo full testing before it can be certified. Voas (2000) puts forward a model for certification to take place remotely, after deployment.

In 2000, SEI listed 'lack of certified components' as the third of four inhibitors to adoption of component technology (Bass et al, 2000). Alvaro et al (2005b) cites literature that certification as a precondition for CBSE to be adopted in the large. Certification can be carried out by: developers (first party), users (second party) or an independent testing lab (third party), according to Councill (1999). Voas (1998b) describes a certification triangle for the three aspects that can be assessed - product, process or personnel.

Voas (2000) advocates separation of certification from software development. He puts forward three techniques that focus on rare conditions from the system perspective;

desirable behaviour testing; abnormal testing; and fault injection.

A historical perspective on certification splits the field into two ages, as shown in Figure 2.18. Prior to 2001, the focus of certification was on mathematical and and test based models, whereas later work looks at predicting quality requirements (Alvaro et al, 2005b). Wohlin and Runeson (1994) provide a usage model and usage profile to carry out usage tests and provide a certification of reliability with a degree of confidence. The Trusted Components Initiative (TCI) developed an approach using pre and post conditions on APIs (Meyer, 2003). Voas (1998a) defined a certification framework using black-box testing, system level fault-injection and operational system testing - to assess the component and its behaviour in context.

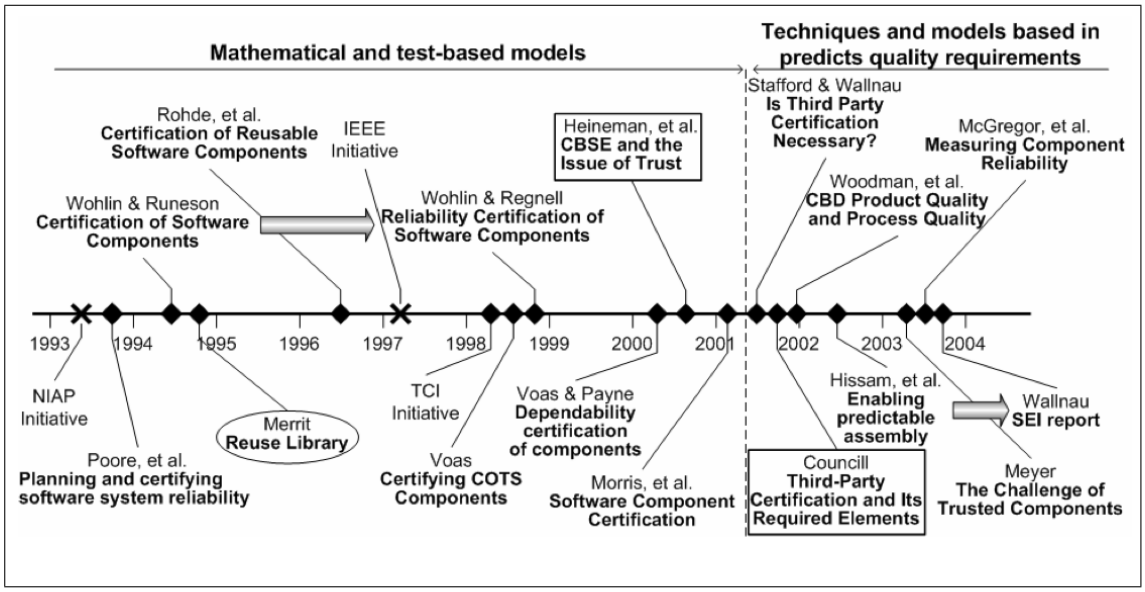


Figure 2.18: Research on software component certification timeline (Alvaro et al, 2005b)

Entering the 'second age' of certification in 2000, the focus changed to include new properties (Alvaro et al, 2005a). Woodman et al (2001) provided requirements applicable to certification: accuracy, clarity, replaceability, interoperability, performance and reliability. Two main directions for certification are formalism and component quality models - which Meyer (2003) refers to as the high road and the low road.

Aimed at mass-market software, the independent certification in this model is carried out by Software Certification Laboratories (SCLs) which receive testing data from actual users, providing access to higher numbers of test results using real world data. An alternative, aimed at the smaller end of the market, is to ship test certificates with the component (Morris et al, 2001). The application developer can then run and verify the

results of the included tests against the component to make their own assessment. A draw-back of certification is that it ignores the target context, testing the component in isolation. Although, this provides confidence that a component meets its specification, application developers will still need to thoroughly test the component in their specific environment.

Bader et al (2003) attempt to identify aspects and features that can be classified as trust attributes (list includes vendor reputation and customer loyalty). They note that trust in components has a relationship to testability and is inversely related to risk. Also noted is the difference between vendor and consumer views on trust and the importance of non-functional attributes in building trust. Attributes identified include:

- Functional specification - perhaps extend design by contract
- Non-functional specification: NFR and QoS, dependency requirements, reputation, support
- Degree of associated risk
- Time and usage history.

Bader et al (2003) suggests the publication of trustable components using wrappers via brokers. This could then use a contract generator to build a usage contract for the wrapped components. Wu et al (2003) are interested in performance-sensitive properties to be stored in repositories for use in predictive models for a product. The use layered queuing networks for behaviour prediction.

Taleghani (2007) is working on the use of software model checking to verify the properties of software. This allows the state-based search strategies for software verification to be used as a reduced set for certification. Developers can supply a model, rather than code for the certification process.

A survey of the state of the art in component quality evaluation has described it as an immature area (Alvaro et al, 2005b). They later put forward a Certification Quality Model (Alvaro et al, 2010) including four levels: characteristics > sub-characteristics > attributes > metrics. Sub-characteristics are grouped into life-cycle and runtime with additional information including technical and organisational information. The metrics framework used is an implementation of GQM.

One of the issues when rating components is the overhead of compiling the ratings

and carrying out assessment. Recommender systems make use of collaborative filtering to select items that may be of interest to a user. These are widely used in online repositories, with Amazon being a leading example. Adding value to this, users of the system can rate entries, which are aggregated and viewed by others. Massa and Avesani (2009) puts forward a system where trust can be established through social interactions, then used to select collaborations and services. This crowd-sourcing is widely used and accepted for software selection in Appstore.

Key points: A range of approaches to quality in CBSE have been discussed, which should, in turn, improve trust in the reuse of third party components. For some, quality processes are the path to quality products; others have developed quality models to provide metrics for assessment. A more direct and formal approach is the certification of software products, requiring certification bodies, metrics/models and the request for certification. In an era of crowd-sourcing, some repositories collect user ratings which can be used as a quality indicator. Others build in self-testing and contracts to give developers confidence.

2.2.7 Automation, Intelligence and Tool Support

Automated software engineering is inherently knowledge-centred (Pedrycz et al, 2011), with two main camps in qualitative and quantitative approaches. A survey of mining software engineering data discusses the transition from using qualitative data for verification, to applying quantitative techniques to discover hidden information in data repositories (Mendonca and Sunderhaft, 1999). Briand (2002) discusses how knowledge engineering can support software engineering in the areas of: planning and monitoring; quality and process; decision support and automation. Briand's definition of knowledge engineering encompasses the use of artificial intelligence, computational intelligence, knowledge bases, data mining and machine learning. Mining techniques include classification trees, association discovery techniques, clustering techniques, artificial neural networks, optimised set reduction, Bayesian belief networks, visualisation and visual data mining (Mendonca and Sunderhaft, 1999).

Computational intelligence is a coherent and symbiotic collection of information technology including fuzzy sets (granular computing), neural networks and evolutionary computing (Pedrycz et al, 2011). The authors provide examples of computational intelligence

applications including fuzzy models of software processes; neural networks in data visualisation, and logical models of software quality. An emerging area is search-based software engineering, which reformulates software engineering as a search problem through representation of the problem, fitness functions for preferred solutions and a set of manipulation options The data for search-based software engineering is made available through recording metrics across processes (Harman et al, 2009).

The combination of the limitations of current methods for evaluation, improving repository and discovery options and greater uptake of CBD, has led a drive for automated tool support to aid software selection. Ruhe (2002) compared ten selection processes published up to 2003 and stated that none of them were ready to be included in a decision support system. Key to automation is the characterisation of components, pioneered by Prieto-Diaz (1991) and now more market-driven by metadata utilised by repositories. This characterisation often includes substantial ontologies such as the Trove in freshmeat (freshmeat, 2007). In many cases, software selection relies on a search engine, some search terms and the user's patience to trawl through results. This manual process entrusts the search engine algorithm to order the results and then the users to make a comparison based on their criteria, evaluation and intuition. However, for those requiring quality in their software selection it is important to have objectivity, repeatability and transparency in the selection process.

The major visual tools to support CSBE are composition environments and decision support tools. SCARLET (Maiden et al, 2003) provides an integrated process support for the selection process. The tool is integral to the SCARLET process to achieve four decision making goals: acquire information; analyse information; use information for decision; and reject non-compliant candidates. Voinea and Telea (2005) uses information from a CVS to provide a visual interpretation of the evolution of a component-based system.

Examples of visualisation related to CBSE is the baseline characteristics and values in Figure 2.19, and the CLARiFi graph, which helps the user understand performance against selection criteria (Figure 2.20). COMPOSE (Kotonya and Hutchinson, 2005) provides a visualisation to assist impact assessment for a particular change to a component (Figure 2.21).

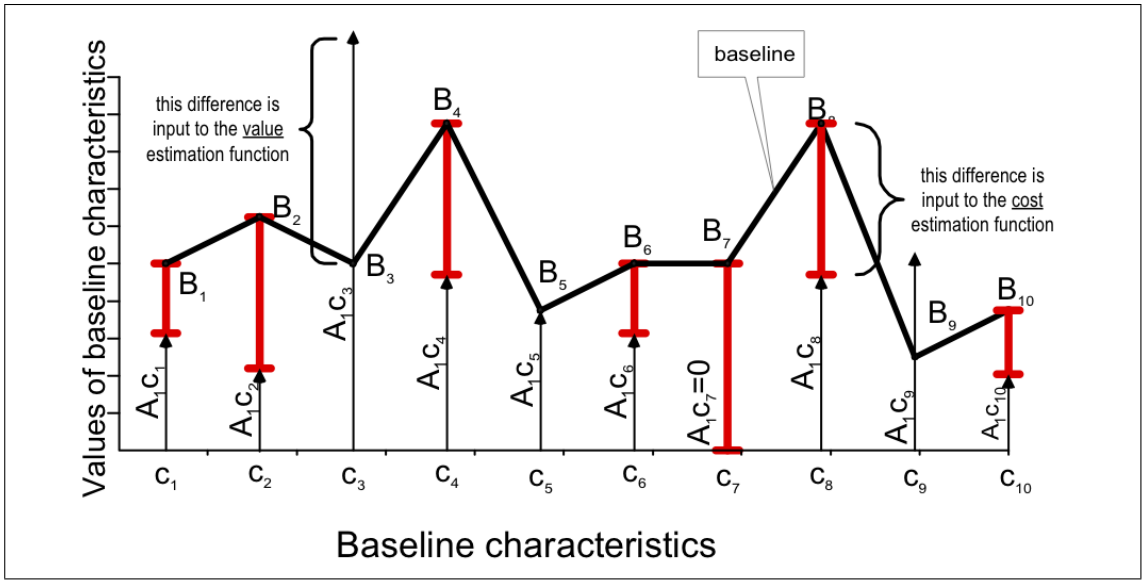


Figure 2.19: The baseline estimation principle (Kontio, 1995)

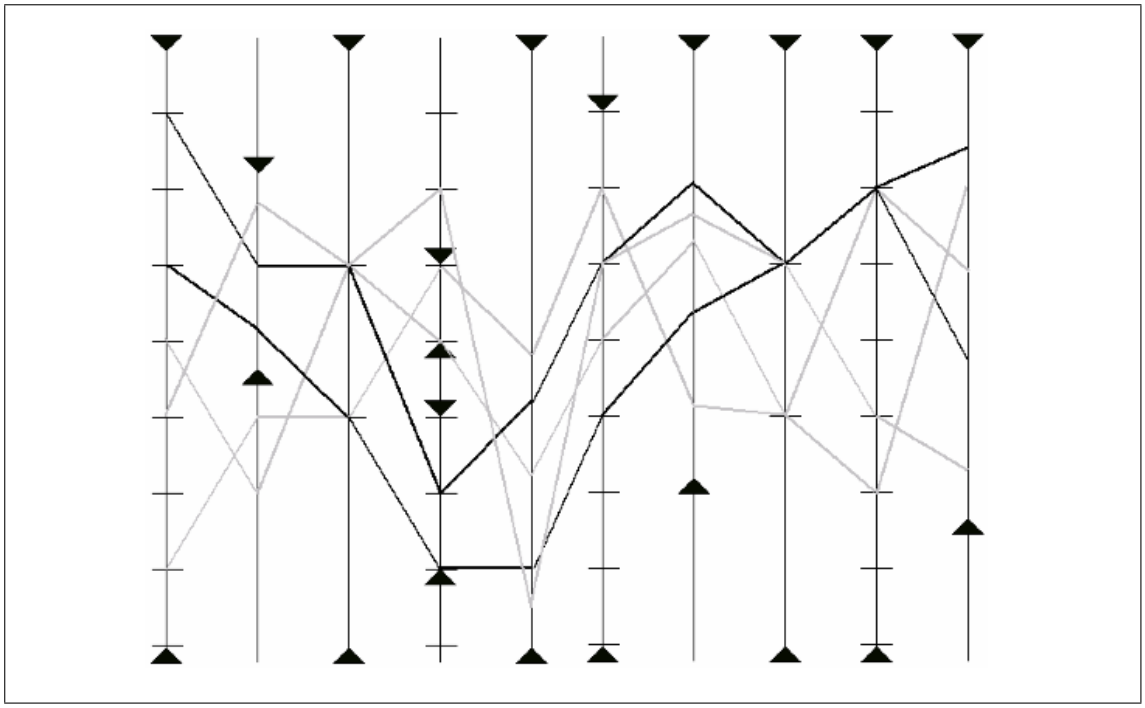


Figure 2.20: Example of graphical interface to collect integrator's data (Clark and Clark, 2007)

The use of AI techniques for retrieving code and software artefacts from repositories includes the application of rough-fuzzy sets (Rao and Sarma, 2003), neuro-fuzzy search robots (Kuo et al, 1999), fuzzy-subtractive clustering (Nakkrasae et al, 2004) and entropy-based fuzzy k-modes (Stylianou and Andreou, 2007). These help to sort the software into clusters based on their descriptions.

Other work focuses on providing tool support for searching. Maracatu is a tool for search and retrieval for software developers based on faceted information including platform, component type and component model (Garcia et al, 2006). Brou (2005) implements a search tool which utilises XML descriptions and XQuery to allow browsing of a repository by field.

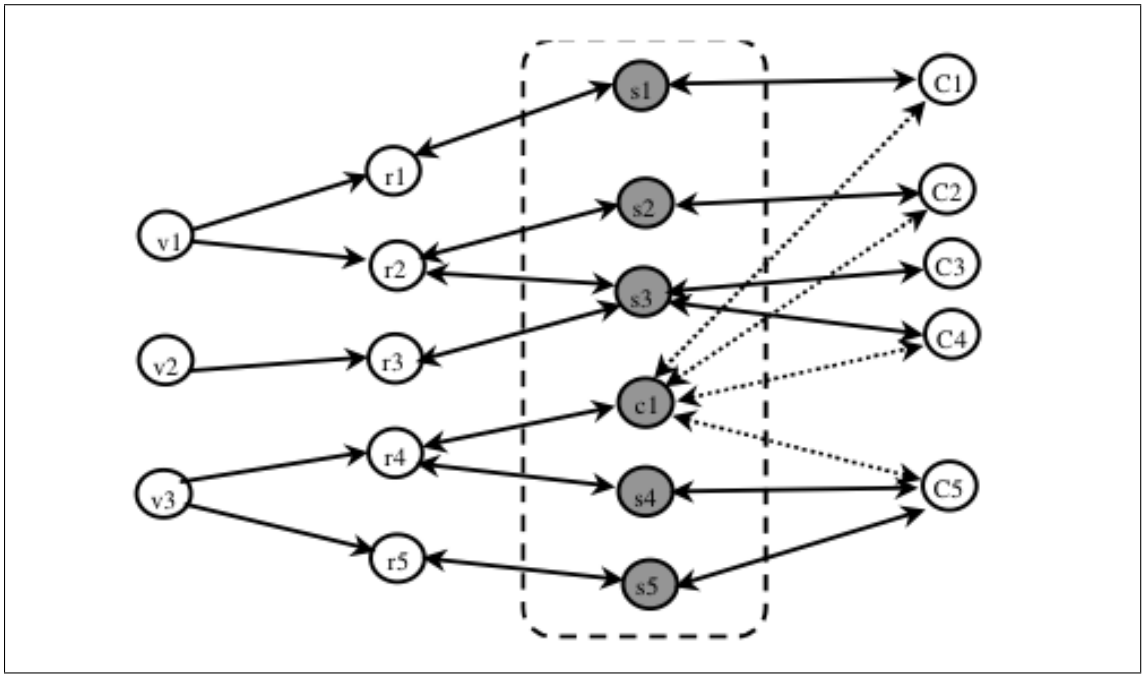


Figure 2.21: Status of the decision support system after two settings (Kotonya and Hutchinson, 2005)

Andreou et al (2006) encodes components for classification in binary strings representing all characteristics of interest. A genetic algorithm is used to discover several different classifiers, which can be used to match components with similar characteristics up to a set threshold (e.g. 40%). The classifiers were tested on a generated dataset of 1000 components.

Mohamed et al (2007b) have developed the MiHOS tool for handling mismatches in COTS selection. It provides a portfolio of qualified solutions and provides interactive

decision support. For criteria, the authors suggest OTSO or Incremental Quality Model Construction (IQMC). Provides a measure of mismatch which can be focussed on a specific attribute, for example security. Neubauer and Stummer (2007) modify OTSO, replacing AHP with multi-objective decision support. The approach gives a visual tool to allow the selective exploration of thresholds on criteria and their impact on the values of other criteria in the remaining candidates. The thresholds can be seen in Figure 2.22 across six criteria.

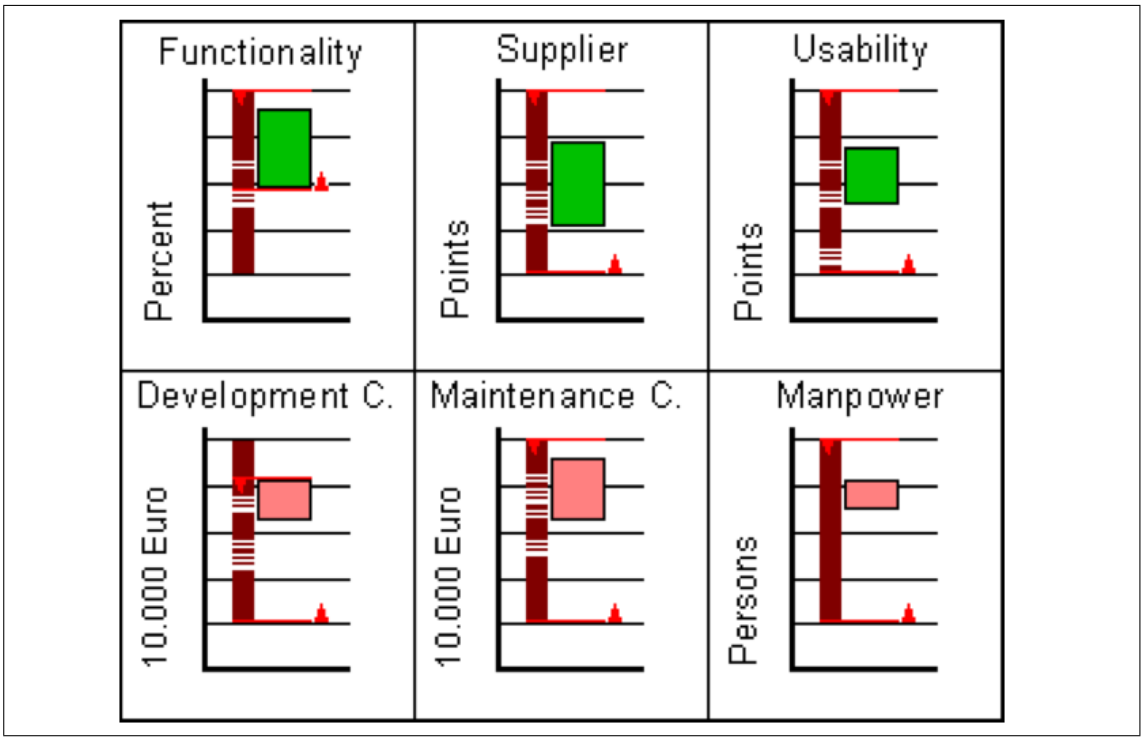


Figure 2.22: Status of the decision support system after two settings (Neubauer and Stummer, 2007)

Andreea Vescan and Grosan (2008) uses an evolutionary approach for multi-objective view of component selection. Dependencies are handled by ensuring that all individuals in the initial population satisfy them. Encoded as a string of requirements, they can be applied to a set of components. They use a greedy algorithm and evolutionary techniques on the 'chromosome'. Lill et al (2005) developed the CoExSel to support trade-off analysis between functionality, costs, reliability and time to market. Component searching is based on matching individual requirements with those in repositories. Provides facility for adding experience data to the component record. Uses formulae from the literature for estimating costs, reliability and effort (time to market) and black box java classes

and packages for evaluation.

One of the areas where computational intelligence has been applied is in the retrieval of components from in-house and open source libraries. In some cases this requires a formal specification to allow classification, for example Z notation and component modelling technique (CMT) (Nakkrasae et al, 2004). Their reuse model is made up of a repository of formal specifications of components and retrieval mechanisms based on a component similarity value. Brou (2005) queries open source program libraries using XML and UML descriptions. Doxygen software is used to generate information on internal structure, which is possible with source-code available in OSS projects. A domain thesaurus is used to create an XQuery request for retrieval.

Kuo et al (1999) note the issue of semantic vagueness and multiple interpretations in software engineering decision making. They apply neuro-fuzzy models to create a search robot for personalised software component retrieval. It relies on keyword translation and supervised learning with user input. Fuzzy neural networks are used to provide an adaptive thesaurus. Nakkrasae et al (2004) use a formal specification of components (Z and CMT) to allow fuzzy subtractive clustering to group components in the repository. They define a component as X = {S,F,B}, structural, functional and behavioural, each of which is multidimensional. Rao and Sarma (2003) works on an in-house repository. They use case-based reasoning on eight attributes with linguistic variables on each, one of which is 'specification'. The candidates are grouped into equivalence sets based on rough-fuzzy membership values. Stylianou and Andreou (2007) use fifteen categories/attributes from (Andreou et al, 2006) and look at part of the repository selected using entropy-based clustering. These entries are clustered using a fuzzy k-modes algorithm.

Key points: There has been some work on tool support for component selection, primarily around the use of AHP and visualising decisions. Computational intelligence is being applied to the retrieval of components from libraries, however they are usually in-house and source code-aware.

2.3 Critique

There have been a number of published processes for component based development, and even more for selecting COTS and software components. There is no indication

that any of these have become preferred by industry. This is supported by the study by Li et al (2005), which indicates that most companies do not use a formal process for component selection. The question is then, why are they not being used and what needs to change to increase uptake? It is well know that architectural mismatch has been an emergent difficulty in CBSE (Garlan et al, 1995). Kotonya and Hutchinson (2005) state that the nature of CBSE requires that the selection process be documented. In an area of greater challenge than in-house non-component-based software engineering, a systematic approach must be taken to software development - so to have no process is not an option.

The survey by Li et al (2005) indicates most companies are using traditional software processes, with no adjustment for the use of OTS. The most common approaches for discovery of COTS included: searching the Internet, taking customer suggestions (which may be a constraint) and hands-on trials of a few candidates. It is difficult to say how representative the sample is; however it seems feasible given other issues with component selection. The exception to this may be where in-house processes are mandated by the organisation, as in the case of SEL at NASA (Morisio and Tsoukiàs, 1997).

The Navarrete et al (2005) study indicates that the agility of published processes is not high. As agile methodologies increase in popularity, this would flag a clash of development approaches. Rifkin (2003) suggests that this in itself would cause problems as the successful adoption of a new process depends on how well it fits with existing procedures in the organisation.

There may also be a social aspect to this lack of uptake: developers are used to searching the Internet for recommendations for many problems they face, and thus have taken to a quick search and information from some trusted sources to short-cut through a thorough selection process.

A further issue, or a related one, is the time it takes to go through a thorough, manual process. In Kontio (1995), which is the basis for many selection processes, the selection is manual, with a team of people each considering a volume of pairwise comparisons. In Ruhe (2002) the author evaluates the readiness of ten published CBSE processes for using Decision Support Systems (DSS). In essence, the processes are too manual to use, and are not geared for automation and DSS.

Many of the selection processes allow for internal iteration as requirements and selection criteria are adjusted as the result of tradeoffs (Comella-Dorda et al, 2002, Kotonya

and Hutchinson, 2005). Fewer of the papers refer to the need to return to the process in the maintenance stage, with Reifer et al (2004) listing the maintenance lessons surrounding COTS. While Kotonya and Hutchinson (2005) focus on the impact of change in COTS systems, there is little support for the developer in the return to the selection process.

Components are part of a culture of reuse of which repeatable, reusable processes are a part. In Sassi et al (2003), a range of processes are taken through a mapping lens to assist in selecting the right process for a given selection task. Processes will vary from task to task as criteria change for each software acquisition. The most similar instances of a process will be the original selection and any subsequent maintenance re-selection, which has not been addressed in the literature.

Key to discovery and automation of tasks is a description of the requirements and of the candidate components. In this case, industry leads with the schemas developed to describe the software in their repositories (Christiansson and Christiansson, 2004). Although there is no standard amongst the repositories, it is possible to gain an understanding of what industry and users consider valuable information when looking for software. Characterisation has been discussed in the literature, and includes formalisation of descriptions for repositories in Nakkrasae et al (2004) which is suited for in-house use, but unlikely to be adopted by industry.

There have been criticisms of the techniques used in component selection. Ncube and Dean (2002) considers WSM and AHP and the reasons they are not advisable for the aggregation of component scores. Although alternatives have been put forward, including Outranking (Ncube and Dean, 2002) and mathematical programming (Neubauer and Stummer, 2007), there is a danger of increasing complexity and reducing understanding from the user perspective.

A key issue in CBSE is architectural mismatch (Garlan et al, 1995). This reinforces the need for context in the shortlisting of components, while Weyuker (1998) argues that all components should be tested, and tested in context.

Some work applying AI and automating processes has taken place. Of existing processes, Ruhe (2002) would indicate they are not suited to automation, although that is what is need for those selecting components. More can be done, particularly if the underlying process is developed with AI and automation in mind. Understandability can

also be improved with tools, with a few processes including some visualisations - again, more can be done.

With consideration of the preceding discussion, the following aspects are considered gaps in the component selection literature and practice:

- Lack of a standard specification
- Lack of uptake of existing processes
- Lack of support for system evolution (revisit selection process)
- Wide use of basic and possibly unsuitable techniques
- Need for context to be central
- Few tools for automation and decision support.

These will be the guiding issues for this study as the strategies are developed to support component selection.

2.4 Summary

This chapter has reviewed the CBSE literature along with the literature considered relevant to the research problem. The key issues that the investigation addresses are: specification; suitability of process; support for evolution; alternatives to aggregation; support for context; and, tool support. These align to the four Research Elements. The following chapter discusses the methodology used to carry out the investigation.

Chapter 3

Research Procedures

This chapter outlines the approaches used to develop, implement and evaluate strategies for software component selection. The four research elements under investigation drive the choices described in this chapter. An outline of the philosophy, approach and methodology is provided. The research takes an exploratory approach and employs a mixed methodology which includes quantitative and qualitative research methods. An adaptation of the Spiral Development Model provides the framework for the investigation including the development of the strategies for software component selection and the implementation and evaluation of the solutions. The Spiral template used to plan and guide the course of the work is given in Section 3.2, and is a contribution of the investigation (C9).

The context of the research includes the component marketplace, literature and the characteristics of the researcher. New and existing software used in the investigation is the critical instrumentation for the work. Data collection and treatment are discussed to describe the external data used in the project, and the processing and formats involved. Evaluation of both the process and the product is undertaken at various levels of granularity throughout the project: internal to each Spiral, at the end of each Spiral, and for the overall investigation. A framework for this evaluation is described, and a summary of the Spirals undertaken in this project is provided at the end of the chapter to illustrate and preview the approach taken.

3.1 Methodology

All research is based on assumptions about reality, how it is perceived and understood.

Scientific Philosophy: *The investigation of questions that arise from reflection upon science and scientific practice.*[1] is one approach to how we come to know about the world.

Knox (2004) proposes a hierarchy of research needs in which the student must consider all levels of the underlying philosophy, relevant paradigms, research methodology and specific techniques used. Although there are patterns and traditions in the choice of approach, each project must look at the applicable alternatives afresh.

For research at the doctoral level Knox suggests that critical analysis is required to inform how the philosophy impacts on each element within the research process and the relationships across the whole research process. Philosophy thus informs the researcher through reflection upon their respective sciences and the paradigms existing within them.

As noted in Chapter 1, the problem being addressed in this research is:

> *What strategies and techniques can be developed to support the selection of third party software components?*

The four elements of this research build the specification and process, then focus on strategies for selection and evaluation. They are listed in Chapter 1 and reiterated here:

- **RE1:** *Development or extension of a template for the specification of components*
- **RE2:** *Development of a process for the selection of software components*
- **RE3:** *Investigation of and implementation of strategies for the shortlisting and evaluation of suitable software components*
- **RE4:** *Evaluation of the effectiveness of the template, process and strategies via case studies*

The nature of the research question that underpins this work lies at a junction of the disciplines of software engineering, computer science and information systems. Each of these has evolved with some differentiation of axiology, methodology and ontology, creating a complex environment for defining a specific philosophy or paradigm to explain their particular environment. In light of these complexities, the approach, methodology and methods for this study are informed by the following discussion of philosophy and paradigms.

[1] http://www.answers.com/topic/philosophy-of-science

A paradigm is the underlying assumptions and intellectual structure upon which research and development in a field of inquiry is based (Kuhn, 1996). Since the publication of Kuhn's 'The Structure of Scientific Revolutions' in the 1960s, paradigm has been used in science to refer to a theoretical framework - exemplars of how research is done in a domain. Each paradigm is comprised of three parts, ontology, methodology, and axiology (Kuhn, 1996). The ontology provides the language for the paradigm, the terms for the objects and their relationships. Each paradigm has accepted methodologies and techniques (such as experimentation within the scientific paradigm). Methodologies are recognised ways to conduct a study within the paradigm, and are discussed in greater depth later in this Chapter. The third component is the axiology, the rules for making judgements and decisions within the paradigm. The axiology describes the value structure for the paradigm. This can be viewed along three dimensions: systemic, extrinsic and intrinsic values (Hartman, 1969).

Research is carried out to help understand, explain and predict phenomena in the Real World. Philosophies often used in scientific research are Positivism/Post-positivism, Critical Theory, Pragmatism and Constructivism[2] (Easterbrook et al, 2007). Multidisciplinary fields, such as computer science, software engineering and information systems are difficult to align to a single philosophy, with particular studies using a 'best fit' to inform research methodologies. An alternative is to consider combined approaches such as methodological pluralism or the use of methodologies from an 'opposing' philosophy to add rigour or context (Knox, 2004). In mixed methods design, strategies from different philosophies are used within a project to give a wider range of coverage and fuller picture of the study than may otherwise be possible (Bonoma, 1985)

Positivism and Post-positivism have emerged in science and state that all knowledge must be based on logical inference from a set of basic observable facts. The main difference between the two is the objectivity of the researcher to the study (Figure 3.1). In Positivism (a.k.a. Realism), there is an assumption that an objective reality exists, independent of the researcher (Crossan, 2003). Positivist research deals with measurable behaviour and approaches a problem by breaking it down and understanding the parts. For the positivist, empiricism - observation and measurement - are the core of scientific research. The 1950s produced an alternative view to research, where all research is

[2]Constructivism and Interpretivism are considered interchangeable in some of the literature

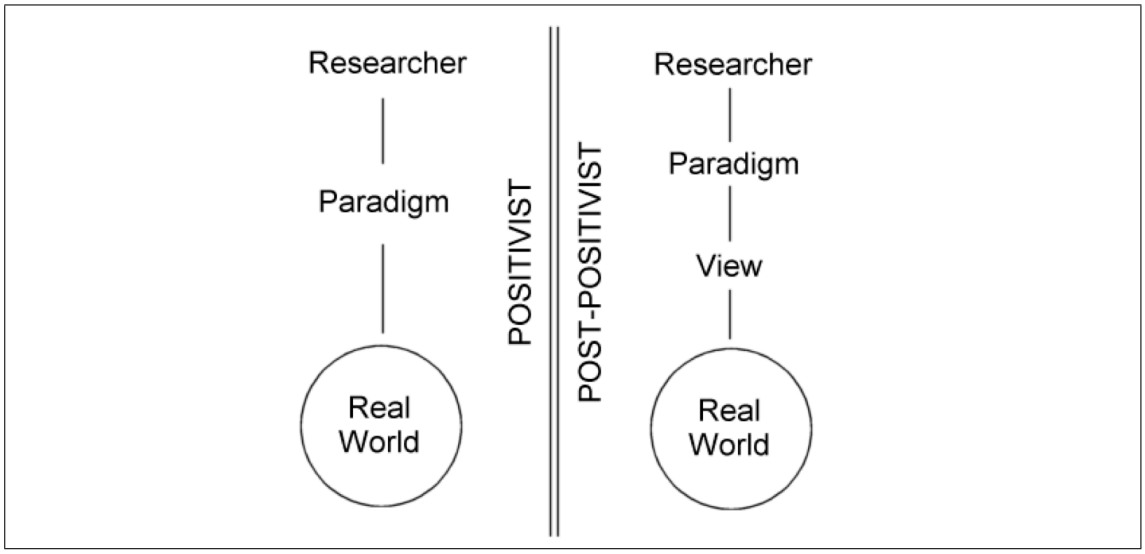


Figure 3.1: Positivism and Post-positivism

based on a *view* of the Real World. This Post-positivist (a.k.a. Anti-realist) movement still believe in absolute truths, however all theories are provisional (Kuhn, 1996) as the perception of reality is fallible. Falsification of theories is still considered valid, but for the Post-positivist researcher all research must be considered in context. Observations can be taken using multiple techniques to triangulate measurements, which may account for errors and give greater confidence. Moving further from absolute truth, Interpretivists consider all knowledge to be subjective and socially constructed, and rely on the participants' view of the situation (Creswell et al, 2003).

Each of computer science, software engineering and information systems has issues identifying with specific paradigms, partly because each is an applied area of research. Computer science aligns with the sciences and takes a positivist or post-positivist approach (Johnson and Onwuegbizue, 2004) with the expectation that researchers will seek truth using the scientific method - exploring hypotheses through experimentation. However, computer science can also be seen as a blend of science and engineering (Dodig-Crnkovic, 2002) which is evidenced by the variation in faculty that computing schools are placed in. Software engineering exacerbates the difficulty in classifying a discipline into science or engineering in that a solved science problem becomes an engineering problem, which may in turn become (or generate) science problems (Lázaro and Marcos, 2005). The information systems aspect of software engineering might be expected to move away from the science viewpoint. However, a survey of information systems journal papers

from 1983-1998 found that 97% were considered positivist (Mingers, 2001). Paradoxically, Tichy (1997) reports that in computer science journals, around 50% of software related articles had no empirical evidence for their claims. This confused state of affairs is often attributed to computer science and software engineering being 'young' when compared to traditional sciences. It may be that the real cause is the lack of flexibility and applicability of the paradigms that are being applied. This may also contribute to computer and information sciences recording the lowest PhD completion rates of all study programs (after 10 years)[3].

Although this study may resemble Post-positivism, it is important to also consider whether a Pragmatic approach may fit better, aligning with a solution-oriented approach. Creswell (2009) states that a Pragmatic approach is real world practice oriented, problem centred and utilises a mixed methodology. This view is supported by Easterbrook et al (2007) who states that pragmatists use any available methods to shed light on the issue under study. Pragmatic views in science require little analysis of philosophy as the researcher is free to choose a framework that best suits the outcome required, and this drives the choices in the research methodology, strategies and outcomes. Pragmatism supports pluralistic approaches: Creswell (2009) notes that several authors have utilised pluralism in focusing attention on the research problem. While a Pragmatic approach is partially able to capture the requirements of this research, the problem is complex, requiring more than a straight forward problem-solution determination. Pragmatism in this instance would reduce the ability of the researcher to show how the decisions, artefacts and results found during the process of the study are as important as the product of the study.

It could be argued that there is some alignment with the Easterbrook et. al. (2007) definition of a Critical Theory view for this research question. He states that 'research is a political act' and that it may be used to overcome societal obstacles. Thus, for example, many users of software products are at the mercy of developers and producers and would possibly be 'emancipated' through improved selection of third party software components.

The research question fits best into a post-positivist view of science, as, although there is subjectivity in interpretation of results and path of inquiry, the human factors

[3]http://www.phdcompletion.org/quantitative/book1_quant.asp

are minor compared to the computation and data manipulation involved. There are some aspects of the work that lean towards Interpretivism. In the beginning of the study the researcher is not able to state in advance what will be learned. Knowledge will be constructed as a response to information and processes that are encountered in a variety of contexts, unknown at the beginning of the exercise. It requires the interpretation of data produced over the course of the study, and utilisation of emergent ideas to inform choices to progress the research. It is the focus of the research that puts the study as post-positivist: it is the effectiveness of the selection strategies that will be evaluated, not how they are chosen or accepted. The researcher remains open to techniques from other philosophies, if they suit the area under consideration. Figure 3.2 provides a summary of the philosophy and implications for framing the research that follows.

Aspect	RE 1	RE 2	RE 3	RE 4
Philosophy	Post-positivism			
Domain	Software Engineering / Computer Science / Information Systems			
Approach	Exploratory Spiral Development Model			
Methodology	Qualitative	Qualitative	Qualitative & Quantitative	Qualitative & Quantitative
Methods and Techniques	Survey (Literature)	Survey & Case Study	Case Study & Quasi-experiment	Case Study & GQM

Figure 3.2: Research design used in the study

Having established that the study is not directly engineering a solution to a problem, an approach is taken to ensuring that knowledge is progressively developed. This requires establishment of discrete and disciplined points throughout the study where reflection and review, evaluation and record keeping occur. The research approach is therefore exploratory and not restricted to techniques that have been applied in a similar context.

This research models the component selection process and aims to assist and enhance that process. The philosophy requires the researcher to interpret data with an expert approach to the problem: evaluating research elements for the development of strategies to assist the selector (application developer). The researcher is intrinsic to the investigation

and affects the directions and outcomes of the research.

Consideration of the project as a whole sees it as a progression of theory building (process and strategies), implementation and evaluation, which define the project approach. The exploratory nature of the work requires iteration and refinement. Methods such as grounded theory (Glaser and Strauss, 1967) are of interest as the data and observations drive the research as a theory is developed. Grounded theory is not applicable in this project, as it requires that the researcher not drive or bias the work. However, this investigation has drawn inspiration from grounded theory with respect to iterative theory-building.

The use of systems development as a research methodology has been argued (Nunamaker et al, 1991, Hasan, 2003), particularly in the field of information systems. Nunamaker et al (1991) promotes systems development as one of four research strategies which can provide an integrated, multi-method approach to information systems research. The three complementary methods are: theory building, observation and experimentation. Within systems development, Nunamaker lists five principal parts: constructing a conceptual framework; developing a system architecture; analysing and designing the system; building the system and experimenting, observing and evaluating the system. A more contemporary view refers to parts three and five as prototyping and technology transfer, and indicates that there is iteration and overlap in the process. The crucial element to justifying systems development as a research method is the focus on knowledge creation (Hasan, 2003), rather than the code as an end in itself.

While computing projects may have a well-defined path, to approach this research problem in a planned way was considered high risk. The Spiral Development Method is a software development methodology that structures iteration and monitors risks (Boehm, 1988). The SDM iterates through four sectors: Determine objectives; Identify and resolve risks; Development and test and Plan the next iteration (Figure 3.3). With a conceptual shift to broaden the 'Development and test' sector to activities including strategy building, case studies and experimentation, the SDM reveals its potential as a structure for conducting research.

The SDM has been used to provide an incremental approach to knowledge building through the development and implementation of strategies in this project. Each iteration of the SDM is a well-defined cycle with consideration of stakeholders needs, alternatives,

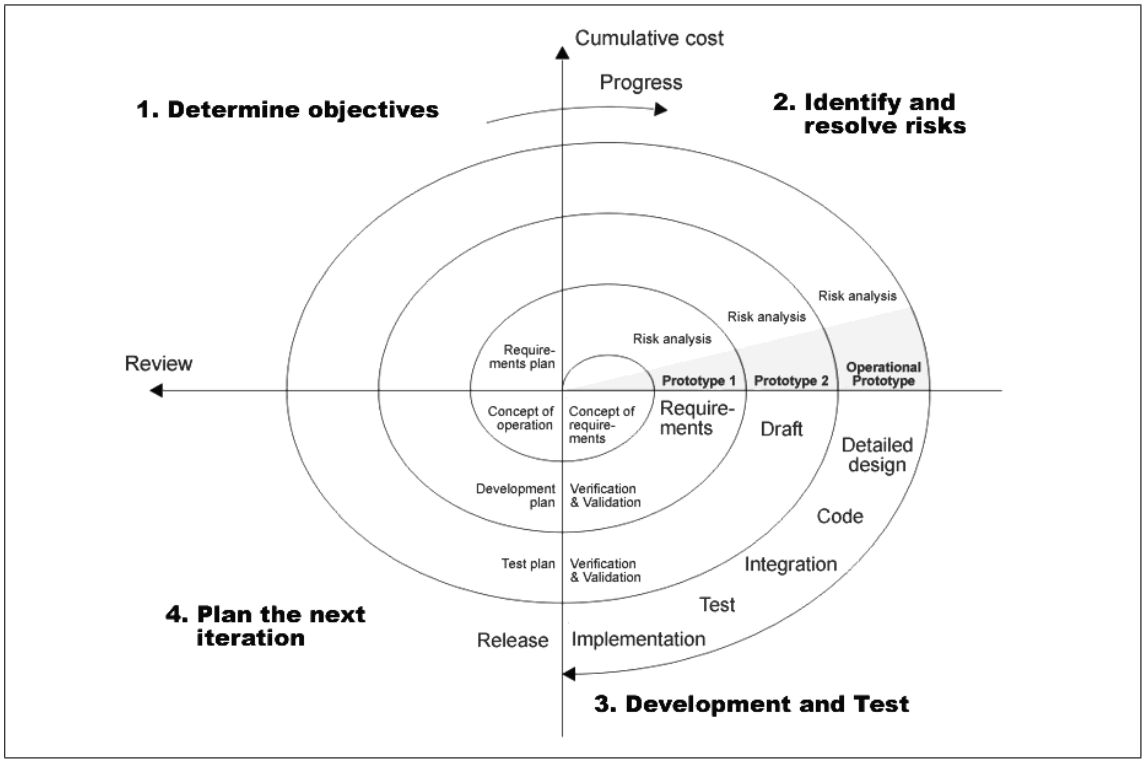


Figure 3.3: Spiral Development Model (Boehm, 1988)

Quantitative	**Quantitative** or **Qualitative**	**Qualitative**
Experimentation	Case Study	Grounded theory
Quasi-experimentation	Survey	Phenomenological
Correlational	Action Research	Historical
Ex post facto		Content analysis
Observation		Ethnography
Developmental study		

Table 3.1: Research methodologies

risks, validation and review of outputs. To provide information for some level of repeatability in this project, an exposition of the context of all decisions is required. The spiral documentation allows the reader to see how ideas developed and the justification for choices that were made.

Research methodologies are often classed as qualitative or quantitative (Leedy, 2002). This relates to the type of data being investigated, and the goals of the research. A summary of methods is given in Table 3.1.

Where a researcher is trying to explain or predict trends in data, or the behaviour of a population, quantitative methods methods will be applied. These typically deal

with numeric or structured data which can be processed via statistical techniques compatible with a Positivist philosophy. Experimentation is the most rigid of quantitative approaches, where the known variables are controlled, and the variable(s) of interest are systematically manipulated to assess their effect. The experiment is a repeatable process to support or disprove a stated hypothesis. Where the variables cannot all be controlled, less rigorous techniques, such as quasi-experimentation, can be used. Two other techniques which can be quantitative or qualitative, depending on the data and its treatment, are case studies and surveys. However, case studies consider a particular problem in depth, and may not be open to generalisation.

Qualitative methods consider textual or complex data. They view the researcher as an instrument that affects the outcomes of the research. The assumptions and influence the researcher brings to the study must be declared. Qualitative research may involve survey or observation of information sources across time (historical), media (content analysis) or the community (ethnography). Guidelines for qualitative studies exist (Leedy, 2002), all relying on the researcher to reflect, then assert the findings (in contrast to statistical analysis). The results of the research are context sensitive, making it difficult to replicate a study.

The literature on research methodology often considers qualitative and quantitative research as opposites and implies that a particular philosophy, science or paradigm can only utilise a particular group of methodologies. More recent discussions (Crossan, 2003, Knox, 2004, Esteves and Porter, 2004) state that multi-method approaches can combine qualitative and quantitative methods and add rigour through the triangulation of results. For example, quasi-experimentation with human subjects can be reinforced by surveys and interviews. This can provide better understanding of the experimental results, or quantify the trends seen in the surveys.

In order to refine the approach it is essential to discover the affinities between the research areas, philosophies and methodologies. As this study is in software engineering, accepted techniques must be considered. As a science in a people-oriented domain, much of the research to date has followed social science techniques, or put forward unquantified theories (Tichy, 1997). However, there is a movement towards increasing rigour in software engineering through experimentation and empirical results (Kitchenham et al,

2002, Wohlin et al, 2000). Assisting the move to quantitative approaches is benchmarking (Sim et al, 2003), the development of metrics (Pfleeger, 2001), and the compilation of historical information for research and process improvement. A widely applied method for generating measurable data is the Goal/Question/Metric (GQM) approach (Basili et al, 1994).

The GQM is an industry standard for creating metrics linked to goals. The approach was originally developed for evaluation of defects in NASA projects, and was expanded to a larger context (Basili et al, 1994). It is particularly useful for: understanding and baselining an organisation's software practices; guiding and monitoring software processes; assessing new software engineering technologies; and evaluating and certifying improvement activities (DACS, 2011).

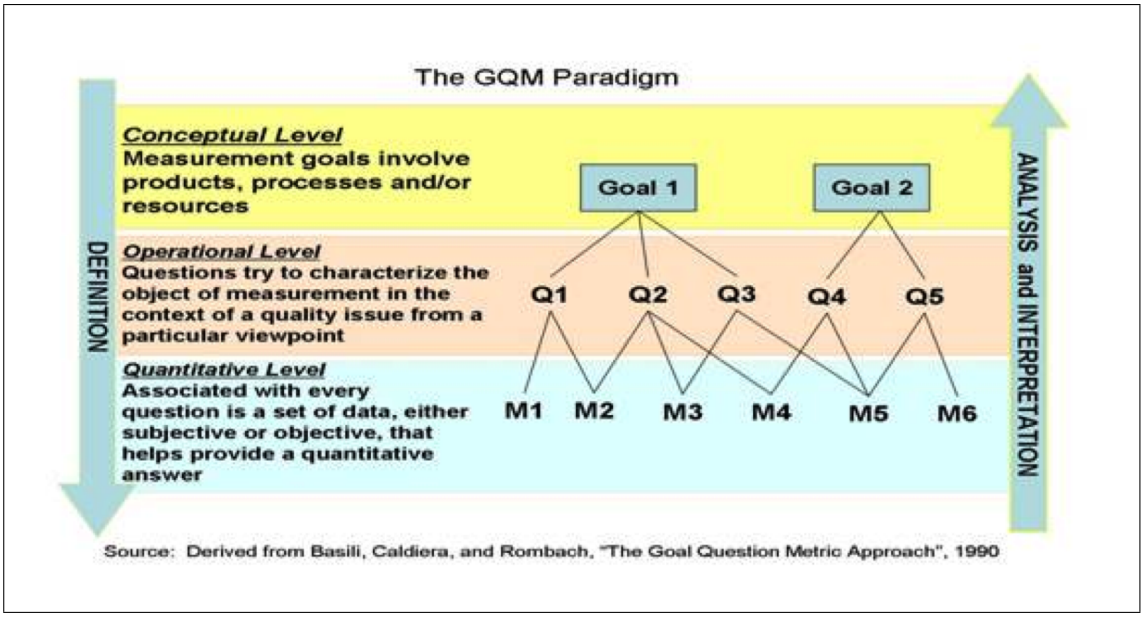


Figure 3.4: GQM paradigm (DACS, 2011)

The measurement model for the GQM has three levels: conceptual (GOAL), operational (QUESTION) and quantitative (METRIC) as shown in Figure 3.4. The result is a hierarchical structure starting with goals, which are elaborated into a number of questions, which are themselves split into related metrics. The metrics can be objective or subjective, and depend on the viewpoint from which they are taken.

The GQM lends itself to a wide variety of applications beyond software processes and can be compared with the Balanced Scorecard approach for organisational strategy (Buglione and Abran, 2000). The CBSE literature includes the application of GQM,

SPIRAL 4	**Purpose** **Issue** **Object** **Context**	**Evaluate** **effectiveness of** **strategies for data representation** **Spiral 4**	**Results**
Goal 4A	Focus Viewpoint	Quality: *Enhance shortlisting for more accurate results* Quality Assurance personnel	
	Q4A1 Q4A2 Q4A3	Has Spiral 4 improved results? Are the updates well documented? Is the process repeatable?	YES YES YES

Table 3.2: GQM Summary - Spiral 4 (extract)

most often as the method for developing criteria and/or evaluating COTS or components (established by (Kontio, 1995)).

One of the benefits of applying GQM is that it can be aligned with the SDM, with the steps falling into Sectors of the Spiral. For each Research Element, the investigative approach, data considered and evaluation techniques vary. **RE1** used a qualitative survey of existing component characterisations. The process for selection (**RE2**) was developed with a similar survey approach, and evaluated through a case study. With **RE3** the strategies and their effects were assessed via case studies including quasi-experimentation. The case studies provided context to focussed exploration and evaluation of the potential solutions. These quasi-experiments were the limit of the rigour in the project as the subject matter cannot be controlled to the level of true experimentation. The final Research Element (**RE4**) reviews the entire project and is qualitative, with quantitative aspects drawn from the use of GQM. This eclectic, multi-method approach was summarised in Figure 3.2, which shows the entire investigation (**RE**1-**RE4**) and the research design used.

An example for this project is given in Table 3.2. The purpose of Spiral 4 is to evaluate the effectiveness of the strategies developed for data representation. This is indicated in the header of the table, along with the context, which is 'Spiral 4'. From there, the GQM evaluation focuses on each of the goals. In this example, Goal 4A is targeting quality, which is defined for this Spiral as 'Enhance short-listing for more accurate results'. The viewpoint used for the quality goal is that of quality assurance personnel. To allow an assessment of the strategies in terms of quality, the goal is broken down into three questions. Each response is used in the overall evaluation of the satisfaction of the goal.

3.2 Approach

This investigation has quality amongst its goals and thus aims to formalise the selection process. Key to uptake of processes is their usability. The value added must be greater than effort involved and automation through tool support can assist with this. From the title of this thesis the use of strategies, enabled by the application of artificial intelligence techniques, is fundamental to the work. At the same time, the aim is for the process, tools and evaluation to be dynamic to adapt to changes in the user's context. It is also of value to consider the selection process at a more abstract level - allowing for reuse through alternate implementation and application. To this end, the investigation develops a pattern and an implementation for the selection process, following current documentation standards. These ideals form the basis of the goals aimed for in this investigation.

3.2.1 The Spiral

The Research Elements are addressed through the Spirals that make up the project. The investigation began with the definition and commitment to the Research Elements. From there, a series of cycles took place through analysis of information at hand; formulation of strategies; their implementation; and review and planning. This iterative approach was guided by the construct of the SDM.

A generic Spiral is now described, which is then instantiated in the respective Spiral sections. The terminology used in each Spiral has been adapted to suit the research context. Each Spiral follows a template made up of the four sectors and their component parts: Objective Setting; Risk Assessment; Development and Validation; and Planning. These map to the four sectors on the SDM diagram (Figure 3.3) with slight changes in titles. The Spiral Summary (Section 3.8) describes all seven Spirals and includes tables summarising the planning, implementation and review of the included activities.

The SDM template used in this study follows:

Sector 1: Objective Setting

The first sector defines the objectives for the current iteration. This involves the definition of the problem(s) dealt with in the Spiral and instantiating the project goals for the

current Spiral. The stakeholders involved in the Spiral are indicated, along with their acceptance criteria. It forms the basis of the the later review and evaluation of the Spiral outcomes.

Problem Definition: The Problem Definition details the task to be undertaken. It includes any scope, limitations or assumptions particular to that task.

Goals: The goals of each Spiral represent the desired outcomes for the solution developed for the problem. These goals are used for determining the approach taken, as well as being the basis for the GQM evaluation at the Spiral and Project levels. The project has six high-level goals: an attempt is made to honour these in each Spiral. These are described in more detail under Evaluation in Section 3.7. At the Spiral level they are instantiated and a more detailed and specific set of goals defined.

Stakeholders: The stakeholders identified throughout the project are described in Table 3.3. This project is aimed at a real world need and is thus driven by the stakeholder needs. In each Spiral, the Win conditions - acceptance criteria - are considered for each stakeholder. The stakeholders for the entire project are the application developers, component developers, component brokers, quality assurance and academia. The stakeholders are the drivers for the goals of each Spiral in the SDM.

Stakeholder	**Description**
Application Developers	The direct users of the results of this project are the application developers. They will be assessing third party components for their specific application and context, and will need documentation to support their decisions.
Component Developers	Marketability depends on their components being selected for use by application developers. They will be looking for a facility that allows them to hook into this testing and ranking process with minimal effort or change to their existing products.
Component Brokers	As the enablers to component buying and selling, they are interested in what they may be able to add to their user tools. They may need to provide options for application developers to download and test numbers of components before purchasing, or allow testing to be carried out on their servers.
Quality Assurance	Many organisations have auditing of processes to improve the quality of their operations. We need to provide documentation of the process and decisions to satisfy QA requirements.
Academia	This is the main driver behind this research and the shape it takes. Academic concerns affect time limitations, focus on academic merit (as opposed to commercialism) and the methodology used.

Table 3.3: Stakeholders for project.

Sector 2: Risk Assessment

The risk sector considers the problem being addressed and the context of the work. An analysis of the risks that may arise is included, and the responses are detailed. In each Spiral, the context includes the outputs of the previous Spiral, along with any preferred approaches to attain Spiral goals.

Risks are defined and then assessed to help support later decisions in terms of risk and context. A risk is defined as 'the potential for realization of unwanted, negative consequences of an event' (Rowe, 1977). To be considered a risk, there must be: a loss associated with it; uncertainty or chance involved; and/or, some choice involved (Charette, 1989). Activities undertaken in risk management include identification, strategy and planning, assessment, mitigation/avoidance, reporting and prediction (Karolak, 1996). It can be helpful to go through common risk factors when identifying risks. Factors may relate to: organisation, estimation, monitoring, development method, tools, risk culture, usability, correctness, reliability or personnel (Karolak, 1996). Each risk is then assessed for risk exposure. Myerson (1996) defines risk exposure as the probability of an unsatisfactory outcome (UO) multiplied by the loss the UO would create.

For this project, the risk assessment is synthesised into tables identifying risks, probabilities and strategies. An example (summary) is shown in Table 3.4. The various risk factors are considered in the first column, resulting in a list of identified risks in the second column. For each, the probability is estimated, shown in the third column. The final column indicates strategies chosen for dealing with each risk (e.g. avoid, minimise, contingency).

Each risk is then monitored to detect whether the strategies need to be applied. The probabilities and effects are reviewed throughout the project. For each Spiral, there are risk management tables following the described templates. This provides a level of understanding of the task and prepares the way for the feasibility to be assessed.

As a result of the risk assessment, it is possible to make an informed assessment of the feasibility of each planned task. Where there are risks, a decision is made as to how to progress. This may include restricting scope for the Spiral or the project as a whole.

Risk Type	Risk	Probability	Action
Technology	Issues with XML software	Moderate	Avoid: Survey and trial software
People	Lack of experience in XML	High	Minimise: Training and tutorials
Organisational	Change in funding and/or facilities	Low	Contingency: None
Tools			
Requirements	Changes to requirements causing rework	Moderate Moderate	Avoid: Thorough research Minimise: Data model to handle change, use of versioning
Estimation	Time required underestimated	Moderate	Contingency: Ask for expert help

Table 3.4: Risk assessment and strategies example from Spiral 1

Sector 3: Development and Validation

Development of strategies, experimentation, coding and testing take place in this sector as an iteration through procedures, data analysis and refinement. These tasks are typical of a research project, and are an extension of the SDM. Once satisfied with the current iteration, a trial is undertaken, usually in the form of a case study (in this project). In some of the Spirals, all steps of Sector 3 were repeated one or more times in an internal iteration. This provides the opportunity to continue a line of research to build on observed results.

As this study is academic research, some changes have been made to the activities and what is recorded for each Spiral. The following information and activities would typically be given for a research project as a whole. In this case each Spiral is somewhat independent, with varying emphasis on instrumentation, procedures, analysis, data and trials. Thus, as part of development and validation, the following will be actioned and documented:

Instrumentation: Tools and facilities required including inputs and processing, along with justification of choices.

Specific Procedures: Procedures used in this sector are detailed for each Spiral. They include surveys, quasi-experiments and case studies.

Data Analysis: The methods of data analysis are defined. This is used to determine the effectiveness of the procedures.

Refinement: In some cases a selected option for problem solution does not work as expected, or alternatives and enhancements become apparent during the analysis. At this point the solution can be refined and the process iterates through Specific Procedures, Data Analysis and Refinement until criteria are satisfied or greater problems are identified.

Trial: Each deliverable of a Spiral is trialled and assessed. These may be manual or automated trials, or a combination.

Data Collection: Data for trials is accessed from various sources, an existing case study, real world data or simulated data.

Treatment of Data: Any processing of data is included in this section, including that done using tools developed for the project.

Sector 4: Planning

With the investigations and trials complete a review process is undertaken to assess the success of the work. Criteria from Sector 1 are used to evaluate the success of the Spiral. The results of the evaluation are considered in the review of the Spiral, before planning for the next Spiral. The two activities in Sector 4 are:

Evaluation: The assessment process is carried out using a GQM evaluation and qualitative assessment. Identified risks are also considered, including whether they occurred and the impact of the event.

Review and Planning: Once the work is evaluated, it is time to consider the next stage in the project, with reference to the outcomes of the current Spiral. This may involve changes and enhancements to the current work.

3.2.2 Research Elements and Spirals

The following discussion outlines the flow of the project, which unfolded as each Spiral was undertaken. The specification and process address the first two Research Elements and provide a foundation for the strategies in the later Spirals (Table 1.1). **RE3** was the focus of Spirals 3-6. The final element, **RE4**, is addressed through a case study and the reflection on the collected results of all the Spiral evaluations in the Conclusion.

In Spiral 1 the focus was on the specification of the component, which potentially could enable or limit the strategies to come. The second Spiral developed the Process.

Spiral 3 investigated ways to apply artificial intelligence to the selection activity, while Spiral 4 made improvements by enhancing the representation of data. The CdCE Process was completed in Spiral 5 where metrics, testing and evaluation were implemented. Automation of aspects of the Process presented the opportunity to improve the representation of the choices through the ClassifierSuite in Spiral 6. Once the Process, procedures and tools were in place, a final case study was undertaken, with the results of this and all other Spirals feeding into the evaluation. How this contributes to the overall project is shown in Figure 3.5, with different branches of the tree being built up over the course of the investigation.

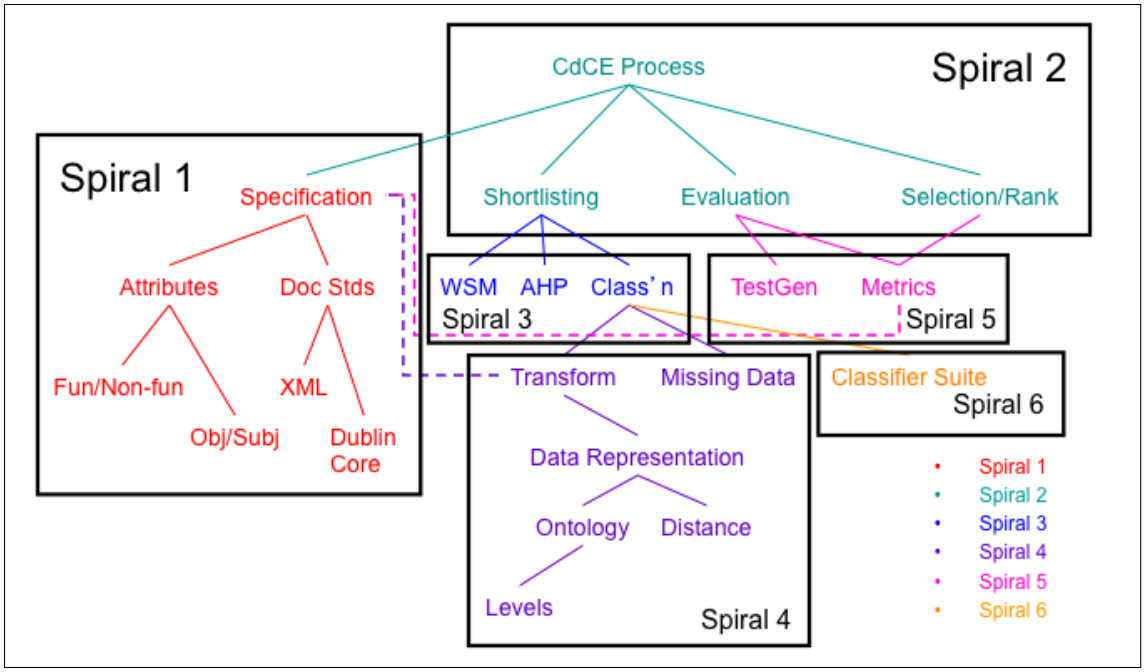


Figure 3.5: Tree representation of Spiral development throughout the thesis (colour-coded)

3.3 Context of Study

This study has been carried out in the context of the third party software marketplace from 2001-2011. The resultant system has been assessed using case studies to evaluate the merits of the concept of providing such a tool to assist component selection. Facilities used included Windows and Mac computers, a range of software tools and development environments and Internet access. Sources of information and guidance were journals, books, websites, conferences and academic and industry contacts.

As this is a Post-positivist study, researchers are considered part of the context - contributing a perspective to the design and analysis of the work. The researcher has an undergraduate degree in computer science with industry experience in database, Internet technologies and maintenance of large third party information systems. The researcher has also lectured in Software Engineering, providing a strong theoretical basis and perhaps shifting from a Computer Science perspective. Teaching areas in software engineering included formal methods using Z notation, software design and metrics. Later in the project the researcher took on a research role in grid computing for data mining, and subsequently in eResearch and supercomputing research and education.

3.4 Instrumentation

The nature of this investigation has required the implementation and use of many software applications and tools. A number of Java applications have been written for this project, along with scripts for linking them together. Most of the programs developed in Java as pipe/filter applications to be run in batch mode. The ClassifierSuite is an interactive program, also developed in Java and utilising the Abstract Windowing Toolkit(AWT)[4] for the user interface. The programs and scripts are summarised in Table 3.5. Third party class libraries were used in many of these programs, particularly for the handling of XML data. In addition, other third party software and applications were used as the development environment for coding the in-house software and for assessing and checking results. These are listed in Table 3.6. The Weka application has been used during development and is integral in the implementation of the CdCE Process.

Common throughout the project is the use of XML[5] and Z notation (Spivey, 1992). XML allows for the interchange of structured data in a machine-independent way. It is the de facto standard for data exchange. Z notation is a formal specification language able to describe component behaviour and interfaces in an abstract manner.

While the researcher is a part of the context, she is also an instrument used in the study. The researcher came to the project with programming, software engineering, statistics and web design skills. Through the course of the project, additional skills in XML processing and proficiency with tools (e.g. Weka) have been developed.

[4] http://download.oracle.com/javase/1.4.2/docs/api/java/awt/package-summary.html
[5] http://www.w3.org/XML/

Item	Description	Spiral
XML schema	Schema to describe ideal and candidate components	1 to 7
XSLT scripts	Scripts to reformat the XML files and make them more readable for the user	4 to 7
Intelligent	Java program developed to read in XML ideal specification and output training data in Weka's ARFF format	3 to 7
CdCETransformer	Java program developed to read in XML ideal specification and real world data and output the data in required format	4 to 7
FM2CdCE	Java program developed to read in XML real world data (freshmeat) and output the data in CdCE XML format	4 to 7
Trove2CdCE	Filter to take in freshmeat Trove and generate the XML file for the ontology	4
TestGen	Java program developed to take in the technical specification (Z notation) and generate a test suite based on equivalence classes	5 to 7
ClassifierSuite	Java program to visualise and explore results of running multiple classifiers to see the impact of criteria choices	6 and 7
Bash shell scripts	Scripts written in Bash Shell to automate the processing of data and the collation of results	4 to 7

Table 3.5: Software applications and scripts developed during the investigation

Item	Description	Spiral
XML	eXtensible Markup Language	1 to 7
Z notation	Formal specification language	1 to 7
XMLSpy	XML editor and validator	1 to 3
Jess Expert System	Expert system allowing the definition of rules and the evaluation of data	3
Java 2 SDK	Virtual machine for running Java programs	3 to 7
Borland JBuilder	Software development environment for Java	3 to 4
XMLWriter	XML editor and validator	3 to 6
Weka Machine Learning	Java implementation of machine learning algorithms	3 to 7

Table 3.6: Third party software tools and languages used

3.5 Data Collection

In Spiral 2, the data was collected manually from the ComponentSource[6], Tucows[7] and Flashline[8] software repositories. Manual collection involved accessing the website for the repository and using the search facility provided to locate and then compile a list of candidates. This not only provided data on the candidates for software selection, but also on the data models used and the search/selection tools provided by the sites.

For Spiral 3 onwards, it was necessary to have access to full metadata records for one or more repositories to work on automation and scaling of the CdCE Process. While a component repository would have been preferred, this was not possible[9]. The freshmeat repository of open source software projects was considered adequate for demonstrating and developing the selection process and tools.

freshmeat provided access to the XML metadata for their repository, along with a formal description of the data model. Another option was to simulate a component repository by developing multiple in-house components. This was at odds with the 'real world' intent of the project, and could potentially cause problems with objectivity. Access was also available for the SourceForge[10] repository, but required too much time to understand the PostgreSQL database and compile it into a flat table.

freshmeat has facility for access to the repository via RDF files[11]. Two exports of the data were taken two years apart, the first with 33,262 entries and the second with 41,885. Of principal interest was the file fm-projects.rdf which included full information for all projects. In addition, the fm-trove.rdf file included the repository's comprehensive ontology - the Trove. It was decided to align to the freshmeat Trove as the ontology for this project, rather than developing a new, and likely less complete, ontology.

External sources of data are listed in Table 3.7. Each Spiral chapter (Chapters 4 onwards) includes information on the data processing specific to it. Data representation and associated processing is the focus of Spiral 4, with the details recorded in Chapter 7.

[6] http://www.componentsource.com/

[7] http://www.tucows.com/

[8] Flashline was bought out by BEA Systems, to become part of their Aqualogic SOA stack. BEA has since been acquired by Oracle

[9] ComponentSource refused access to their data for commercial reasons. Tucows focuses on shareware applications rather than components. For these reasons an alternative repository was needed

[10] http://sourceforge.net/

[11] URL to full repository is http://freshmeat.net/backend/fm-projects.rdf.bz2 although they are moving to a new API which will replace this

Item	Description	Spiral
Component Source	Website supporting component development and brokerage	2
Tucows	Website repository for commercial software	2
Flashline	Website repository for commercial component brokerage (now part of Component Source)	2
SourceForge	Website hosting open source projects	2 to 3
Freshmeat	Website hosting open source projects	3 to 7

Table 3.7: Data sources used

3.6 Treatment of Data

The format and storage for the data throughout the project has used XML (primarily), RDF, ARFF and text files. XML is the format of choice for the project as it is machine readable and the standard for interoperability. RDF is the standard used by freshmeat, and is built on XML and XML DTD. To better present the XML files, a suite of XSLT scripts were developed. For example, the testing phase (Chapter 8) includes XML files for recording the results of tests. As the tests were executed manually, the XML was converted to a web page using XSLT.

Weka can accept ARFF files for input and outputs the results as text files. The ARFF input files were generated by two of the Java applications written for the project: Intelligent and CdCETransformer. Intelligent generates the training data while CdCETransformer takes the data from the freshmeat repository (in CdCE format) and outputs it in ARFF to match the training data. As the automation of the experiment increased, these Weka output files were post-processed by a Bash Shell script and collated into existing XML data files to create summaries and shortlists.

A transformation was required to enable the freshmeat data to be used with the CdCE tools. This was done via the FM2CdCE filter, written in Java, and outputting an XML file. A similar process took place for the Trove via Trove2CdCE, generating an XML file. More information on the transformation process is included in Spiral 4 (Chapter 7).

Since the tools were developed to implement the strategies under investigation in this project, they are described further in the appropriate chapters.

3.7 Evaluation

The work in this project has been reviewed and evaluated throughout, most directly through case studies and quasi-experiments embedded in each Spiral. Tables holding information that relates to evaluation of the research project are indicated by double lines on the borders.

A review took place at the end of each Spiral before planning the next package of work. This involved an assessment of the stakeholder Win conditions; applying the GQM; and undertaking the review and planning sector for each Spiral. An external level of review was facilitated through peers - in presentations and papers published.

As noted, a review of each Spiral has been carried out based on GQM. This allows a structured approach to the evaluation across each and all Spirals. Use of GQM results in a hierarchy of evaluation criteria rooted in the project (or Spiral) goals. Figure 3.4 provides an example of a section of a GQM tree.

Each Spiral of the investigation in the thesis includes an evaluation based on six criteria of interest throughout the work. These underlying themes have influenced the decision making: **quality**, **usability**, **intelligence**, **innovation**, **dynamics** and **reuse**.

Quality: The overall goal in software engineering is to improve quality in software. This criterion is included to ensure that each spiral of the investigation has a commitment to quality - through the use of standards, development of processes and choice of solutions.

Usability: The researcher has a strong view that software engineering must provide useful, applicable solutions to real world problems. In this project, automation and AI techniques are to be applied to enhance the selectors' understanding of the decisions they are making and they can be easily applied (through the automated approaches) to larger datasets.

Intelligence: This refers to the application of AI techniques and automation to enhance the CdCE Process. This may be through the development of tools or by the use of standards to make the application of AI easier. For example, the XML specification is machine readable which makes it easier to develop code to utilise AI techniques.

Innovation: As a PhD thesis and as a useful research project, innovation is always required. This is through the development of new strategies and tools or by applying existing techniques in novel ways.

Dynamics: As an approach with real world applicability, the investigation needs to

SPIRAL 4	**Purpose** **Issue** **Object** **Context**	**Evaluate** **effectiveness of** **strategies for data representation** **Spiral 4**	**Results**
Goal 4A	Focus	Quality: *Enhance shortlisting for more accurate results*	
	Viewpoint	Quality Assurance personnel	
	Q4A1	Has Spiral 4 improved results?	YES
	Q4A2	Are the updates well documented?	YES
	Q4A3	Is the process repeatable?	YES
Goal 4B	Focus	Usability: *Provide tools and knowledge base for users*	
	Viewpoint	Application developer	
Goal 4C	Focus	Intelligence: *Apply ontologies and knowledge management to shortlisting*	
	Viewpoint	Application developer	
Goal 4D	Focus	Innovation: *Include knowledge management and missing data treatment*	
	Viewpoint	Academia	
Goal 4E	Focus	Dynamics: *Allow for update and substitution of knowledge base*	
	Viewpoint	Application developer	
Goal 4F	Focus	Reuse: *Where possible make use of existing code and artefacts*	
	Viewpoint	Application developer	

Table 3.8: GQM Summary - Spiral 4

provide flexibility and accommodate change. This change may be through the ability to modify aspects of the CdCE Process (how it is implemented) or in the user's need to be able to revisit and revise selection tasks as their system evolves.

Reuse: A common theme in software engineering is reuse - particularly in CBSE. Reuse can also include methodologies, patterns and tests. Throughout this project there is a drive for reuse in its many flavours.

In terms of the GQM model, goals are further described in terms of their purpose, issue, object, context and viewpoint (Basili et al, 1994). The purpose in applying GQM in this project is to evaluate the effectiveness of the work in each Spiral. The object for each evaluation is how it has addressed the problem definition for the Spiral under evaluation, in the context of that Spiral. The focus is the respective goal category. Each of the goals has been assigned a stakeholder to indicate the viewpoint used - quality is taken from the quality assurance view, innovation from the academia view and the others are viewed as the application developer. This is summarised in the top section of Table

3.8.

Each of the goals is interpreted into a more contextualised goal at each of the Spiral evaluations. For example, the goals for Spiral 4 are listed in Table 3.8. This is then split into a series of questions, as shown for Goal 4A. The questions are answered with reference to the evidence from the project. In most cases this will be a subjective result - an indication of whether evidence shows the item was addressed. For this reason, metrics are not listed in the evaluation. With this information from each Spiral, and consideration of the project as a whole, a thorough evaluation is possible. Chapters for Spirals 1 to 4 include a 'Post-Spiral Update', where the related work in later Spirals is discussed, indicating how it affects the products and contributions for that Spiral.

3.8 Spiral Summary

The SDM has been used in this project, not only for software development, but also for structuring the approach to implementing and evaluating strategies. As such, this flow has influenced the choices made in the project and is key to understanding the overall outcomes.

The four tables that follow summarise Spirals 1-7 according to the format used by Boehm et al (2003). Stakeholders across all Spirals have varying levels of direct interest in the outcomes.

Spiral 1

Table 3.9 summarises Spirals 1 and 2. Spiral 1 aims to develop a component specification on which to base the rest of the investigation. Key aims are to adhere to specification standards. The specification was developed through review of existing schema and standards and the application of the alpha version schema to sample components. This work was carried out in 2001, with revisions to the schema as later Spirals required them. Decisions such as the use of XML and alignment to Dublin Core are still valid at the time of writing. This part of the project was presented at the Young Researchers Workshop at the International Conference on Software Reuse (ICSR 2002) (Maxville, 2002) and discussed with Dr Ruben Prieto-Diaz, A/Prof. Bill Frakes and Prof. Bertrand Meyer at that conference. These discussions influenced and reinforced the work.

	Spiral 1 - Specification	**Spiral 2 - Process**
Stakeholders	Brokers, Schema Developers and indirectly Component Developers, Application Developers and Quality Assurance	Application Developers, Quality Assurance and indirectly Component Developers, Brokers and Schema Developers
Objectives, Constraints and Priorities	Characterise components utilising standards; consider existing schemas used in industry (Dublin Core); prepare for the automation requirement need for process	Develop an intuitive, iterative, repeatable process suited to automation; must utilise or enhance specification from Spiral 1; include future testing in context
Alternatives	Use existing schema	Use existing schema or undertake ad-hoc selection
Evaluation Risks	Incompatibility with common standards; new community standard supersedes this work	Lack of access to datasets; a new standard process is accepted by industry; process is too prescriptive and complex to be used
Risks Addressed	Maximise compatibility; detach process from specification - make it replacable	Secured access to datasets early in spiral
Risk Resolution	Compatibility confirmed, process and strategies to be replaceable	Generic process so strategies can be experimented with; utilised a range of datasets through their front-end interface
Product Elaboration	Create XML specification for component with adherence to standards	Define steps in process for software selection; use specification to describes two types of component - ideal and candidate
Process Elaboration	Research existing academic and industry standards for component characterisation, map between models	Survey existing processes in the literature and informal survey of developers - looking for essence of selection process
Verification and Validation	XML validators; Sample COTS/ Component descriptions; peer review at ICSR (Maxville, 2002)	Case study of manual application of process; peer review at APSEC (Maxville et al, 2003b)
Commitment	Use specification for rest of project, revise as needed	Process is basic framework to allow experimentation on strategies for the rest of the project; revise if required
More Info	Chapter 4	Chapter 5

Table 3.9: Spiral Summary - Spirals 1 and 2

Spiral 2

The focus for Spiral 2 was on developing an intuitive, iterative, repeatable process for selection that would be suited to automation. Existing processes were considered, along with informal interviews with local software developers on the processes they used. The CdCE Process was developed to include most aspects of existing processes, along with test generation and evaluation. At this point, the Process supported context-based evaluation through the use of metadata and testing the candidates in the target environment (context). The CdCE[12] Process was trialled through manual application of the process to a selection task. This work was published and presented at the Asia-Pacific Software Engineering Conference (APSEC) (Maxville et al, 2003b) and benefitted from feedback received at the conference.

The CdCE Process was successful in the case study and trials and became the foundation of the rest of the investigation. Particularly useful was the separation of steps in the process via static XML files. This provided self-documentation, but also enforced

[12]Previously referred to as CdCT - Context-driven Component Testing

the decoupling of steps. Through low coupling, changes made to implement the AI and automation strategies were less likely to ripple throughout the previous work.

Spiral 3

With the specification and process defined, the focus turned to developing strategies to apply within this selection framework. Spirals 3 and 4 are summarised in Table 3.10. Two broad strategies were considered: automated test generation and support for shortlisting. In Spiral 3 the shortlisting strategies were explored including issues in filtering and evaluation, specifically:

- Problems with suitability of evaluation techniques (Ncube and Dean, 2002)
- Lack of automation or suitability to be automated (Ruhe, 2002)
- Issues with adoption of processes (Li et al, 2006).

In particular, the evaluation techniques used in the literature often used WSM or AHP, which present issues associated with aggregation and assumptions of independence, as discussed by Ncube and Dean (2002). It can also be extremely time-consuming to work through all the pairwise comparisons required.

A wide literature review of potential AI techniques was undertaken, which initially indicated that expert systems were appropriate. Trials with Jess (Friedman-Hill, 2008) were successful, but required manual generation of rules. One issue for the project was that every selection task was different, so there was no long term learning option between subsequent selection tasks. Sample training and test data was developed and tested with the Weka Machine Learning application (Weka) (Hall et al, 2009) application and two of its tools: an Artificial Neural Network (ANN) and the C4.5 classifier. Tests with the ANN indicated that it needed a two-layer network to be able to distinguish when there was an interplay of criteria. Comparative trials took place within a case study and showed similar performance (in error percentage) between C4.5 and ANN. C4.5 was selected as it output rules which could be interpreted by people, as opposed to the black box of the ANN.

The training and test data was generated from the ideal specification in a manner similar to test case generation. Different aspects of Spiral 3 were published and presented at the International Conference on Software Engineering & Knowledge Engineering (SEKE)

	Spiral 3 - Shortlisting	**Spiral 4 - Data Representation**
Stakeholders	Application Developers, Quality Assurance and indirectly Component Developers and Brokers	Application Developers, Quality Assurance and indirectly Component Developers and Brokers
Objectives, Constraints and Priorities	Focus on shortlisting; main options AI, testing and iteration; context considered key; implementation of initial strategies - AI and iteration	Enhance shortlisting via improved data representation; apply techniques to real world dataset; implement ontology
Alternatives	Expert systems, neural networks, Analytic Hierarchy Process (AHP), WSM	Stay with existing (text/boolean) comparisons; choose different dataset; develop own ontology
Evaluation Risks	Poor choice of AI; access to data; integration issues; complexity of implementation	Lack of access to datasets; enhancements not effective; integration issues - ripple effect
Risks Addressed	Literature review for AI selection + experimentation and exploration; datasets defined early; all experimentation within one step of process	Early requests for data; research into techniques for data representation; continue with independence of applications
Risk Resolution	Experimentation and exploration of AI options led to C4.5; Secured local copies of dataset from freshmeat backend; minimised impact of changes to other part of process through defined interfaces and parameter files	freshmeat worked well, but not ComponentSource or SourceForge; gradual exploration through transformations 1-5; effective pipe-filter with static input files contains ripple effect
Product Elaboration	"Intelligent" - reads in an XML ideal specification and exports training and test data in ARFF format for Weka	"Intelligent" - revised; Transformer - conversion of data to bring into format for Weka; scripts for automation
Process Elaboration	Looking for a way to set up rules for deciding on shortlisted components - how best to generate human-understandable rules to automate decision process; defined requirements for AI tools and surveyed widely for suitability	Many iterations of data representation and missing data handling
Verification and Validation	Tested suitability through made datasets; small tests and comparisons on manually generated dataset; peer review at SEKE (Maxville et al, 2004b) and COMPSAC (Maxville et al, 2004c)	Thorough experimentation on various transformations, ideal specifications and real dataset; comparison with previous results; case study presentation and paper at PEECS (Maxville, 2005)
Commitment	Proof of concept worked - could improve data representation and semantics; need a larger dataset	Transformation 5 found to provide best recall/relevance, will use for all future tests; shortlisting complete, now looking at other parts of process
More Info	Chapter 6	Chapter 7

Table 3.10: Spiral Summary - Spirals 3 and 4

(Maxville et al, 2004b) and the IEEE International Computer Software and Applications Conference (COMPSAC) (Maxville et al, 2004c). Identified issues were: the loss of information from the component metadata which could have been used; and, inadequate handling of missing data in the freshmeat dataset.

Spiral 4

Spiral 4 concentrated on data representation as a result of the review of Spiral 3. The solution at this point was equivalent to an SQL query providing a Boolean output of match/no match. As much richer information was available, Spiral 4 aimed to exploit this. Another issue was the handling of missing data (fields within records) in the repository.

A parameter was added to allow the user to choose which value was used for missing (default is -999) and whether to interpret a missing value as a match, partial match or non-match. While the default is non-match, the facility to set various options allowed exploration of how missing data was handled.

In Spiral 3 the attributes were numeric, date, text and descriptive text (longText). The text-based fields were of most interest and fell into two clear categories: those that could have standardised values (ontology), and those that were less predictable (freeText). FreeText entries include the software name and the developer name. Ontology attributes are based on a restricted vocabulary which uses a thesaurus to standardise terms. The terms are organised into hierarchies for each criterion to represent the relationships between the terms. LongText entries are fuller descriptions which are dealt with using information retrieval techniques such as removal of stop words and punctuation, and calculations based on recall/relevance calculations.

Each attribute type (class) has its own method for assessing the closeness of a match, implemented via polymorphism in Java. Five 'transformations' were developed and compared, beginning with the basic T1 (boolean as in Spiral 3) through to T5 (ontology with levels). Within each transformation (T1, T2, T3, T4, T5) is a facility to include an increased amount of information from the original dataset. Trials and then a case study were able to differentiate between the transformations, with T5 exhibiting the highest recall and relevance. This work was summarised in a paper and presentation at the Postgraduate Electrical Engineering & Computing Symposium (PEECS) (Maxville, 2005). As a result of this work in data representation, T5 and the default missing data handling are used for all subsequent work.

Spiral 5

Table 3.11 summarises Spirals 5 and 6. With strategies in place for shortlisting, Spiral 5 focussed on fleshing out the remainder of the steps in the CdCE Process. Metrics had been determined in earlier work, so the implementation of the steps aligns to them: Functional Fit (`FFIT`), Functional Excess (`FEXS`), Adaptation Effort (`AEFT`), Testing Fit (`TFIT`) and Test Result (`TRES`). The main strategy in this Spiral was the implementation of the test generator, based on the behavioural specification. The Z specification is parsed to find interfaces and type information. The test generator uses this information to generate a

	Spiral 5 - Evaluation	**Spiral 6 - ClassifierSuite**
Stakeholders	Application Developers, Quality Assurance and indirectly Component Developers and Brokers	Application Developers, Quality Assurance and indirectly Component Developers and Brokers
Objectives, Constraints and Priorities	Implement automated tools to support steps 4-8; context-based testing from behavioural specification	Provide support for selection process; model on manual approach found helpful during case studies
Alternatives	Static evaluation; other forms of test generation/ranking	Halt development at Spiral 5; try statistical selection support - e.g. PCA
Evaluation Risks	Poor choice of test generation technique; integration/ripple effect	GUI more difficult than expected; tool not helpful; integration issues - ripple effect; time required for this additional task
Risks Addressed	Basic testing approach used; continue with independence of applications	Iterative development on key functionality first; continue with independence of applications; set time limit on development
Risk Resolution	Able to implement basic test generator; reused classifier for evaluation; XML for links between steps	Able to make functional application with key viewing tools to explore data; independent from rest of process and tools; development and case study complete within set time
Product Elaboration	TestGen - test generation tool; XML schemas and XSLT; scripts for automation	ClassifierSuite - tool for assisting selection of shortlist; guidelines for use
Process Elaboration	Outlined overall approach; worked through each step with emphasis on user view of process	Defined useful tools from researcher experience; three test sets for comparison
Verification and Validation	Thorough experimentation; full case study; described as pattern for software selection in peer reviewed IET Software (Maxville et al, 2009)	Thorough experimentation; full case study; peer review at IEEE CEC (Maxville et al, 2008)
Commitment	Implementation of evaluation allowed for full proof of concept on CdCE process; pattern for software selection	Useful tool to assist selection; key for using process on larger datasets; possible application for other selection visualisation
More Info	Chapter 8	Chapter 9

Table 3.11: Spiral Summary - Spirals 5 and 6

test suite based on equivalence partitions. Equivalence classes can be the default (for common variable types) or user defined. Users can also link in real data to generate test cases.

A key aim of the project was to include context in the testing and evaluation. This is done via the context schemas in the Z Specification. The schemas allow the base functionality testing to be extended by adding test to target usage (`CX_U`), stress (`CX_S`), reliability (`CX_R`) and performance (`CX_P`). It also allowed for sequences to be included.

When testing was expected to be the focus of the project, the aim was to include oracle functionality from the Z specification. It was realised that the evaluation could be effective if built from a basic behavioural specification: the user defined interfaces; partitions on types; and sample data. A full Z specification to oracle level was considered a barrier to uptake, but can be addressed in future work.

Spiral 6

Spiral 6 aimed to provide a tool to assist in the selection of criteria for the shortlist. With automation, all combinations of criteria choices are able to be evaluated and all results considered. This was set up in the scripts supporting the investigation and used to get a fuller understanding of the results. The researcher had developed diagrams to extract key information from the results and to help support decisions made in the case studies. In some cases two quite different sets of criteria could be combined into a shortlist, which had not been an outcome when working manually through the criteria.

The ClassifierSuite tool renders the results across the power set of the selection criteria (e.g. `{{A,B,C},{A,B},{A,C},{B,C},{A},{B},{C},{}}` unioned with the mandatory criteria). The developer can then include/exclude a criterion, or click through on the results of a set of criteria to see the raw XML. Methods to derive various statistics, to annotate the graph and to save/print add to the usefulness of the tool.

The risk with Spiral 6 was the additional time required. The researcher considered the tool to be very important for supporting selection and enforced a strict deadline for its development. The resulting trials on three existing case studies were published and presented at the IEEE Congress on Evolutionary Computation (CEC) (Maxville et al, 2008).

Spiral 7

In Spiral 7, the evaluation for the full project was the focus, as required for **RE4**. Some evaluation had already taken place through the final sector of each of the Spirals. Spiral 7 was constrained to the work that had taken place in the previous Spirals - a case study to review the effectiveness of the strategies that have been implemented. The Spiral is summarised in Table 3.12. Working through real world application of the strategies was specified in **RE4**, also constraining the alternative approaches available for this Spiral. Within the constraints, it would have been possible to apply a piecewise evaluation of the strategies, or to utilise external evaluation. The piecewise approach was not taken as it had already been applied in the previous Spirals. One or more external evaluators would have been a preferred approach, however time and administrative limitations made that impractical.

In this Spiral, the major risk was around the selection of the scenario for the case

	Spiral 7 - Project Evaluation
Stakeholders	Application Developers, Component Developers and Academia
Objectives, Constraints and Priorities	Evaluation of the Process through a case study. Constrained to using the freshmeat data and the CdCE Process as developed in previous Spirals
Alternatives	Options include alternative scenarios, piece-wise evaluation rather than as a whole, use of external testers
Evaluation Risks	Problems with chosen scenario - not exercising all Steps, poor candidates in shortlist
Risks Addressed	Have alternative scenarios available if required
Risk Resolution	Selected case study scenario (emailer) did result in an unsatisfactory shortlist - none of the programs called be installed and executed. XML editor scenario brought in as a replacement
Product Elaboration	N/A
Process Elaboration	Followed CdCE Process as documented
Verification and Validation	Complete case study, GQM; peer review IET Software (Maxville, 2009)
Commitment	Satisfied with case study as providing a good representation of the CdCE Process
More Info	Chapter 10

Table 3.12: Spiral Summary - Spiral 7

study. From Spiral 3 onwards, the major scenario for case studies was the 'emailer'. This had worked well in the shortlisting efforts and was mature in terms of Z specification and usage models. However, when the shortlisted candidates were downloaded, it was found that they were all poorly supported and could not be installed. Some of the issue was the open nature of the repository projects - there is no insurance that they will be maintained, and most were in a neglected state. The scenario itself may have added to the issue, as the combination of C++ and emailer may have been an unlikely choice. A number of alternative scenarios were followed through to ensure that strong candidates would be available for evaluation. The result was that the XML editor scenario was used in the case study.

With the scenario hurdle resolved, the case study was undertaken and the Process and tools worked well. The switch to a completely new scenario and working through all the steps occurred more quickly than expected: the fresh selection task and the panic of having the worst risk take place probably worked together for this result. The case study met all of the requirements and gave a good representation of the project. The Spiral then moved to the collation and reflection on each of the Spirals and the investigation as a whole.

This section has provided a short overview of Spiral 1 to 7. Further details on each of Spirals 1-6 are found in Chapters 4 to 8, with Spiral 7 in Chapters 9 and 10.

3.9 Summary

This chapter outlined the perspective taken for addressing the research problem. The investigation has systematically addressed each of the research elements, utilising the SDM to provide structure to an exploratory, Post-positivist study. The SDM supports the regular review and reflection required to manage risk where each iteration is dependent on the results of the previous work. The GQM has been applied to add rigour and consistency to the evaluation. This chapter included the Spiral template and summary information for each of the Spirals. The chapters that follow expand on each of the Spirals and their outcomes.

Chapter 4

Specifying Components

The first Spiral of the investigation addresses **RE1** - characterising and specifying components. A structured, machine-readable specification is key to automating parts of the selection process. As a basis for the specification template, background research was conducted on general resource description and data representation. Software and component specifications developed by others were also reviewed. Data modelling was approached using object-oriented techniques.

The key product of this Spiral is the specification template: SoftWare Verification Markup Language (swvML) is an XML schema which includes functional and non-functional attributes to describe software components. This same schema is used to describe both the ideal component (requirements) and each component in the repository of component data. This specification template adheres to XML Schema and Dublin Core[1] standards and has been integrated into the CdCE tools developed in this investigation. In later Spirals, the schema has been used to support automation of the selection process and to hold transformed software project data from a repository.

For the specification, there have been some key goals, which are given in Table 4.1. The specification template is a key part of the investigation's approach to the **quality** goal, which will form the basis of later strategies to support software selection. These strategies will utilise the template to allow automation via machine **intelligence**. Real world examples are considered in line with the **usability** goal. The template will include a balance of **reuse** and **innovation**, and allow for change and extension as required (**dynamics**).

The background and implementation of Spiral 1 follows. After the evaluation and

[1] http://dublincore.org/

SPIRAL 1	GOALS
Quality	Produce a specification template to facilitate quality CBSE development
Usability	Include real world needs in the development of the specification
Intelligence	Provide a specification that can facilitate intelligence and automation
Innovation	Consider novel approaches to the specification
Dynamics	Allow for change and the extension of the specification
Reuse	Where possible make use of existing code and artefacts

Table 4.1: Goals for Spiral 1

planning for the Spiral, Section 4.7 outlines the updates to the specification template that took place in subsequent Spirals.

4.1 Spiral 1 Overview

This Spiral focusses on **RE1**, developing or extending a template for the specification (characterisation) of components. The investigation extends into the closely related areas of COTS specification and the description of electronic resources. While in-house specifications are considered, they tend to utilise a white-box level of knowledge, and may not apply in the black-box context of components.

The swvML schema is intended to be comprehensive, making it applicable in various phases of component distribution and use. The fields included in the data model for the schema reflect the requirements of the stakeholders in each phase. Initially the needs of the component developers are considered - informing potential buyers of the features of their software. The XML instance document provides information through which the application developer can shortlist candidate components. A facility for searching through component documentation (XML instance documents) may be provided by a component broker, usually via a web-based repository. Existing XML schemas for components have been developed by brokers to assist in the automation of adding component information to their catalogues. The swvML schema aims to be used beyond component discovery - assisting with technical specification, testing and comparison for selection of components. It is also open to extension and reuse by application developers. Use cases for the expected applications of the schema and instance documents follow.

The use cases identified for the overall investigation are shown in Figure 4.1, those not greyed indicate the use cases related to Spiral 1. Component developers (**Register Component**) are interested in communicating the functionality of their software, along

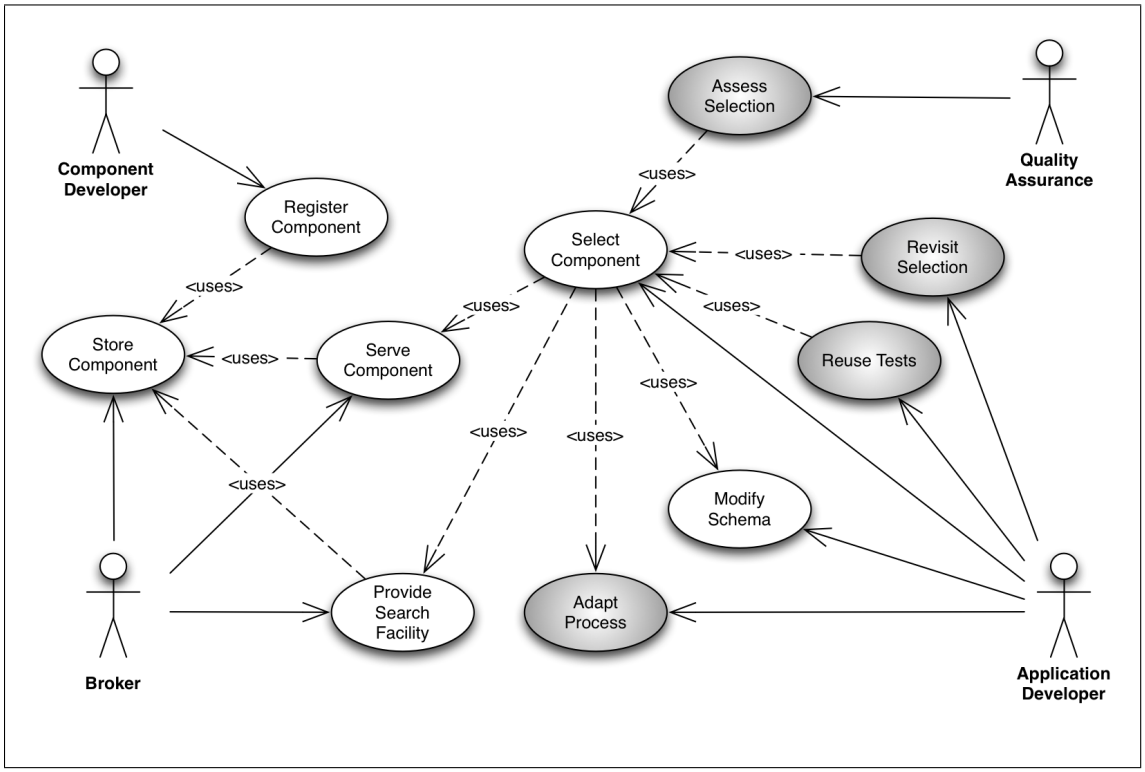


Figure 4.1: Use cases for the component specification schema (those not in the scope for this Spiral are greyed)

with languages used, compatibilities and dependencies. They also need to provide contact and administrative details, along with documentation. Application developers (**Select Component**) are also interested in these attributes, but will need an interface to allow for discovery of suitable software, and tools to assist with shortlisting and evaluating candidates. Component brokers (**Store Component**, **Serve Component** and **Provide Search**) require a stable schema for which to build tools, and perhaps allow automated conversion to an internal schema for their brokerage. Reuse is important for reducing effort for developers and can aid in standardisation, so a balance is required between generic and specific in the template implementation. The application developer may want to use all or part of the template to build new schemas (**Modify Schema**). This discussion shows that the specification has information of interest to all of the users, with supporting documentation as an aid to brokers and schema developers.

The stakeholders' interests in the specification of components varies based on their usage of the template. The identified stakeholders are included in Table 4.2. Application developers benefit from the standardisation of the information on components to aid the selection and integration process. They may also add to the template for their own

Stakeholder	Win Conditions
Application Developers	Specification includes all information required for decision making Adheres to standards Usability of template and tools (e.g. XSLT) Well documented and extensible
Component Developers	Clear documentation of how to use Includes items specific to their developed product
Component Brokers	Adheres to standards Objectivity of included information
Quality Assurance	Well documented Able to be validated
Academia	Adheres to standards Model is well designed Inclusions and omissions are justified Survey is representative Peer reviewed

Table 4.2: Win conditions for stakeholders (Spiral 1)

purposes. Developers of components may use a specification standard as a guide to documenting their products, rather than having to decide what to include case by case. Brokers may extend the template to include ratings and other evaluations done on site, as at Component Source[2].

When using the specification in industry, QA personnel need to have documentation of the specification to audit the conformance of data to the schema. From the academic perspective, the specification standard adds to the existing literature, as does the survey of existing specifications.

4.2 Spiral 1 Context

The background information utilised in the development of this specification template has come from four main areas:

- Resource description
- Data representation
- Software specification
- Component characterisation (functional and non-functional).

[2]http://www.componentsource.com/

According to the topic map introduced in Chapter1 (Figure 1.2), along with the goals for Spiral 1 (Table 4.1), it can be seen that areas used in this phase are central to the topic map, drawing on views from multiple disciplines. This Spiral captures and formulates a pathway of study using conclusions drawn from the review of literature (Chapter 2).

Resource description

A number of standards exist for providing information about electronic resources. These standards, along with existing schemata for software components, have been considered when developing the swvML schema. Although standards and implementation are usually closely tied, they can be considered independently.

Dublin Core is a standard developed to facilitate the discovery of electronic resources over the Internet. Development of Dublin Core began in 1995 and aimed for simplicity and to ensure an international consensus on its contents and structure. Many other standard templates are built upon Dublin Core, including AGLS[3], EdNA[4] and IMS[5] (see Table 4.3). Dublin Core fields referenced in these schemata are indicated by 'DC': fields with no equivalent are blank, otherwise the local schema name is given. In keeping with the idea of interoperability, the swvML schema uses many of the Dublin Core fields where their intention is similar to the requirements for component specification. This is mainly in the identification and contact sections of the component description.

Electronic resource description and metadata are most commonly held in XML documents. The intention when developing XML was to provide a class of data objects to support content such as: 'industry-specific markup, vendor-neutral data exchange, media-independent publishing, one-on-one marketing, workflow management in collaborative authoring environments, and the processing of Web documents by intelligent clients. It is also expected to find use in certain metadata applications.' (Cover, 2011). According to Hunter (2003), XML is the 'de facto standard for representing metadata descriptions of resources on the Internet'.

XML lends itself to sharing and reuse of templates, which can be utilised to build a schema as well as make it available for others to build upon. Investigation of XML Schema documentation led to examples of methods for implementing currency descriptions in

[3]Australian Government Locator Service http://www.agls.gov.au/
[4]Education Network Australia http://www.edna.edu.au/edna/go/resources/metadata/edna_metadata_profile
[5]IMS Global Learning Consortium http://www.imsproject.org/metadata/

Item	Dublin Core	AGLS	EdNA	IMS
Identification				
Name	title	DC	DC	DC
Date	subtypes	DC	DC	DC
Version				version
Language	language	DC	DC	language
Location	identifier	DC	DC	location
Description				
Function	description	DC	DC	DC
Detail				
Type	subject		categories	
Category				
Domain				
Audience		audience		
Contact				
Author	creator	DC	DC	DC
Vendor	publisher	DC	DC	DC
Support	contributor	DC	DC	agent
CorporateName	subtypes	DC	DC	DC

Table 4.3: Comparison of selected attributes in schema standards. Text in a column indicates the name used for that attribute in the standard. 'DC' is listed where the standard uses the Dublin Core name and format for an attribute.

XML, and for implementing type libraries (XML). Both of these approaches were adopted in the swvML schema, for the pricing information and for the structure of the various levels in the schema specification.

Data representation

There are a number of different categories of attributes in the specification. Some are ordinal, where the values have an inherent order and can be compared (e.g. price, date). Others are textual (e.g. company name) and do not lend themselves to manipulation or reasoning. When an attribute is text based and gives a long or short description, other text processing can be applied. This includes keyword searching and generation of ranking measures for relevance of the text to the given search terms.

Faceted classification of software was considered as a possible method for categorising the subject areas of individual components. The classification is developed on a number of categories, allowing for more useful information to assist application developers when searching for components (Prieto-Diaz, 1991). These categories can include the program

size and complexity, quality of documentation, programming language and ratings from users. Using faceted classification would have resulted in a single attribute encoding a number of component attributes. However, the complexity of creating a classification system, encoding and ordering was prohibitive. An alternative became available with the uptake and implementation of ontologies, making it possible to adopt existing classification schemes. The reuse of an existing classification scheme would align one of the goals of the project, and a survey of available knowledge bases is required to see if a suitable one is available.

Software specification

Recording specifications can be done with varying levels of formality. When an informal approach is used, ambiguity is more likely and it can be difficult to automate the understanding and processing of information. With a more formal approach, the specification will be less ambiguous but there may be issues with readability and learning of the notation used. As the aim of the functional specification is to allow for automated test generation, a formal approach is needed. Options for automated test generation from specifications include semi-formal UML diagrams (Yoon et al, 1999), Z notation (Stocks and Carrington, 1993), Vienna Development Method (VDM) (Dick and Faivre, 1993), and others. Each of these can be represented in XML, while literature exists for test generation from each. Z notation has advantages in the standardisation of the language, active development of tools for its use and prior familiarity of those involved in this project.

Component characterisation

In the time this project has been in progress, software repositories have become common to the point of being part of the general population's vocabulary (e.g. the Apple App-Store). Some of these specialise in components, and have characterised components with practical data models and XML, similar to the swvML schema. Another common aspect of describing components considers non-functional assessment or certification. Some in-house repositories use formal methods to describe components and aid searching and matching (Fidge, 2002, Rao and Sarma, 2003, Nakkrasae et al, 2004, Stylianou and Andreou, 2007). These were used by the developers (during development) and are available

for use in the repository. Although formal specification is used for components in this project, this is only for the description of the ideal component (functional requirements), the expected data for a repository entry is more basic: the context and interface metadata. Characterisation is also discussed in Section 2.2.1.

4.3 Spiral 1 Approach

As the initial Spiral of the development, Spiral 1 had no existing artefacts on which to build. The work to be done is based on literature, industry applications of similar type and preferences of the research program. At the time of the first Spiral, XML was gaining popularity as an interchange format for the description of documents and electronic resources. Dublin Core was becoming standardised and utilised widely, along with other standards.

The component specification template was derived from surveying existing work, industry usage (not just components), document standards and the research aims and delimitations. A compiled list of all component attributes was developed and consolidated. Each attribute was considered for inclusion. This was then modelled in UML (Figure 4.2) to abstract the concepts, identify gaps, and crystallise the expected usage. The resulting data model was compared with existing document templates to create a mapping between common attributes in order to add any that were missing. The UML was converted to XML schemas following conventions in the XML and document standards communities and concepts of modularity and reuse.

The motivation for developing a schema to describe components is to assist in the distribution and discovery of components available electronically. The tasks of finding components, and of making information about components available, is simplified by having a standard format for describing the component. This schema provides a structure for component specification, including contact information and technical details. By using a documentation standard, component developers will not need to modify their documentation for each brokerage site, and prospective component users will know what information they can expect to find when selecting candidate components. The added benefit is that some aspects of component discovery and selection can be automated with a machine-readable component description.

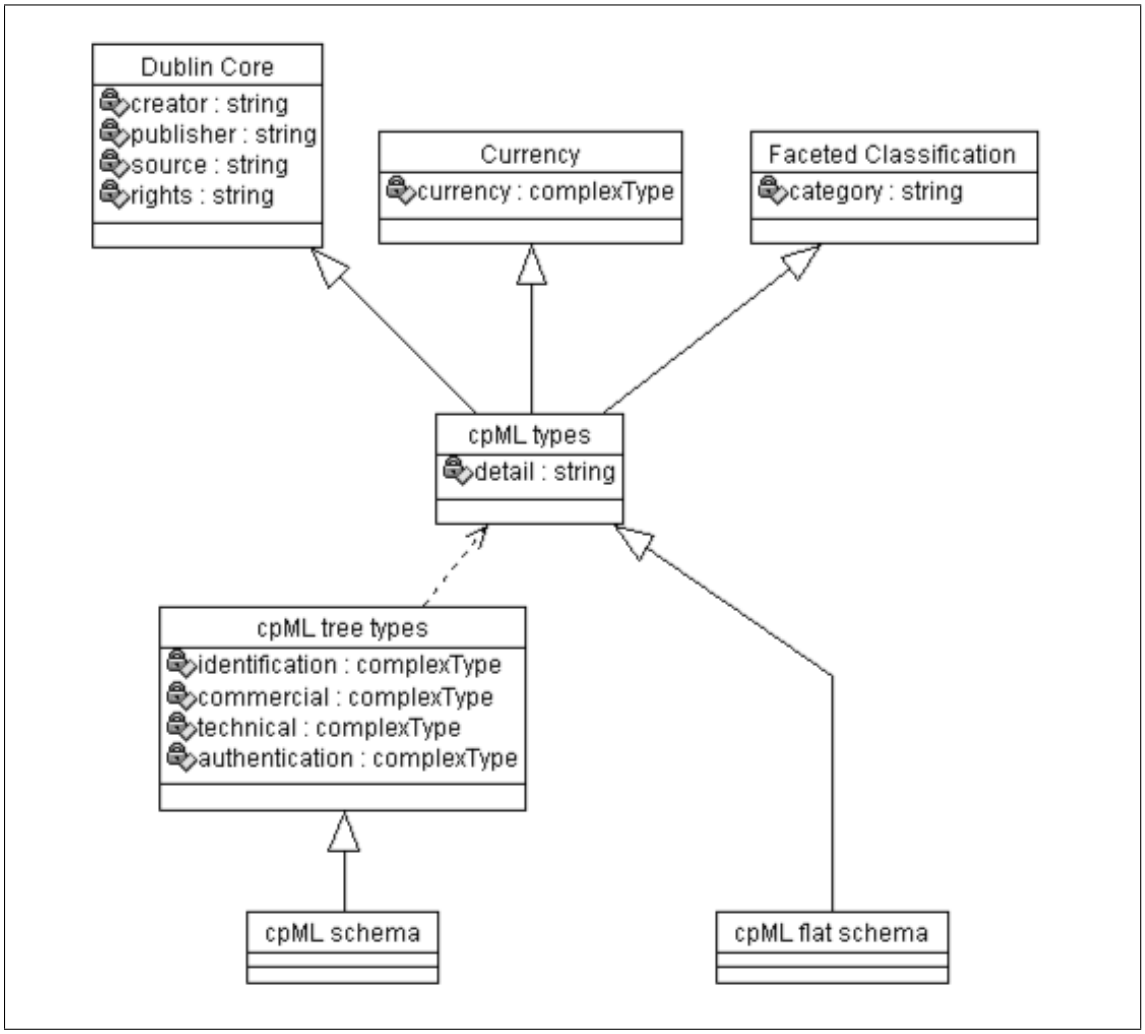


Figure 4.2: Original class diagram for the component specification schema

This schema has been developed to be put forward as a possible standard for component specifications. The approach has been to survey existing specifications and expected requirements to determine what should be in the specification. Object-oriented development techniques have been applied to assist in viewing the 'big picture' and the template is documented using W3C standards as a model.

Guidelines considered in the approach to the specification included: adhering to existing standards; matching attributes to user needs; including context and behaviour; and, supporting classification schemes (implemented in Spiral 4). These address the overall goals of reuse, usability, intelligence and dynamics.

Adhering to Existing Standards

The schema is designed to make use of existing standards for electronic resource documentation where possible. This adherence to standards makes it possible to seamlessly include information specific to software components, while maintaining a standard (e.g. Dublin Core) across the organisation's information stores.

Once a list had been compiled from component broker sites and papers, related attributes were grouped into six categories: identification, description, contact, commercial, technical and authentication. Because components can be viewed as electronic resources, the specification extends Dublin Core. Twelve of the attributes refer to Dublin Core. The specification is detailed in Tables 4.4 to 4.5.

Matching Attributes to User Needs

Use cases were developed for the specification to identify the many ways that the template may be used (Figure 4.1). It was important to make the attributes comprehensive, providing a component with an accompanying document that includes information needed for various stages of its life.

Object-oriented techniques were used to model the data, with an awareness that the implementation may not be object-oriented. Use case diagrams provide a model from which to consider the perspective for the various users (actors). The original class diagram for the component specification is in Figure 4.2. This was the data model as at the end of Spiral 1.

Including Context and Behaviour

The context of the component refers to the environment in which it operates, with development context unlikely to be the same as the target context. This information is held in the technical section of the schema, along with the behavioural specification. Context is highly important when assessing the suitability of a component and the amount of coding required for adaptation into a new context. Context comes into play at two main points of the selection process: when shortlisting the candidate components, and when testing and evaluating the candidates. In Spiral 1 we need to consider both of these, with the details for testing and evaluation to come in later Spirals.

The target context for a component includes the operating system, framework, related components and development language. These are included in the `technical` section of the template. Additional non-functional information may include certification and authentication. The support for behaviour is through attributes for `interface` information, again within the `technical` part of the template. This allows each interface to be recorded, including inputs, outputs and errors.

4.4 Spiral 1 Implementation

Tables 4.4 to 4.5 list the fields and contents for the swvML data model. The data model includes fields to meet the basic requirements of Dublin Core.

Item	**Description**	**Type**	**Schemes**	**Min Occurs**	**Max Occurs**
Identification		complex			
Title	name of the component	xs:string		1	1
Date	date released	xs:string	ISO8601	1	1
Version	current version number	xs:string		1	1
Language	component language (en)	xs:string	RFC1766	0	1
Source	URI to access component	xs:string	URI	1	1
Description		complex			
Description	function of the component	xs:string		1	1
Detail	detailed description of fn	xs:string		0	1
Type		LIST		0	*
Category		LIST		0	*
Domain		LIST		0	*
Audience	intended market	xs:string		0	1
Contact		complex			
Creator	developers of component	xs:string		1	*
Publisher	distributors of component	xs:string		1	*
Support	support providers	xs:string		0	*
CorporateName	full company name	xs:string		1	1
Address	for each entity	xs:string		1	*
Street	street address	xs:string		1	1
City	city	xs:string		1	1
State	state	xs:string		1	1
PostCode	postcode	xs:string		1	1
Email	email address	xs:string	URI	0	1
Fax	fax number	xs:string		0	1
Phone	telephone number	xs:string		0	1
URL	URL for org. web site	xs:string	URI	0	1

Table 4.4: Attributes and Types in swvML Schema (Part 1/2)

Item	Description	Type	Schemes	Min Occurs	Max Occurs
Commercial		complex		0	1
Price	price and currency type	xs:string		0	*
Licence	licence covered by price	xs:string		0	*
Demos	URL of demo s/w	xs:string	URI	0	1
Support	level of support provided	xs:string		0	1
Legals	URL for copyright info	xs:string	URI	0	1
Documentation	URL for documentation	xs:string	URI	0	1
SourceCode	is source code available	xs:string		0	*
SourcePrice	price for source code	xs:string		0	*
Technical		complex			
TechDescription	technical description	xs:string		0	1
DevLanguage	development language	xs:string		0	1
Software Framework	software framework	xs:string		0	1
Operating System	operating system	xs:string		0	1
Hardware		xs:string	IMT	0	1
Platform	h/w platform	xs:string		0	*
Processor	processor requirements	xs:string		0	*
RAM	RAM requirements	xs:string		0	1
DiskSpace	disk space requirements	xs:string		0	1
Links		xs:string		0	1
Containers		xs:string		0	*
Compatibility		xs:string		0	*
Relation	Related components	xs:string		0	*
Extensibility		xs:string		0	*
Reference	3rd party reference			0	1
ID reference	ID	xs:string		0	1
Organisation	organisation holding refs	xs:string		0	1
Authentication	authentication information	complex		0	1
Testing	testing info	xs:string		0	*
Level	level attained	xs:string		0	1
Organisation	testing organisation	xs:string		0	1
Certification	certification information	xs:string		0	*
Level	level attained	xs:string		0	1
Organisation	certification organisation	xs:string		0	1
Development Status	status of component (beta)	enum		0	1

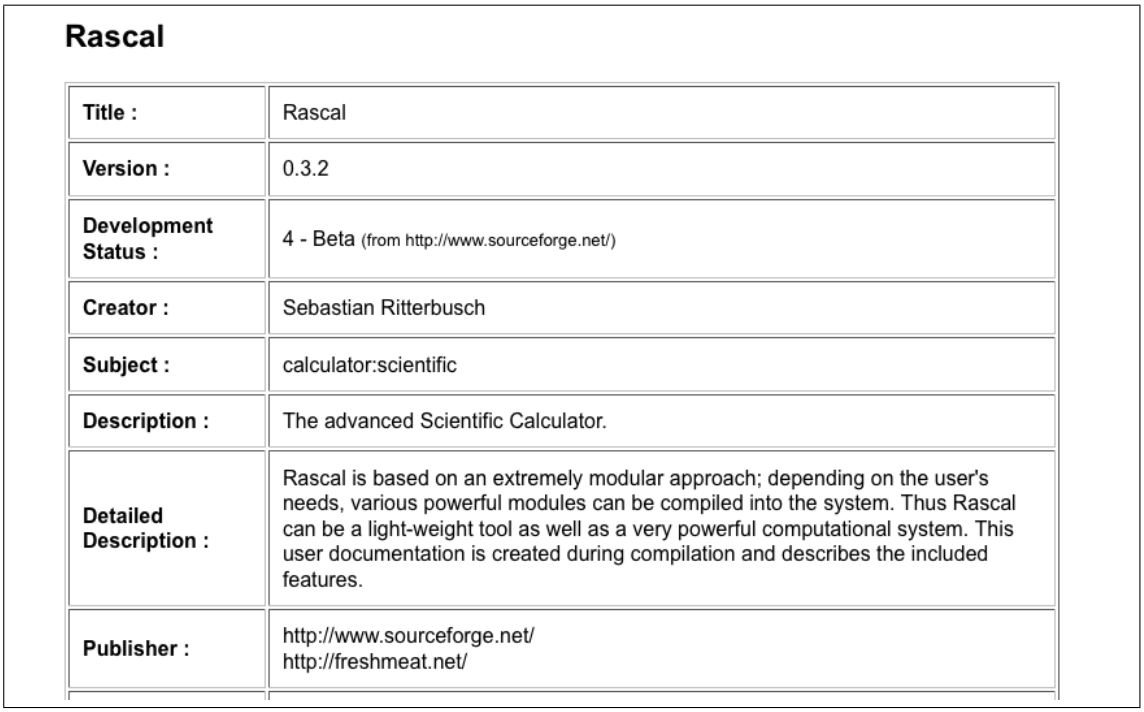
Table 4.5: Attributes and Types in swvML Schema (Part 2/2)

```
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="swvML_ap.xsl"?>
<Description
   xmlns="http://eng.murdoch.edu.au/~valerie/swvMLap/1.0/"
   xmlns:dc="http://purl.org/dc/elements/1.0/"
   xmlns:swv="http://eng.murdoch.edu.au/~valerie/swvML/1.0/"
   about="http://eng.murdoch.edu.au/~valerie/swvML/1.0/">
      <dc:title>Title of software</dc:title>
      <swv:version>version</swv:version>
      <swv:devStatus>beta</swv:devStatus>
      <dc:creator>creator</dc:creator>
      <dc:subject subjectScheme="SFC">subject categories</dc:subject>
      <dc:description>description</dc:description>
      <swv:detail>detailed description</swv:detail>
      <dc:publisher>publisher/vendor</dc:publisher>
      <swv:support>support contact</swv:support>
      <dc:date>date</dc:date>
      <dc:type>type</dc:type>
      <dc:format>format</dc:format>
      <dc:identifier identifierScheme="URI">http://eng.murdoch.edu.au/~valerie/swvMLap/1.0/
      </dc:identifier>
      <dc:source>source</dc:source>
      <dc:language>en</dc:language>
      <dc:relation>relation</dc:relation>
      <dc:rights>http://eng.murdoch.edu.au/~valerie/swvMLap/copyright.html</dc:rights>
      <swv:licence>freeware</swv:licence>
      <swv:demo>source</swv:demo>
      <swv:documentation>http://eng.murdoch.edu.au/~valerie/swvMLap/doc.html
      </swv:documentation>
      <swv:sourceCode>available</swv:sourceCode>
      <swv:technical>
       <swv:techDescription>This is the place to enter a more detailed and technical
       description of the software component.  It could contain information about
       algorithms used, data structures and anything else that may help the programmer.
       </swv:techDescription>
       <swv:devLanguage>devLanguage</swv:devLanguage>
       <swv:framework>framework</swv:framework>
       <swv:standard>standard</swv:standard>
       <swv:operatingSystem>operatingSystem</swv:operatingSystem>
      <swv:systemRequirements>
         <swv:platform>platform</swv:platform>
         <swv:processor>processor</swv:processor>
         <swv:memory>memory</swv:memory>
         <swv:diskSpace>4KB</swv:diskSpace>
       </swv:systemRequirements>
       <swv:interface>
         <swv:name>name</swv:name>
         <swv:function>function</swv:function>
         <swv:input>input</swv:input>
         <swv:output>output</swv:output>
         <swv:error>
         <swv:errorName>errorName</swv:errorName>
         <swv:errorDescription>errorDescription</swv:errorDescription>
            </swv:error>
         <swv:error>
         <swv:errorName>errorName2</swv:errorName>
         <swv:errorDescription>errorDescription2</swv:errorDescription>
            </swv:error>
        </swv:interface>
  </swv:technical>
   </Description>
</xml>
```

Figure 4.3: Example of the swvML v1.0 template

Other fields use type schemas from existing schema standards/conventions. The data modelling used class diagrams, which were matched in the multi-level XML implementation. The class diagram in Figure 4.2 shows the relationship between the different levels of the schema. For maximal reuse there are type schemata to define the types to be reused, and may later be replaced by industry standards (e.g. currency). From there, the complex data types are built to be included in the tree version of the swvML schema. Alternatively, a flat schema could draw its data types directly from the type files.

To extend the schema, it can be referenced and an application schema drawn from it. Prior to extension, a data modelling activity should take place to find the best fit for new attributes. In some cases there may be an existing standard for a type of field (e.g. ISO 8601 for Date), and they should be used where possible. In addition, the XSLT stylesheet should be extended to include the new fields.



Rascal

Title :	Rascal
Version :	0.3.2
Development Status :	4 - Beta (from http://www.sourceforge.net/)
Creator :	Sebastian Ritterbusch
Subject :	calculator:scientific
Description :	The advanced Scientific Calculator.
Detailed Description :	Rascal is based on an extremely modular approach; depending on the user's needs, various powerful modules can be compiled into the system. Thus Rascal can be a light-weight tool as well as a very powerful computational system. This user documentation is created during compilation and describes the included features.
Publisher :	http://www.sourceforge.net/ http://freshmeat.net/

Figure 4.4: Example of XML file for Rascal software, rendered using the XSLT stylesheet for greater readability

Some attributes are required and can only appear once, indicated by **Min Occurs** and **Max Occurs** of 1. Where an attribute is not required, **Min Occurs** is equal to 0. Attributes which can have multiple appearances in the XML document are indicated with a * for **Max Occurs**.

Figure 4.3 is a sample specification using the Spiral 1 schema (swvML v1.0). The indenting of attributes indicates the respective levels within the data model, with most elements appearing as sub-elements of a main element.

XSLT stylesheets can be used to transform XML in a variety of ways. For swvML, a stylesheet takes the raw XML file and renders it to make it a more readable HTML page (Figure 4.4). Similar stylesheets are available for most XML files created for this project.

4.5 Spiral 1 Evaluation

The focus of Spiral 1 was the development of a specification template, approached through a survey of literature and practice, which informed the data model. This was then used as the basis for the swvML schema for characterising components - the main contribution of Spiral 1 (C1).

The evaluation for this Spiral is based on the Spiral goals (Table 4.1) and the stakeholder Win conditions (Table 4.2). Many of the Win conditions relate to adherence to standards, which includes the use of XML and alignment to Dublin Core. The discussion has highlighted the survey of existing schemas that was used to ensure the inclusion of attributes relevant to all stakeholders, and the justification for each. Included attributes are objective and where a subjective attribute is included, it must show its source (e.g. user ratings from a particular repository). Documentation for the specification is included in this Chapter, including the data model as a class diagram and a discussion of how the model can be extended. XML editors and validators were used to ensure that the XML Schema adhered to these standards and that the related XML documents would be compatible with tools for handling XML. The final stakeholder win condition is the peer review of the work, which has been accomplished through faculty presentations, presenting the work in post-graduate symposia and submission to the Young Researchers Workshop at the International Conference of Software Reuse (ICSR) (Maxville, 2002).

SPIRAL 1	**Purpose** **Issue** **Object** **Context**	**Evaluate** **effectiveness of** **the developed component specification** **Spiral 1**	**Result**
Goal 1A	Focus	Quality: *Produce a specification template to facilitate quality CBSE development*	
	Viewpoint	Quality Assurance personnel	
	Q1A1	Does the specification align to and apply standards?	YES
	Q1A2	Is the specification well documented?	YES
Goal 1B	Focus	Usability: *Include real world needs in the development of the specification*	
	Viewpoint	Application developer	
	Q1B1	Does the specification include attributes from repositories?	YES
	Q1B2	Does the specification include the attributes required for assessing components?	YES
	Q1B3	Has the work been tested on real-world examples?	YES
Goal 1C	Focus	Intelligence: *Provide a specification that can facilitate intelligence and automation*	
	Viewpoint	Application developer	
	Q1C1	Is the specification machine readable and structured?	YES

Table 4.6: GQM Summary - Spiral 1 (Part 1/2)

Spiral Goals

The responses to the Questions on the Spiral goals indicates that the Spiral has been successful in its aims. In the case of the question on innovation, the response was the goal had been partially addressed, which is indicative of the small scope for innovation in the specification.

The following discussion refers to the goals and questions summarised in Tables 4.6 and 4.7. Each goal is included as a heading followed by the discussion relating to the questions within that goal.

Quality

The specification was developed using XML schema and has been validated to check that it conforms to the standard. The template also adheres to the Dublin Core standard for describing electronic resources. The specification has a separate design document, including use cases, class diagrams for the data model and a full description of fields.

SPIRAL 1	**Purpose** **Issue** **Object** **Context**	**Evaluate** **effectiveness of** **the developed component specification** **Spiral 1**	**Results**
Goal 1D	Focus	Innovation: *Consider novel approaches to the specification*	
	Viewpoint	Academia	
	Q1D1	Have innovations been developed?	PART
Goal 1E	Focus	Dynamics: *Allow for change and the extension of the specification*	
	Viewpoint	Application developer	
	Q1E1	Is the specification extensible?	YES
Goal 1F	Focus	Reuse: *Where possible make use of existing code and artefacts*	
	Viewpoint	Application developer	
	Q1F1	Has the work reused external resources?	YES

Table 4.7: GQM Summary - Spiral 1 (Part 2/2)

Usability

Component and software repositories were reviewed to ensure that developer needs were represented. The specification includes all attributes that are commonly available for searching repositories. Assessment will require more information than the basic search query on a repository. In some selection/evaluation processes, the evaluation is carried out using customised criteria, often through the use of GQM (Basili et al, 1994). For this study, the schema will be fixed, with the behavioural specification recorded within the ideal specification to hold the more specific information. Subsequent Spirals demonstrate the use of the specification applied to the assessment of software and the attributes were considered appropriate.

The specification was applied to a number of examples to assess its applicability and to test the schema.

Intelligence

XML was selected as the format for the project based on its ability to structure textual data in a machine-readable format. This is critical for the automation of the selection process.

Innovation

Although there is not a lot of scope for innovation, the specification brings together academic and industry characterisations. The use of the Z specification is novel in the area of component characterisation. The formal specification focusses on interfaces and required behaviour, which is only required in the ideal specification - not in candidates.

Dynamics

The XML Schema is easily extended by adding fields, which would retain compatibility with swvML. Changes to the fields would need to be reflected in their handling in any tools that are created. The data model can be used to ensure that changes fit with the current schema in terms of where the attributes are grouped to make the documents easier to read.

Reuse

At the field level, Dublin Core and W3C standards were used. The XML document and schema is reuse in itself. Also third party XML validators and editors were applied.

4.6 Spiral 1 Review and Plan

The work has performed well with respect to stakeholder Win conditions and evaluation against Spiral goals. The problem description was to develop a structured, machine-readable specification for components. This has been achieved. As well as international conferences, a further (peer) review took place through participation in a post-graduate symposium.

The commitment is that a selection process could now be built on the foundation provided by the specification. The specification is geared towards automation and application of selection strategies in the course of this investigation.

Planning for Spiral 2 required a decision about what aspect of the 'process' to focus on. Figure 4.5 summarises the options that were considered. To explain the direction taken in terms of the diagram, the horizontal bars at the bottom of the diagram indicate the type of matching that can take place. The focus for shortlisting is on the desired component (ideal) specification, which could be compared to the published information

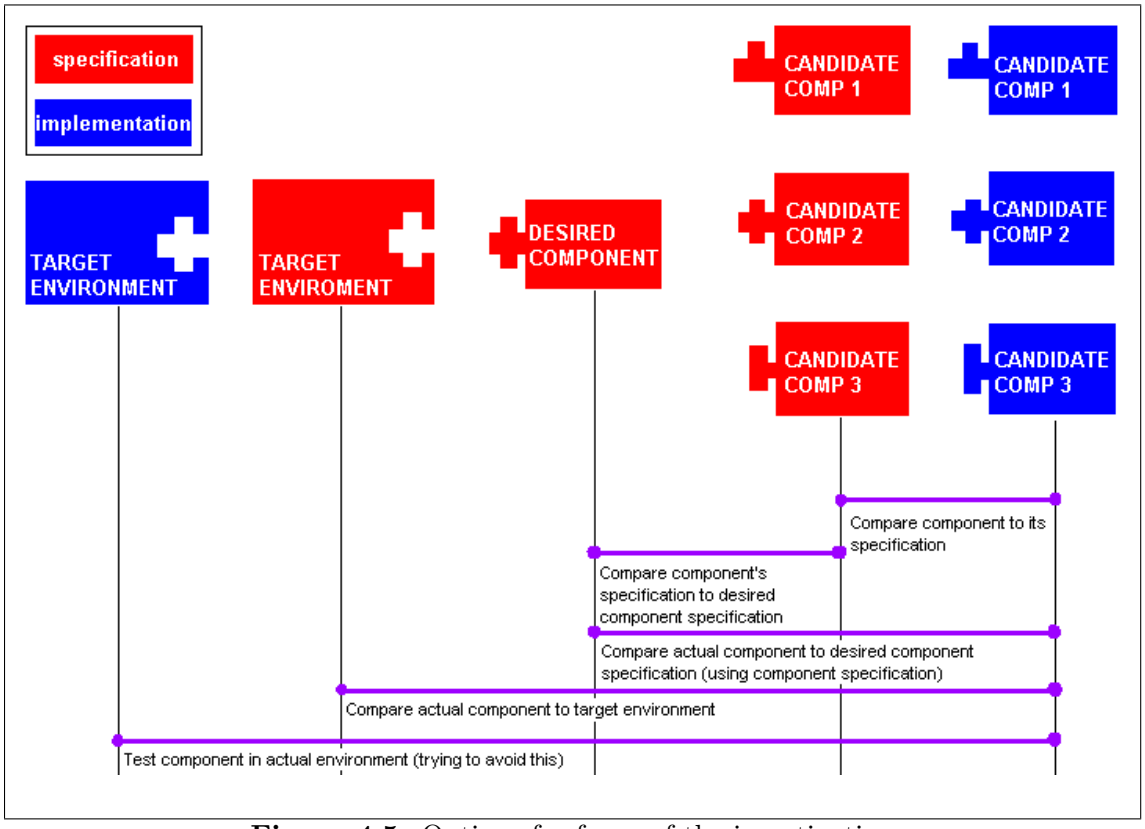


Figure 4.5: Options for focus of the investigation

available for components in repositories ('candidates', shown in red). Given a shortlist of candidates, the preferred approach is to evaluate the performance of the executables for each candidate with respect to the ideal component specification.

Factors behind the decision include reducing risk, and increasing interest and the perceived value of outcomes (researcher's perspective). It was at this point the focus changed from the planned test generation investigation and into the CdCE Process[6]. Spiral 2 began with a commitment to develop or reuse a process, working with the specification from Spiral 1. Strategies would be attached to it in subsequent Spirals.

4.7 Post-Spiral Update

Although Spiral 1 was dedicated to developing a specification template (**RE1**), it is not the only Spiral to impact on the specification. Dashed outline boxes on Figure 4.6 indicates the updates to the specification subsequent to Spiral 1.

In Spiral 3, the swvML specification was used in the generation of training data for the

[6]Previously called the Context-driven Component Testing (CdCT) Process

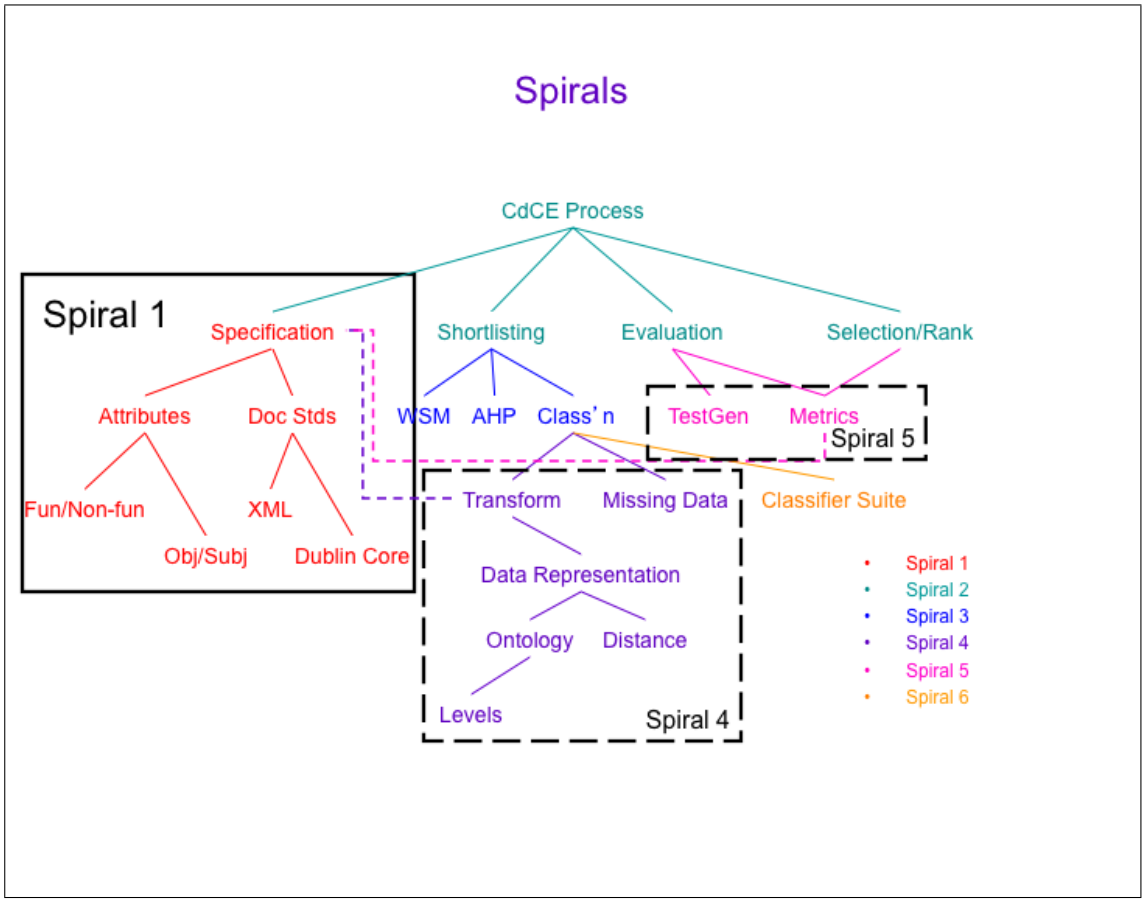


Figure 4.6: Spiral 1 and later Spiral updates to specification

machine learning classifier (C4.5). Through this, a flat schema (rather than hierarchical) was adopted as the default schema to simplify and scale processing of the XML files.

As a result of the review of Spiral 3, Spiral 4 attached five data types to the attributes to improve data representation. Prior to this there had been four data types, which were superceded. Further experimentation in Spiral 4 added ontologies and distance matrices to provide a knowledge base to support the schema.

Additional changes were made to the schema in Spiral 5 to accommodate the evaluation and ranking strategies. Specifically, the Z specification within the `techDescription` element was fully defined and attributes were added for the evaluation metrics. A summary of these updates follows.

Hierarchical to Flat Schema

Experience with processing data from a large repository (over 41,000 items) in Spiral 3 indicated that the flat version of the schema was more appropriate for the case studies.

The developed programs needed to parse the XML document to access the content. This may be done as a stream of information, or the entire file can be read into a Document Object Model (DOM). For large files this can cause memory problems. Also, in this case, the large XML of repository metadata is actually representing 41,000 smaller items (components) which is the level at which the tools need to work. The flat structure allowed the use of a SAX parser[7] and filters instead of loading the entire document tree as a DOM. For this to be possible, all attribute names have to be unique and not rely on the position in the tree to differentiate between duplicates. This also aligns to the Dublin Core approach which prefers flat schemas so they can be read without too much knowledge of the XML document structure.

Even in a flat schema, the abstract data model is the same, with any changes from the original implemented through extensions and enhancements. The final version of the template is described in Section 4.7.1 of this Chapter. The changes after Spiral 1 relate to the use of ontologies instead of faceted classification and the inclusion of the Z specification. Some renaming of fields also took place. Future requirements were also considered, for example there are attributes related to the certification and testing of software components. So, although the related data was not available for this investigation, certification and authentication are attributes in the specification.

Formal Specification

In Spiral 1 interfaces and static evaluation of components were considered sufficient for the project. As the CdCE Process developed (Spiral 2), it was decided to use dynamic evaluation (testing) and the specification was changed to include behavioural information. This change was in the ideal specification for requirements, while interface information could still be included as before. The behavioural specification was used to manually generate tests until Spiral 5, where some specifics of the representation were clarified and a test generation tool developed. The intent had been to explore the use of the behavioural specification to provide more complex test generation (based on a formal specification) and test oracle functionality. This is regarded as future work and the test generation is currently based on interfaces and equivalence classes.

Development of the CdCE Process clarified the requirements for the technical and

[7]Simple API for XML allows the XML files to be treated as a series of events (tags/attributes) and the developer is free to interpret them

behavioural specification of components, resulting in the inclusion of a Z specification in the model. The `techDescription` field is used to store the Z specification. UML, interface and other specification information could also be held under `techDescription`. Spiral 5 moved the focus to the behavioural specification and resulted in clarification of how the Z specification was stored and processed.

In the schema template, the definitions of the contents of the `techDescription` have changed. While the tags are not affected by the change, the internal content is. The `techDescription` after Spiral 5 includes the Z specification, coded in the LaTeX standard. This provides interface, partition and context information for the test generation.

Supporting Ontologies and Classification Schemes

The evaluation and reflection after Spiral 3 indicates that results could be improved if the data representation was enhanced. It was clear that more could be extracted from the data if classifications or ontologies were used for the terminology included in each attribute. Numeric and date values would also benefit from more tailored treatment. Spiral 4 introduced ontologies to the dataset along with transforming software to regulate the incoming data.

Although this had more impact on the tools developed for processing the data, it required a tightening of the definitions of the attributes and the values they could hold. Much of the ontology is based on the freshmeat Open-Source repository and related trove categories.

Metrics

Spiral 5 focussed on the details of the evaluation. For this, nine new attributes were added to the schema for the evaluation metrics. All of them are numeric, with values ranging between 0 and 10. The value in the ideal specification will indicate the optimal value, and preferred range (e.g. optimal 10, range 7-10).

The candidates are evaluated in Steps 4 - 6, populating the metrics of interest. The application developer indicates their required values through the ideal specification and these are used to rank the candidates in Step 7 of the CdCE Process.

4.7.1 Updated Schema Implementation

The following discussion of the implementation refers to the final version of the specification template. Tables 4.8 to 4.10 list the fields and contents for the final swvML schema.

The table indicates groupings of attributes to indicate type. Unlike the initial version, the groupings do not correspond to complex attributes (containing sub-attributes).

While this flat version of the schema is in keeping with the Dublin Core style of schema development, the choice of flat or hierarchical schema when developing instance documents is left to the preferences of each organisation. Software reading schema data directly will need to address the two versions of the schema, or may pre-process the data through XSLT.

As part of the enhancements in Spiral 4, types of attributes in the specification have been expanded, and tools were provided along with a knowledge base for ontology attributes.

Item	Description	Type	Schemes	Min Occurs	Max Occurs
Identification					
title	name of the component	xs:string	DC	1	1
version	current version number	xs:string		1	1
date	date released	xs:string	DC & ISO8601	1	1
language	component language (en)	xs:string	DC & RFC1766	0	1
publisher	publishers of component	address	DC	1	*
identifier	publisher identifier	xs:string	DC	1	1
source	URI to access component	xs:string	URI	1	*
Description					
description	function of the component	xs:string	DC	1	1
detail	more detailed description	xs:string		0	1
type		xs:string	DC	0	*
format		xs:string	DC	0	*
subject	classification/category	xs:string		0	*
Commercial				0	1
price	price and currency type	xs:string		0	*
licence	licence covered by price	xs:string		0	*
rights	URL for copyright info	xs:string	DC;URI	0	1
demos	URL of demo s/w	xs:string	URI	0	1
supportLevel	level of support provided	xs:string		0	1
documentation	URL for documentation	xs:string	URI	0	1
sourceCode	source code access info	xs:string		0	*

Table 4.8: Attributes and types in swvML schema (Part 1/3)

Item	Description	Type	Schemes	Min Occurs	Max Occurs
Technical					
technical	technical description	xs:string		0	1
devStatus	development status	enum		0	1
devLanguage	development language	xs:string		0	1
operatingSystem	operating system	xs:string		0	1
framework	software framework	xs:string		0	1
standard	standards adhered to	xs:string		0	*
platform	h/w platform	xs:string		0	*
processor	processor requirements	xs:string		0	*
memory	RAM requirements	xs:string		0	1
diskSpace	disk space requirements	xs:string		0	1
relation	related software	xs:string	DC	0	*
rel_id		xs:string		0	*
rel_type		xs:string		0	*
rel_source		xs:string		0	*
rel_version		xs:string		0	*
rel_value		xs:string		0	*
Metrics					
FFIT	Functional fit	xs:string		0	1
FEXS	Functional excess	xs:string		0	1
AEFT	Adaptation effort	xs:string		0	1
TFIT	Testing fit	xs:string		0	1
TRES	Test result	xs:string		0	1
CX_P	Performance testing	xs:string		0	1
CX_R	Reliability testing	xs:string		0	1
CX_S	Stress testing	xs:string		0	1
CX_U	Usage testing	xs:string		0	1

Table 4.9: Attributes and types in updated swvML schema (Part 2/3), new/changed items in **bold**

Item	Description	Type	Schemes	Min Occurs	Max Occurs
Contact					
Creator	developers of component	address	DC	1	*
Support	support providers	address		0	*
Address				0	1
Street	street address	xs:string		1	1
City	city	xs:string		1	1
State	state	xs:string		1	1
PostCode	postcode	xs:string		1	1
Email	email address	xs:string	URI	0	1
Fax	fax number	xs:string		0	1
Phone	telephone number	xs:string		0	1
URL	URL for org. web site	xs:string	URI	0	1
Certification				0	1
testing	testing info	xs:string		0	*
t_level	level attained	xs:string		0	1
t_organisation	testing organisation	xs:string		0	1
certification	certification information	xs:string		0	*
c_level	level attained	xs:string		0	1
c_organisation	certification organisation	xs:string		0	1

Table 4.10: Attributes and types in swvML schema (Part 3/3)

4.8 Summary

This Chapter has described the investigation of **RE1**: the development of a component specification template, along with some of the rationale for the decisions that were made in Spiral 1 of the investigation. To position the schema, use cases were developed and the stakeholders perspective became a key influence on the type of data required. Since components are viewed as electronic resources, the specification aligns with a widely used standard, Dublin Core. The schema is implemented as an XML Schema, and provides a predominately flat XML schema, which adheres to Dublin Core policy and simplifies the processing of very large files.

This schema is expected to evolve, although care has been taken to include information felt to be important to all stakeholders in the component based software development community. Extending the schema is simple through XML and schema developers are encouraged to help identify fields that may be brought into the schema standard in the future.

The evaluation of the specification indicates that it has satisfied the stakeholder requirements. The specification is central to the selection process which is developed and explored in the following Spirals. As such, it has evolved, as described in Section 4.7. The contributions discussed in this Chapter are the development of the specification template (C1) and the inclusion of context in the specification (C3), both of which show their value in later Spirals.

The following Chapter explores **RE2**, the development of the process for component selection. Spiral 2 builds on the specification from Spiral 1 and creates a process to provide scope for automation and intelligence.

Chapter 5

The CdCE Process

This Chapter describes the investigations of Spiral 2 and the development of a process for software selection. As noted in Chapter 2, while other selection processes exist in the literature, they did not match the needs of this investigation. For this project, the requirement was to have a process amenable to automation, as discussed in Chapter 1.

RE2 involves the development of a component selection process. The desired qualities for the process are that it be intuitive, iterative, repeatable and suited to automation. Addressing one of the issues with component and COTS selection, the CdCE Process aims to be structured and repeatable. To allow scaling of the process to larger numbers of applications, tool support and automation are imperative. Central to the CdCE assessment is testing of the software to be more certain its behaviour matches the specification.

The goals for Spiral 2 are listed in Table 5.1. The aim of this Spiral was the development of a structured process for selecting and evaluating components. An eight-step process was developed to support context-driven component evaluation. The Process was trialled manually on a software selection task. Each Step within the Process is defined by its inputs and outputs, and the task required. This allows flexibility for exploring

SPIRAL 2	GOALS
Quality	Produce a structured, repeatable process for selecting and evaluating components
Usability	Consider existing processes and organisational requirements, provide tools and automation
Intelligence	Identify areas where machine intelligence and automation can be applied
Innovation	Consider novel approaches to the process
Dynamics	Allow for dynamic assessment with context
Reuse	Where possible make use of existing code and artefacts

Table 5.1: Goals for Spiral 2

strategies by changing the strategies and implementation independently of the rest of the Process.

Following the review and evaluation of this Spiral of work, a summary of the post-Spiral updates is given. The CdCE Process is the basis for work in later Spirals and has been successfully used in a number of case studies. In each Spiral, the focus was on one or more Steps of the Process, providing an instantiation of the Step through tools and more detailed procedures.

As a product developed across this investigation, the CdCE Process is repeatable and traceable, with tool support and the application of machine learning techniques to assist the user. The complete process is contribution C2 of this research. The level of automation provided helps the selection process scale to a larger number of candidates and enhances the information available to the application developer. A focus for the Process has been to allow for the target context to be considered throughout the selection process, which supports contribution C3. This Spiral flags the support for context, which is actioned in later Spirals. An additional contribution (C8) is that the Process can also be used as a pattern for software selection, allowing for alternate instantiations to suit the problem at hand.

5.1 Spiral 2 Overview

As preparation for Spiral 2, the actors involved in the selection process are considered. The requirements for the CdCE Process relate most specifically to application developers and quality assurance officers. The use cases are shown in Figure 5.1. The Process targets three of the use cases: Select Component; Assess Selection and Adapt Process. Application developers are interested in the entire process, as well as the option of using part of it, or substituting in a local technique, such as a different formalisation of behaviour for test generation. Quality assurance assessments can benefit from the artefacts of the process. Traceability of decision-making is a key value-add of the Process. Beyond the use cases, we can consider the application developer revisiting the selection task as the system evolves. The artefacts from the process can be used in maintenance tasks, such as reusing tests or adaptation models.

The main issues for stakeholders are the usefulness of the Process, how it affects

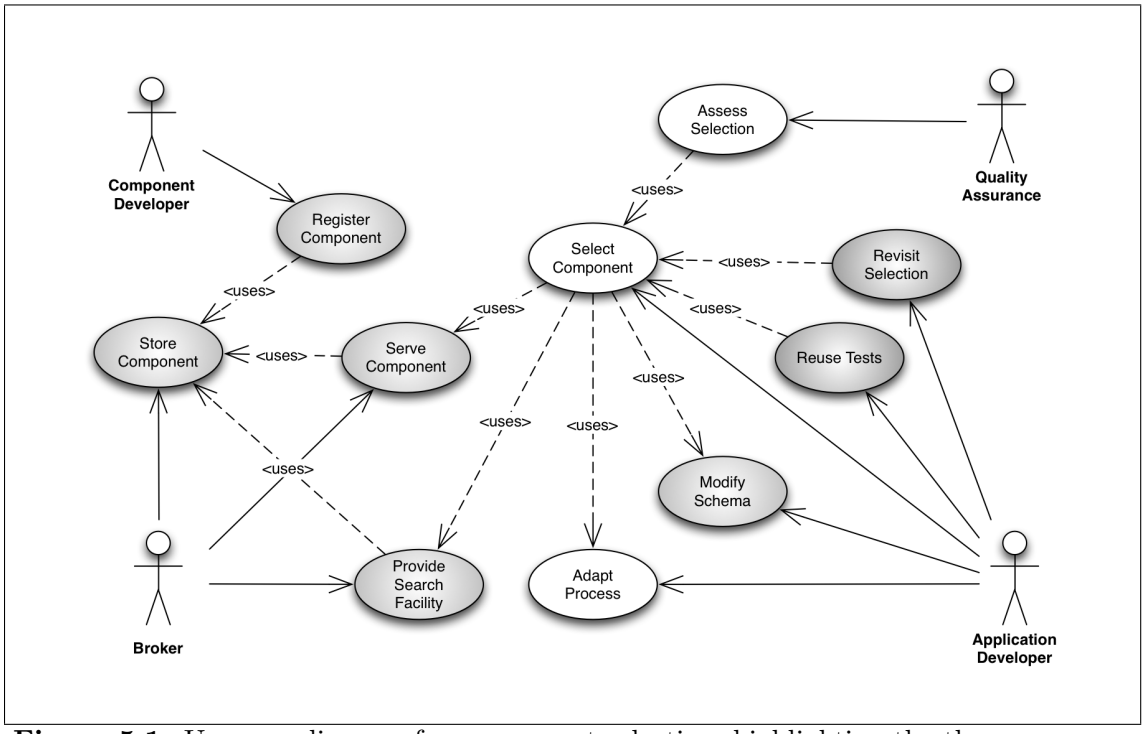


Figure 5.1: Use case diagram for component selection, highlighting the three use cases focussed on in Spiral 2

their operations and the academic merit. Application developers need to know that the Process will be of benefit to them over no Process, in that it adds value (e.g. easier, wider selection and better results). Component developers are interested in how the assessment is done, and how this affects the way they specify their components. Brokers want to know how the process will fit in with their repositories as well as what information will be needed and when. Quality assurance personnel will need the process to be well-structured and documented, with all decision-making recorded and justified. The requirements for academia concern the validity of results, breadth of the survey and some peer review of the work. The stakeholder evaluation criteria are provided in Table 5.2.

5.2 Spiral 2 Context

This chapter outlines the CdCE Process and how it was developed. The literature and existing processes for component and COTS selection are now considered. Details on shortlisting and testing within the CdCE Process and their justification are in subsequent chapters.

Stakeholder	Win Conditions
Application Developers	Intuitive process. Justifiable results. Low overhead to use process.
Component Developers	Know how their component is assessed and compared.
Component Brokers	How to integrate with the repository. What information is required and when.
Quality Assurance	Documentation provided for decisions.
Academia	Inclusions and omissions are justified. Survey is representative. Peer reviewed. Intuitive, iterative, repeatable and suited to automation.

Table 5.2: Win conditions for stakeholders (Spiral 2)

An early decision spanning Spirals 1 and 2 was the type of criteria used in the evaluation. At the time the work was done, it was common to choose functional criteria, with a more recent trend towards supporting non-functional criteria (Beus-Dukic, 2000). Authors including Weyuker (1998) and Shaw and Garlan (1996) indicated the importance of context in testing and in integration of software components. With this in mind, context was included as a key aspect of the approach to selection. Context was initially considered an aid to component testing. This same contextual information can also be used as non-functional assessment criteria, particularly for shortlisting. As a result, the Process was expected to use functional and non-functional criteria in the evaluation.

Another influence of the Weyuker paper on decisions in this investigation is in the choice of static or dynamic evaluation. The author put forward that testing needed to be done in context to provide a true indication of performance. An extension of this idea is that without testing in context, the evaluation cannot provide a complete assessment. There has been a trend to third party certification and testing of COTS (Voas, 2000) in the software engineering community. Although this had value in improving trust in COTS, it did not assess the software in the target context (Morris et al, 2001). Testing of the candidates was considered essential to the CdCE Process to determine suitability for the system under development. The process was refined to include testing as the primary method of evaluation. The decision to require dynamic evaluation was in line with Weyuker (1998) and reflected in the original name of the Process 'Context-driven Component Testing'.

5.3 Approach to Component Selection

The specification template developed in Spiral 1 is the basis for the selection process. The investigation considered existing processes through literature searches, looking for a process that met the goals of the project. There was also a requirement to utilise testing in context as part of the process. It was decided to create a new selection process as the review of literature did not identify a process that met the project criteria. The aim was to create a high-level process in order for it to remain generic enough to be widely applicable.

The first stage in developing the process was to define the scope, in terms of the parts of the selection process to include. From the literature, the high level stages are: 1) screening and 2) analysis (Cechich et al, 2003). A survey of existing techniques was carried out, and is summarised in Section 2.2 the literature review. This survey was combined with a walkthrough of an informal selection process to provide an intuitive perspective. The initial result was a five step process including specification, shortlisting (screening), evaluation (analysis), ranking and reporting. This was expanded to eight steps to allow for test generation and execution, and to separate the steps of the process to facilitate the development of strategies.

The eight steps in the CdCE Process can be seen in Figure 5.2, and is described in detail in Section 5.4. There are two points in the process where evaluations take place. The first is at the shortlisting stage (Step 2) and the second at the ranking stage (Step 7). Common techniques for evaluation use Weighted Sum Method (WSM) or AHP to aggregate the scores (Alves and Finkelstein, 2002). Ncube and Dean (2002) highlight issues with these approaches, including the assumption of independence between attributes, mixing of units of measure and the meaning of output values. An aim of this project has been to find an alternative to aggregation techniques for assessment. This is explored in Spiral 3 (Chapter 6).

Throughout the process, XML documents are used for inputs and outputs of each Step to provide transparency and documentation of results. These files also make it possible to replace steps in the process with alternatives that may be dictated by organisational standards. For example, a UML based specification and test generation setup could be used, or the process could be interfaced with workflow or QA software. For evolving systems, it would be advisable to use version control to track the changing parameters

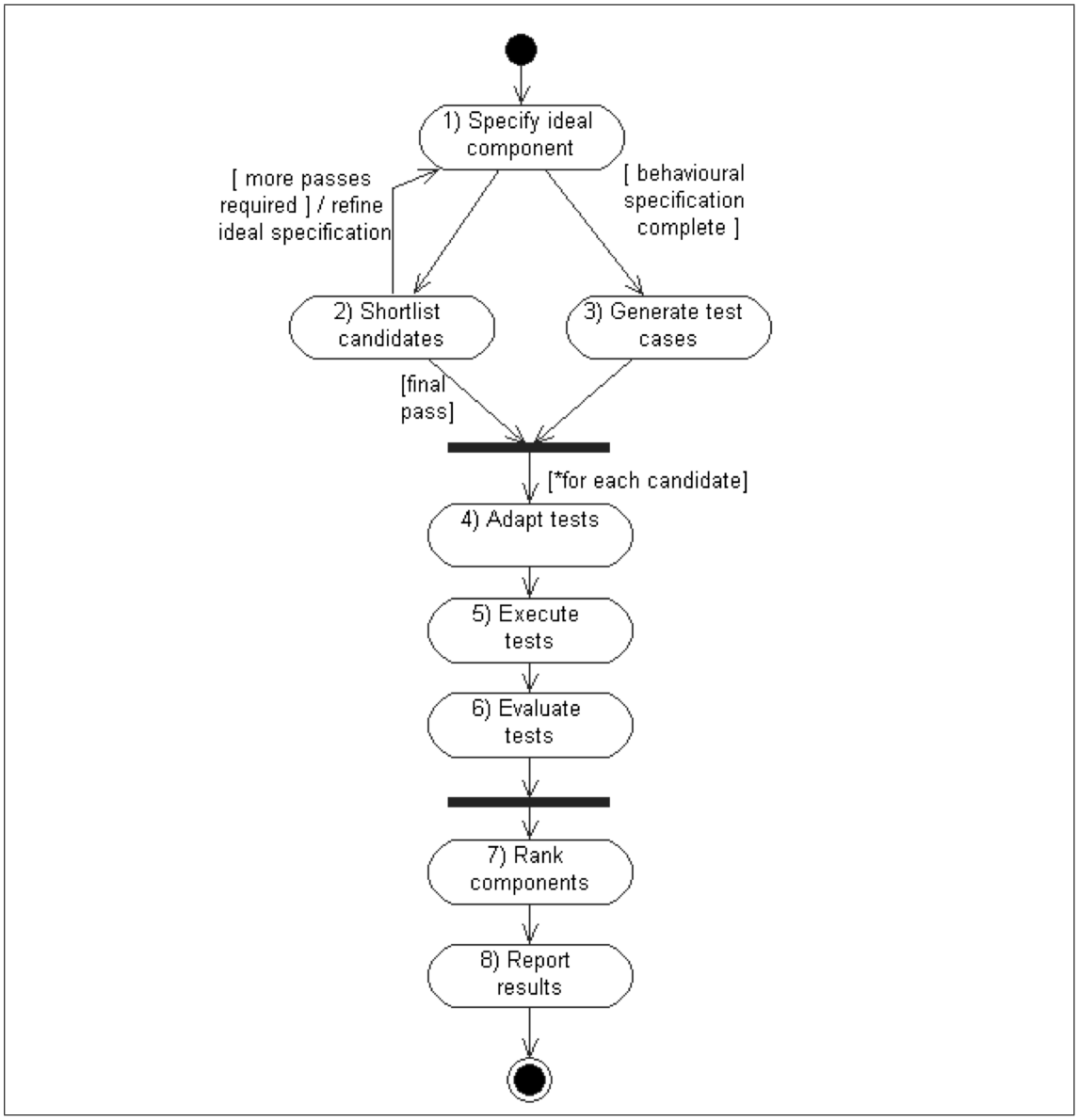


Figure 5.2: The CdCE Process

to the process over time.

Although one aim of the CdCE project was to provide a defined process, there is a need for flexibility as the selection task is intrinsically iterative. In the event there are unsatisfactory results, the specification can be refined and the task in question re-run. This may occur if there are too many or too few candidates or if unexpected results are returned from the particular choice of criteria.

For the CdCE Process, experts (application developers) define the ideal specification and thresholds (required values) on acceptance of results. They would also be involved in deciding to alter the ideal specification in response to the shortlist and when overall

results are satisfactory. While the testing steps require expertise, the Process encourages automation and reduces effort through the automated generation of test cases. This deals with issues of scale in terms of the number of candidates, the size of the repositories and with evolving systems and the revisiting of the selection process.

The evaluation and acquisition takes place in a parallel process to the remainder of the development for a component-intensive system. The selection feeds back into the development which may then feed back into the acquisition (Tran and Lin, 1999).

5.4 Implementation of the CdCE Process

Existing processes were surveyed for potential suitability to use in this project. These were abstracted into a bare bones, intuitive process - the basic steps required in all selection tasks. An activity diagram was developed to define sequencing and interactions between steps, and was then refined through the combination of approaches. Refinement of the process was informed by manual application to small case studies, resulting in the concepts of 'ideal component' to define the requirements, and 'candidate components' which are shortlisted components from the total pool. Context-related information was added into the formal specification within the ideal specification to support the testing of software.

There are eight steps in the CdCE Process illustrated in Figure 5.2. An overview of each step follows, with the rest of this Section describing each step in more detail.

Step 1 : Specify ideal component The desired component is described using a specification template (swvML).

Step 2 : Shortlist candidates The shortlisting step takes the ideal component specification and compares it to component information in repositories.

Step 3 : Generate test cases The ideal specification is used to generate tests to be applied to all the components. These are based on the interfaces and behaviour that are required.

Step 4 : Adapt tests Given the shortlist of candidate components from Step 2 and the tests generated in Step 3, the tests are adapted ready to be run against each of the candidates.

Step 5 : Execute tests Using an appropriate testing environment, all of the tests are executed for each candidate, and their performance is recorded.

Step 6 : Evaluate tests This Step takes the execution results from Step 5 and compiles them into an evaluation against a set of metrics.

Step 7 : Rank components Information from the preceding steps is combined to determine a ranking or comparison of the components.

Step 8 : Report results A report is generated to provide reasons for the decisions made and give information to assist with adapting the component to the target application.

To describe the CdCE Process in more detail, the inputs and outputs of each step are defined. These are in XML with stylesheets to allow viewing independently of the CdCE Process tools. The implementation of the steps in terms of tools and specific procedures took place in later Spirals, and is summarised in Section 5.8. The following description of inputs and outputs of each Step represent the Process at the end of Spiral 2.

CdCE Step 1 - Specify Ideal Component

This Step involves the task of defining the requirements for a third party component. The specification of the requirements is supported by the XML schema template from Spiral 1 with the behaviour described in a Z specification. The inputs and outputs of Step 1 follow:

Input 1.1 : Desired attribute values: The ideal component specification includes information on as many required attributes as possible, using the swvML data model as a guide for elicitation.

Input 1.2 : Priority of attributes: Each attribute is be marked with its level of priority. These are the hard and soft requirements for the selection. The assignment of priorities builds on the information given from Input 1.1.

Input 1.3 : Relationships between attributes: Particular combinations of attribute values may affect the suitability of a component. This input provides a way to record these dependencies, conflicts and interactions.

Input 1.4 : Formal specification: Information about the desired component behaviour will assist the test generation process and help increase the context used in the assessment of the component. The formal specification does not need to be complete; it may be limited to the interfaces provided by the component. Application developers can determine the most critical interfaces and functionality for the software and provide a specification for only those functions.

Output 1.1 : Ideal Specification: The above inputs are put together into the specification format to produce an ideal component specification in XML. A sample specification is included in Figure 5.3.

```
<?xml version="1.0"?>
<Description xmlns="http://www.scis.ecu.edu.au/swvML/1.0/"
          xmlns:dc="http://purl.org/dc/elements/1.0/"
          xmlns:swv="http://www.scis.ecu.edu.au/swvML/1.0/" >
     <dc:description  type="mandatory">XML editor</dc:description>
     <dc:detail type="mandatory">XML editor</dc:detail>
     <swv:licence type="mandatory">GNU General Public License (GPL)</swv:licence>
     <swv:devStatus type="mandatory">6 - Mature</swv:devStatus>
     <dc:date type="mandatory" min="01-01-2005" max="31-12-2007">31-12-2007</dc:date>
     <swv:technical>
             <swv:devLanguage type="mandatory">Java</swv:devLanguage>
             <swv:operatingSystem type="mandatory">Linux</swv:operatingSystem>
             <swv:systemRequirements>
                     <swv:memory type="mandatory" min="15" max="50">20</swv:memory>
                     <swv:diskSpace type="mandatory" min="30" max="50">40</swv:diskSpace>
             </swv:systemRequirements>
             <swv:Zspec>
             ...
             </swv:Zspec>
     </swv:technical>
</Description>
</xml>
```

Figure 5.3: Example ideal specification of an XML Editor

CdCE Step 2 - Shortlist Candidates

In this Step the ideal specification is used to filter the repository information to create a shortlist of candidate components. This may be a manual task, or an automated approach can be taken. If the shortlist is not satisfactory, the specification can be refined and the shortlisting repeated. A summary of the inputs and outputs for Step 2 are given below.

Input 2.1 : Ideal component specification: This XML specification is an output from Step 1 (Output 1.1).

Input 2.2 : Component repository information: This may be a list of component repositories and COTS brokers for manual searching, or an input file in an automated system. The choice of repositories for consideration is documented through this item. Any processing or transformation of data should also be documented.

Output 2.1 : Selected candidates The shortlisting process outputs a list of candidates and their known information in swvML format.

Output 2.2 : Analysis of repository data: It is possible that the shortlisting process can be improved by incorporating information about the available components. In some cases a field is missing in every record, or other mismatches may occur across the repository. This may mean tightening or loosening criteria, shifting ranges for numeric values or incorporating information from clustering the data. The shortlisting process advocates an overall analysis of the repository data with respect to the ideal component to inform the user.

Output 2.3 : Update to ideal specification: If there are changes to be made to the ideal specification, these updates are sent back to Step 1 to be incorporated before another shortlisting. The changes may include adding or removing criteria, or to alter a value (e.g. memory required). This would not be altering the formal specification, and don't affect Step 3 of the process (which relies on the Z specification).

CdCE Step 3 - Generate Test Cases

In this Step the formal specification of the component is used to create abstract test cases. Creating test cases that are 'abstract' provide a consistent basis for testing components. These can then be adapted for each component to create the actual tests. The abstract tests can include context-based tests that take usage profiles and critical functionality into account. Chapter 8 provides more details on the generation of test cases, as developed in Spiral 5.

Input 3.1 : Ideal specification: The specification is an output from Step 1 (Output 1.1). In Step 3 the formal specification and context schemas (in Z notation) are used to show the desired behaviour for test generation and the testing priorities.

Output 3.1 : Abstract test specifications: Using the specification, a set of tests is developed, manually or via an automated test generation process.

CdCE Step 4 - Adapt Tests

The tests generated in Step 3 are ready to be adapted to the interfaces provided by the candidate components. In this step the adaptation model is defined for each candidate to provide a transformation between the abstract tests (and therefore the target system) and the individual candidates.

Input 4.1 : Abstract test cases: These abstract test specifications are an output from Step 3 (Output 3.1).

Input 4.2 : Candidate interfaces: Information about the interfaces provided by each candidate is required for the adaptation. This involves the downloading of the actual components and/or documentation as interface information is not generally held in the component metadata.

Output 4.1 : Adapted tests: Tests for each candidate component, are stored in a format (e.g. XML) to suit the test environment.

Output 4.2 : Adaptation models: Information on how to adapt the components to suit the target system are recorded. These models are stored as XML and can assist the integration of the selected component.

CdCE Step 5 - Execute Tests

The tests are in XML format to allow transformation to particular harnesses. No specific test harness is dictated in order to allow flexibility for local needs such as platform, programming language and other factors. Alternatively, the tests may be run manually. The inputs and outputs for this Step are given below.

Input 5.1 : Adapted tests: Adapted tests were an output from Step 4 (Output 4.1).

Input 5.2 : Candidate executables: Each candidate or a demo version will need to be downloaded and installed.

Output 5.1 : Test results: Full results of tests for each component are stored in XML.

CdCE Step 6 - Evaluate Tests

This step takes the XML documents containing the test results and collates them using metrics to form a picture of each component's performance. Details of the metrics and evaluation are provided in Chapter 8, including the XML files for recording this information.

Input 6.1 : Test results: Test results output from Step 5 (Output 5.1).

Output 6.1 : Evaluation metric scores: The metrics for evaluation indicate the performance against tests and are compiled for each component. These allow the comparison between components to be undertaken.

CdCE Step 7 - Rank Components

In Step 7, the metrics collected for the candidates are used to inform their ranking. Many options for ranking such as WSM, AHP or Outranking can be considered. The CdCE approach to ranking is described in Chapter 8. From a generic viewpoint, there will need to be metrics and results, and a ranked list will be output from this Step.

Input 7.1 : Evaluation metric scores: These results of the testing are an output from Step 6 (Output 6.1).

Input 7.2 : Ideal metrics: The thresholds for each of the metrics are recorded in the ideal specification.

Output 7.1 : Ranked components: The ideal specification relating to the evaluation metrics is used to rank the components. This output provides a ranked list of components.

CdCE Step 8 - Report Results

In this Step all information harvested for and generated by the CdCE Process is collated and a report and recommendation produced.

Input 8.1 : Ranked list of components: This ranking is an output from Step 7 (Output 7.1).

Input 8.2 : XML documents generated during the selection process: From previous Steps, Outputs 1.1, 2.1, 2.2, 2.3, 3.1, 4.1, 4.2, 5.1, 6.1 and 7.1.

Output 8.1 : Report: The rankings from Step 7 and selected information from earlier steps are collated in the report. It includes an executive summary, index, ideal specification(s), component information, reasoning, comparison matrix, and adaptation models.

5.4.1 Context-based Tests

Third party components have been developed in a particular environment, and will have undergone testing for that environment and expected usage. To increase confidence of a suitable component being chosen, it is important to consider not only the target environment, but also the requirements for performance and reliability and how the component will be used and stressed. The concept of providing a representation for these tests emerged during the Process development in Spiral 2. The approach was to encode the information for the tests in Z schemas, consistent with the Z specification for generating the base tests. Original schema names were CX_values, CX_probability, CX_sequence, CX_frequency, CX_response, CX_critical and CX_environment (Maxville et al, 2003b). These were modified and instantiated in Spiral 5, which is discussed in Chapter 8.

5.5 Results: Using the CdCE Process

The purpose of this case study was to explore the feasibility of the CdCE Process. In this Spiral, the process was applied manually to provide proof of concept for the selection process; it was applied to a real world problem to identify issues associated with each step and the specifications, strategies and metrics required. The scenario is the selection of a component to provide scientific calculation functionality for a target system. The system provides the interface for the user to enter the information to set up the calculation, with the calculator component carrying out back-end calculations.

5.5.1 Calculator Case Study

Step 1 : Specification of Ideal Component

Crucial to the selection process is the clear definition of the required component. This is done by providing a description of the ideal component, including context information. This includes the platform, programming language (desirable), memory usage (disk and RAM), required functionality and usage information. Figure 5.4 shows the Z schemas for the state and initialisation of the ideal component, and are based on published specifications for a simple calculator and generic trees (Barden et al, 1994).

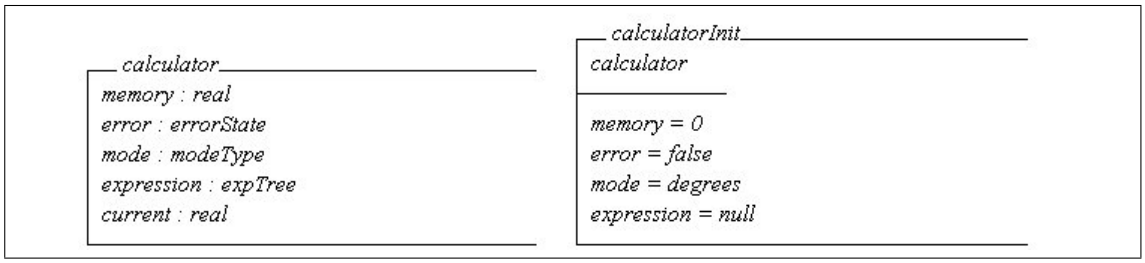


Figure 5.4: State and initialization schemas

Mandatory operations are included in the specification and are indicators of the required interfaces. Additional information about the behaviour of the component is used for test generation, without implying that the candidate components have to use the same internal logic.

The application developer may have usage profiles or other information to guide test generation towards important functionality or input values. This information is recorded in predefined context schemas that are understood by the test generator. An example is given in Figure 5.5, where the most common mathematical operation is known to be addition. Usage based tests can then be generated with 80% (0.8) of the test cases focussing on addition operations.

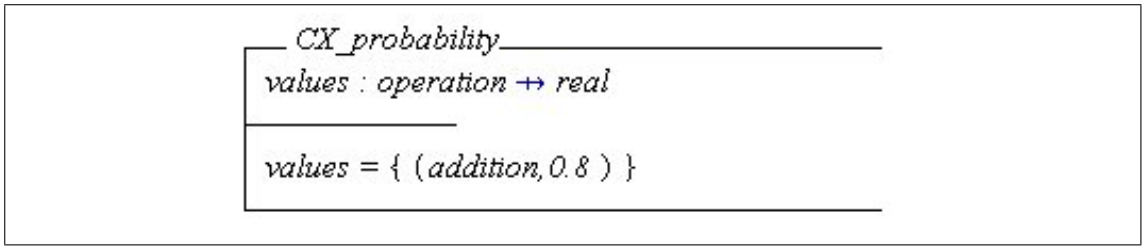


Figure 5.5: Calculator context schema - CX_probability

Step 2 : Shortlisting and Specification of Candidates

The selection process was carried out manually, but enacted using well-defined selection rules to simulate an automated process. The targeted maximum number of candidates is 7 plus or minus 2 for the manual shortlisting process (Miller, 1956). Automation will allow for these limits to be increased or removed, with the user having the option to set the number of candidate components returned. The shortlisting process gives the application developer a structured and repeatable approach to sourcing components. This saves time, allows for a wider search for candidates (through eventual automation) and clear, traceable reasoning for selections made. The ideal specification (and changes made to it) are artefacts that can be used for documentation and for future selection tasks as the system evolves.

Site Code	Total Available Entries	First Pass	Second Pass	Third Pass	Final Pass
I	8000+ components	2	1/2	1/1	1/1
II	533 components	2	0/2	-	-
III	12,212 projects	113	16/113	7/16	7/7
IV	36725 projects	173	36/173	4/36	4/4
V	30,000+ titles	67	11/67	3/11	0/3
Total	$\tilde{8}$7,500 listings	357	64/357	16/64	12/16

Table 5.3: Shortlisting results

The initial task was to select repositories to search for components. Five web sites were chosen, each offering access to software descriptions and implementations. Two of the sites were specifically component brokers, two were foundries for open source projects and one offered a large selection of freeware and share-ware applications. The criteria for each pass of the selection process are taken from the ideal component metadata, focusing on the description and the environment/platform requirements. The first pass identified any software that included 'calculator' in the description. This very broad criterion gave an indication of the number of possibilities the search would need to consider. A total of 357 software possibilities were returned for the five sites. The process then uses any criterion of the ideal component specification to reduce the number of values returned. Table 5.3 shows the number of possibilities remaining after each of the passes, described below. The sweep criteria for each pass are given in Table 5.4.

The criteria for the second pass focused on ways to easily rule out a large proportion of the components based on metadata in the ideal and available components. The

Code	Selection Criteria
A	Description includes 'scientific'
B	Is a 'calculator application or component'
C	Not specific to X11 (or GTK/Gnome/KDE/Motif)
D	Not a specific calculator emulator
E	Environment is (Windows or O/S independent)
F	Has description
X	Has scientific functionality (not just basic arithmetic functions)
Y	Has a programmable interface (not just mouse/GUI)
Z	Has files released/can access web pages

Table 5.4: Shortlisting criteria

second pass used criteria A-F to reduce the possibilities from 357 to 64. The third pass introduced criteria X-Z and also found some additional software that failed A-F upon further investigation. This resulted in sixteen matches. The final pass came as a result of gathering information to fully specify the candidates in CdCE format. The information uncovered in down-loading and reviewing the software and its documentation exposed three possibilities that failed the selection criteria already used. There were also four duplicates (across repositories), resulting in nine candidate components to fully specify and take through the evaluation process.

Step 3 : Test Generation

The generation of tests for the selection and evaluation process is based on the Z specification for the ideal component and its context, and current strategies as described earlier. The approach followed is similar to Dick and Faivre (1993) and Hall and Hierons (1991), where specifications are transformed to disjunctive normal form, from which operations are generated to represent each partition of the input space.

For this case study, the tests were generated manually. The four categories of test cases target the functional areas: memory sequences, mode sequences, expression sequences and expression+memory sequences. These were based on the variables in the Z state schema and provide full coverage of the ideal component operations. Each variable had two partitions - valid and invalid, resulting in the pairs of test cases (e.g. 1 and 1a in Table 5.5). These tests served to provide a quick assessment of the available functionality. Values from each partition were then substituted through a random test data generation process.

Name	Valid and invalid cases
Memory (1)	inputExpr(valid) >> storeMemory >> clear >> recallMemory
Memory (1a)	inputExpr(invalid) >> storeMemory >> clear >> recallMemory
Mode (2)	setMode(valid) >> getMode()
Mode (2a)	setMode(invalid) >> getMode()
Expr (3)	inputExpr(valid) >> evaluate()
Expr (3a)	inputExpr(invalid) >> evaluate()
Expr2 (4)	inputExpr(valid) >> storeMemory >> inputExpr(valid) >> recallMemory >> evaluate()
Expr2 (4a)	inputExpr(valid) >> storeMemory >> inputExpr(invalid) >> recallMemory >> evaluate()

Table 5.5: Test cases for calculator case study

Step 4 : Test Adaptation

Test adaptation is carried out by combining the test sets from Step 3 with the adaptations from Step 2. The result is a set of tests for each candidate component that exercises identical functionality and data for consistency across all components. The adaptations in this case were syntactical - there is a clear mapping between the generated tests and the actual transcript.Test adaptation was carried out manually in this case study.

Step 5 : Test Execution

The tests were run manually against each of the components. The possible results were pass, fail or not applicable (where the functionality was not present). The test suite did not expose any failures, where the functionality was available in the component. The shortlisted candidates passed all of the tests.

Step 6 : Evaluation of Results

This Step converts the raw test execution results, selection criteria and adaptation information to create a picture of each candidate's suitability. The test execution results were converted into a score indicating the performance of the component against functional and usage based testing (number tests passed/number tests in total). The selection process also provided useful information about the component's suitability, based on comparison with the ideal component specification. Another facet affecting the suitability of a component is the effort required to adapt the component to its target system. These pieces of information from Steps 2 and 5 were collated for each component, ready for ranking in Step 7.

Metric	Result	Score	(s)	Conversion Rating /10
% Features available	75%	7	s	7
Excess Features (categories)	12	10	10-s	0
% Interfaces needing adaptation	100%	10	10-s	0
Maturity (years since first release)	1	1	2x s	2
Maturity (stability)	4 (Beta)	4	2x s	8
Cost	free	0	10-s	10
Test results	75%	7	s	7
Simple Total				34/70
Simple Percentage				49%

Table 5.6: Component metrics and ratings

An example of the case study results for a particular component is given in Table 5.6. Following the approach of Solberg and Dahl (2001), the rating for each metric is a conversion from the raw results to a value in the 0-10 range (s). Higher ratings indicate component features are more suited to requirements. A simple indicator of the overall performance of the candidate component is given by summing the ratings or calculating the percentage of rating points achieved.

Step 7 : Ranking of Candidates

Given the ratings for each component against each metric, it is possible to compare and rank the components. By default, the CdCE Process considers all metrics equally. There is a facility to add weightings to each metric to suit a particular project, or an organisation's quality or standards requirements. For example, an organisation may decide that the risk of immature/unstable software or of excess functionality is of high importance. It would then add a weighting to these metrics to increase their effect on the rankings. A cutoff value may also be used for high priority metrics, disqualifying the candidate from the rankings. A simple weighted score sum (Solberg and Dahl, 2001) is used to determine the result for each component:

$$\text{result} = w1 * s1 + w2 * s2 + w3 * s3 + ... + w7 * s7$$

A number of combinations of weightings were investigated. Table 5.7 shows how the overall scores for three components are affected by varying the weightings of the metrics. In this case, C3 consistently achieves the highest scores.

Weighting Pattern	C2	C3	C9
Default - all metrics equal	33	44	34
Tests have triple value, all else = 1	37	64	48
Features has triple value, all else = 1	37	64	48
Adaptation has triple value, all else = 1	33	44	34
Maturity has triple value, all else = 1	57	64	54
Risk factors tripled, all else = 1 (maturity, adaptation and excess features)	71	72	54

Table 5.7: Results of component ranking

Step 8 : Report on Results

Once the candidates have been ranked, the resulting information is presented as a report on the process and advice on the most suitable component(s) for the problem being addressed. The report includes features and shortcomings of the component(s), and the adaptation required to integrate the component into the target system. It can also provide information on the shortlisting process and criteria used for selecting candidates. It serves as the justification of the choice of component for inclusion in the system documentation.

5.5.2 Case Study Observations

This trial of the CdCE Process indicated that it could be used manually to undertake a real world selection process. Items highlighted as needing clearer definition include: a method for comparison/ranking; metrics for evaluation; test generation technique(s); adaptation models; and supporting XML schemata and XSLT transformations. Also highlighted was the impact of scoring method and weighting across the selection criteria.

The shortlisting process exposed wide variation among repositories in terms of total software titles, candidates returned and the documentation provided. These differences should be recorded in a knowledge base to allow them to be taken advantage of. For example, Site IV ordered the results by project activity. This meant that after the first 44 projects, the usual reason for failing the third pass was that there were no files available for the project. By taking the activity metric into account, it would have been possible to reduce the shortlisting effort by 75%. Sites I-IV had significant amounts of information about each piece of software. Site V had little documentation and meant that the developer web site had to be accessed for each possibility in the third pass.

In terms of usefulness for discovering components, Sites I and II are targeted to the component market, and tend to be better documented. Unfortunately they did not

have many components matching the case study criteria. Sites III and IV are aimed at encouraging open source development. Their projects may not be stable, but there are good options for reuse as interfaces are accessible. There are a large number of projects, varying in maturity and level of documentation. Site V mainly provides standalone applications, so little information about integrating the software is available. The site does offer a large number of titles and the available software may prove suitable for integration into other component-based systems.

5.6 Spiral 2 Evaluation

The second Spiral of the investigation addressed **RE2** in developing a process for component selection. To evaluate the Spiral, the stakeholder Win conditions (Table 5.2) and Spiral goals (Table 5.1) are now discussed. This evaluation is based on the work of Spiral 2, although very relevant work took place in Spirals 3 to 6. That work, and its impact, will be discussed in Section 5.8.

One issue with existing processes for component selection is a lack of uptake (Li et al, 2006) which may be improved by providing simpler processes: better indicating the benefits or flexibility to integrate with existing work practices. This Process, developed with consideration of these issues, is adaptable to allow for alternative implementations of Steps to match the local development environment.

The Win conditions for application developers are to have an intuitive process with justification and low overheads. The Process itself was developed to take the basic steps of a generic selection process. It is able to be followed manually, while also providing options for automation. At each Step, documentation is created which can later be used to justify the selection. This is also a win condition for quality assurance. The final win condition for the application developers is that the Process has low overheads, with the main overhead for using the Process at Spiral 2 in providing the ideal specification. As this is a formalisation of defining the requirements, the overhead can be considered low.

Component developers are able to see clearly how their components are assessed through the Process and the use of the ideal specification. The Process requires information from repositories at the shortlisting stage (Step 2) and in the downloading of executables and documentation at Step 4. Brokers can assist integration of selection

processes by providing search facilities with the required fields and/or XML (or similar) output of the repository data. Also of interest to brokers is that fully-functional versions of the software would need to be provided for evaluation.

The academic Win conditions include the need for a justifiable, repeatable process, which has already been discussed. In addition, the low coupling on Steps and the use of XML for documentation, inputs and outputs will assist automation. The structure of the CdCE Process has been developed with reference to processes described in the literature and includes the phases of filter, evaluate and selection. The outcomes of this Spiral were peer reviewed and presented at the Asia-Pacific Software Engineering Conference (APSEC) (Maxville et al, 2003b) and at the Postgraduate Electrical Engineering and Computing Symposium (PEECS) (Maxville et al, 2003a).

The preceding review supports the view that the stakeholder Win conditions have been met.

5.6.1 Spiral Goals

SPIRAL 2	**Purpose** **Issue** **Object** **Context**	**Evaluate** **effectiveness of** **the developed selection process** **Spiral 2**	Result
Goal 2A	Focus	**Quality**: *Produce a structured, repeatable process for selecting and evaluating components*	
	Viewpoint	Quality Assurance personnel	
	Q2A1	Is the process defined according to accepted standards?	YES
Goal 2B	Focus	**Usability**: *Consider existing processes and organisational requirements, provide tools and automation*	
	Viewpoint	Application developer	
	Q2B1	Can the process interface with organisational/development processes?	PART
	Q2B2	Is the process easy to understand and use?	YES
	Q2B3	Has been tested on real world examples?	PART
Goal 2C	Focus	**Intelligence**: *Identify areas where intelligence and automation can be applied*	
	Viewpoint	Application developer	
	Q2C1	Have areas for automation been identified?	YES

Table 5.8: GQM Summary - Spiral 2 (Part 1/2)

The following discussion refers to the goals listed in Tables 5.8 and 5.9. Each goal is included as a heading followed by a discussion relating to that goal. Many of the goals were further addressed by work in later Spirals.

SPIRAL 2	**Purpose** **Issue** **Object** **Context**	**Evaluate** **effectiveness of** **the developed selection process** **Spiral 2**	Result
Goal 2D	Focus Viewpoint	Innovation: *Consider novel approaches to the process* Academia	
	Q2D1	Is there anything novel in the process?	YES
Goal 2E	Focus Viewpoint	Dynamics: *Allow for dynamic assessment with context* Application developer	
	Q2E1 Q2E2	Does the process allow for iteration? Context is included in the selection?	YES YES
Goal 2F	Focus Viewpoint	Reuse: *Where possible make use of existing code and artefacts* Application developer	
	Q2F1	Has the work reused external resources?	PART

Table 5.9: GQM Summary - Spiral 2 (Part 2/2)

Quality

The outcomes of this Spiral adhere to current standards for documenting processes, with the CdCE Process drawn as a UML activity diagram. The description includes input and outputs for each Step. Further details on the implementation of the Steps and the strategies that have been developed are in subsequent chapters.

Usability

An example of the process integrating with the system is shown in Section 5.3. To better address this question, the process would need to be trialled in development environments, which was out of the scope of this project.

The Process is defined in an easily understood form - the activity diagram - and a top-level understanding is easily attainable. The Process can be used manually to encourage better structure for selection, or with automation to allow for larger repositories and more complete documentation.

The initial cases worked through were real world examples, with manual data collection from a variety of sites. The case study in Section 5.5 shows the process applied to assessing real software from public repositories. For full satisfaction of this question, the Process would need to be trialled in a software development environment. Thus it is considered to be partially satisfied.

Intelligence

The areas identified for automated support include: 1) automated documentation in XML between steps, 2) automated test generation, 3) an automated test oracle and 4) automation to support selection based on a set of criteria. These were expected to be able to include some machine intelligence as an outcome of future Spirals.

Innovation

The Process includes the basic steps for a dynamic evaluation of components. These generic steps allow for flexibility to explore various strategies and implementations in this project. It also allows localisation of the Process to match and evolve with organisational standards and contexts. This detachment from any particular implementation is novel and is distinct from the decision support approach to choosing a selection process described in Mohamed et al (2007b).

The Process separates the test generation from the shortlisting. The tests are generated from the requirements, allowing testing to be truly comparative as the tests are then adapted and executed for all candidates. The functional specification is encapsulated in the Z specification, which also provides the contextual information for testing. Together, the approach to test generation and the use of Z notation is novel in the component selection literature.

The concept of the ideal specification is also novel. It provides a standardised form to record the requirements, based on the work of Spiral 1. The schema can be used for describing all components of interest and the requirements for the selection.

Dynamics

As the selection process is interdependent with the wider software development, and with the suitability of available candidates, it is important to be able to adapt to each of these inputs. At the wider process level, the repeatability of the CdCE Process supports revisiting the selection task within a project. The manual case study utilised a procedure of progressively adding selection criteria to filter for candidates. This iteration allowed the selector the flexibility to respond to the available candidates, which may not fully match the ideal specification. It is also possible to re-specify the requirements after shortlisting, which is indicated by the loop back between Step 2 and Step 1.

Context is included in the selection on a number of levels. Firstly it is included through many of the non-functional attributes in the ideal specification. Context is also represented in the tests, not only in the context schemas but also in the requirement for testing in the local environment - preferably the target context. A third level of localisation is the adaptability of the process to local organisational needs and tools.

Reuse

The process was influenced by existing processes, but did not reuse any process as a whole. However, there is some (internal) reuse through the specification template being utilised for a range of tasks throughout the Process.

5.7 Spiral 2 Review and Plan

An evaluation of this Spiral's outcomes gave positive results against all of the stakeholder Win conditions. The activity diagram and areas for automation were identified and met project requirements for usability, context awareness and simplicity. In particular, the process is repeatable and self-documenting through the XML linkages between steps. The GQM evaluation was positive on all questions, with only two areas rated as partially satisfied. However, Q2B1 (**usability**) would need the Process to be trialled in an organisational setting to be answered with confidence. This is beyond the scope of this project and may be considered in future work. Q2F1 (**reuse**) was also not completely satisfied, due to the creation of yet another selection process, where it had been hoped that an existing one could be used. However, the need for a process for this project indicates a gap in the literature which the CdCE Process addresses.

The work from Spiral 2 has undergone external review through conference papers and in an internal Faculty presentation. The contribution of this Spiral is the Process itself, particularly in the repeatability and self-documentation supported. The suitability for automation sets it apart from most selection processes, which are not ready for intelligent support (Ruhe, 2002).

Moving forward to planning for Spiral 3, the researcher identified parts of the Process to target for automation based on the manual use of the Process in Spiral 2. This resulted in a commitment to investigate test generation and filtering/shortlisting in subsequent

Spirals.

5.8 Post-Spiral Update

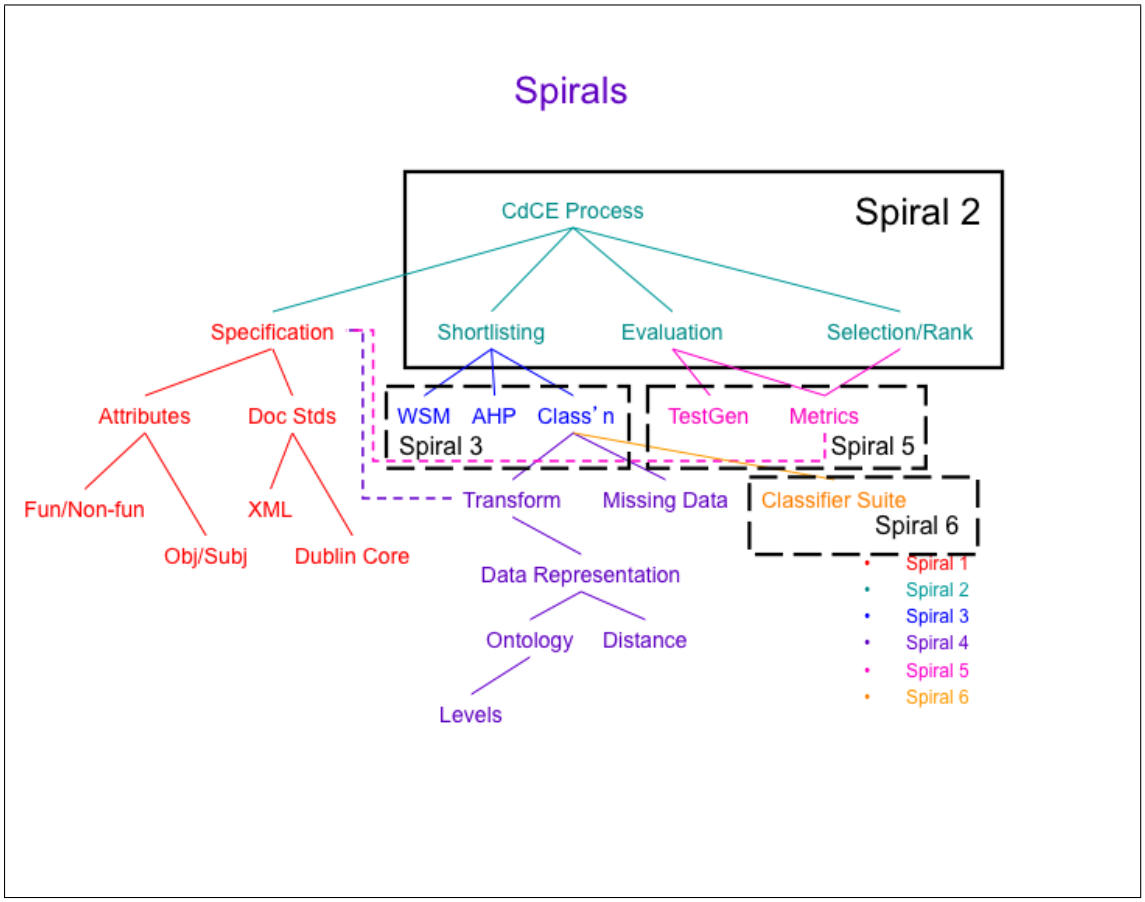


Figure 5.6: Updates to the CdCE Process in later Spirals

The preceding Sections detail and review the work carried out in Spiral 2. Later Spirals have incrementally instantiated, extended and altered the CdCE Process as shown in Figure 5.6. This took place during the exploration of **RE3** in Spirals 3 to 6. Impact on the Process was greatest in Spiral 3 (shortlisting approach), Spiral 5 (implementation of testing and evaluation) and Spiral 6 (support for shortlisting through ClassifierSuite) The final version of the CdCE Process will now be described, and is applied to a case study in Spiral 7 (Chapter 10). The final version of the inputs and outputs of all of the Steps are given in Table 5.10.

As the updates have impacted on every Step of the Process, they are now described on a Step by Step level.

Step	Inputs	Outputs
Step 1	Desired attribute values Priority of attributes Relationships between attributes Formal specification of behaviour in Z notation	Ideal specification (XML)
Step 2	Ideal specification (XML) Component repository information (XML)	Selected candidates (XML) Analysis of component data Update to ideal component specification (if applicable)
Step 3	Ideal specification (XML: Zspec)	Abstract test cases (XML)
Step 4	Abstract test cases (XML) Candidate interfaces	Adaptation models Adapted tests (XML)
Step 5	Adapted tests (XML) Candidate executables	Test results (XML)
Step 6	Test results (XML)	Evaluation metric scores (XML)
Step 7	Evaluation metric scores (XML) Ideal metrics (XML)	Ranked/ordered list of components (XML)
Step 8	Ranked list of components XML documents generated during the selection process	Report

Table 5.10: CdCE Inputs and Outputs

Step 1

There has been no conceptual change to Step 1, however the detail of the XML Schema describing the components and ideal component has changed. This includes the implementation of improved data representation and ontologies and is discussed in Section 4.7. The Schema was extended to include evaluation metrics developed in Spiral 5, which are described in Section 8.2.

Step 2

The shortlisting or filtering for a shortlist was targeted for automation. In Spiral 3, initial work considered using the Weighted Sum Method (WSM) or the Analytical Hierarchy Process (AHP) which were reported as widely used (Ncube and Dean, 2002). Limitations of these techniques led to investigation of alternatives, including Artificial Neural Networks and classifiers. This exploration is described in Chapter 6 and resulted in the adoption of machine learning classifiers for the shortlisting task.

The Process has strong support for iteration. In Spiral 6, an approach was taken to flatten the iteration through the selection criteria by creating a suite of classifiers

to provided the results of using all combinations of criteria from the ideal specification. While the ClassifierSuite does not remove the need to iterate back to Step 1, it does capitalise on the automation of the filtering and assists the developer in choosing among the given selection criteria when a trade-off is required. Full details of the ClassifierSuite are given in Chapter 9.

Step 3

Spiral 2 included the decision to generate abstract test cases from the specification. The test generation itself was implemented in Spiral 5, and did not diverge from the expectations of Spiral 2. Some redefinition of the context-based testing (from Z schemas) was carried out to match the evaluation metrics developed in Spiral 5 (Section 8.2).

Step 4

The description of adaptation in Spiral 2 was quite high level. The expectation for an automated process for adaptation was not possible within time and resources. This resulted in a manual adaptation process being adopted in Spiral 5 (Section 8.4).

Step 5

The execution of tests was envisioned as utilising automation. As each testing environment is different, it was decided in Spiral 5 to automatically generate tests and to leave the execution of them to the user. The tests themselves are recorded in XML to simplify their use in a test harness or other automated environment.

Step 6

Evaluation of tests is a step turning raw results into a set of compiled results. In Spiral 5 the specifics of the metrics were determined, guiding how the raw results populate indicators of performance in base and contextual tests. These are detailed in Section 8.2.

Step 7

As with shortlisting in Step 2, the ranking of components was considered for new strategies and approaches. During Spiral 2, AHP and WSM were under consideration. Having developed the classifier approach for shortlisting in Spiral 3, the same approach could

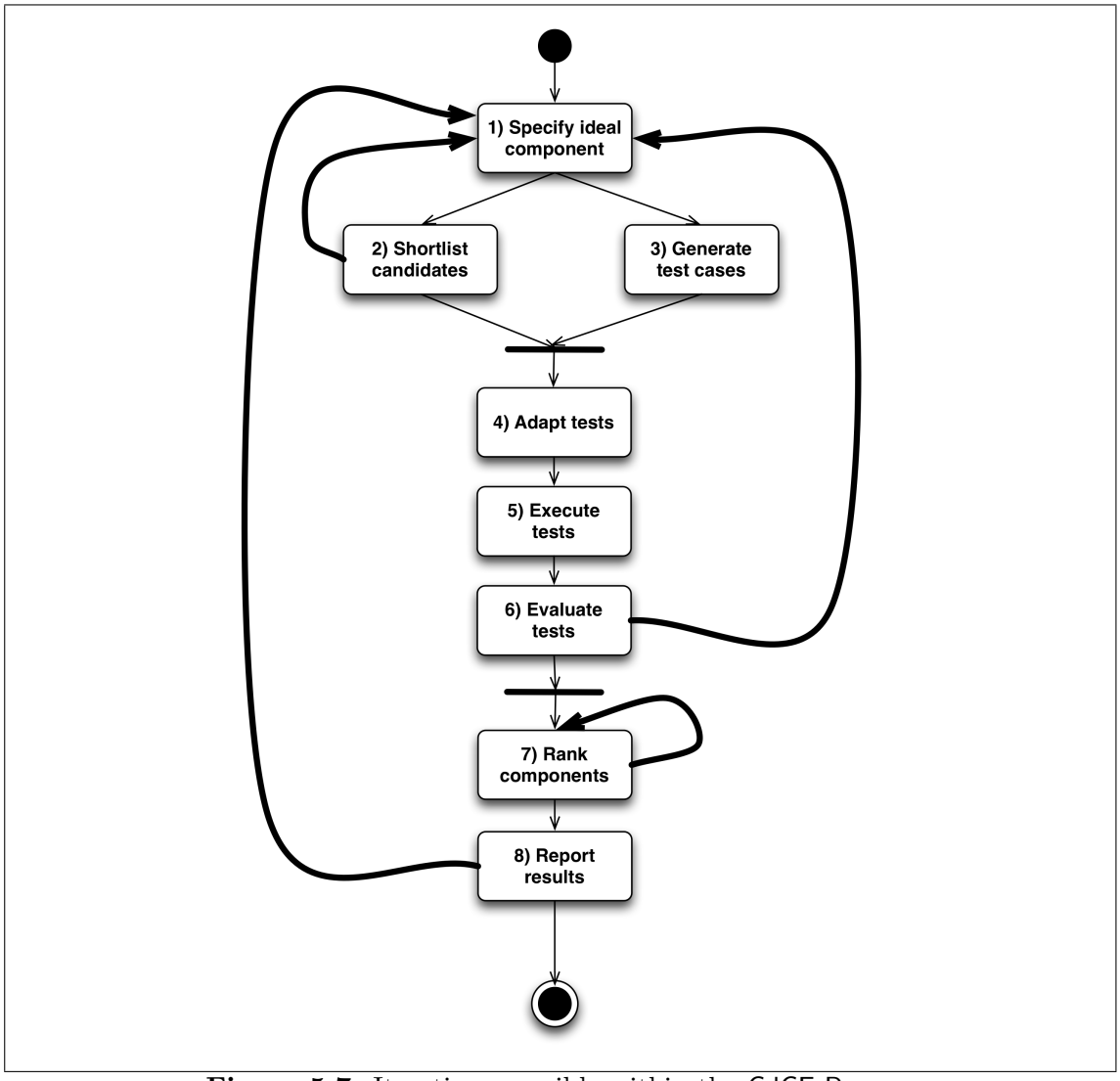


Figure 5.7: Iterations possible within the CdCE Process

be applied to ranking based on the evaluation metrics. These metrics are extensible and interchangeable, however those used in this project align to the testing approach and the compiling of results from Step 6.

Step 8

The reporting of results was always considered to be a compilation of the documents from all steps of the Process. In the final version of the Process, this included the inputs and outputs from all Steps (XML presented through XSLT), along with descriptions of internal decisions, such as the choice of criteria and the shortlist using the ClassifierSuite in Step 2.

Iteration

The initial version of the CdCE Process has one point of iteration in the activity diagram (Figure 5.2). The CdCE Process includes four iteration zones for the tuning of parameters. The first is between specification and shortlisting (Steps 1-2), the second from specification to testing (Steps 1-6) and the third in ranking (Step 7). A fourth iteration is to repeat the entire process with modified criteria. Iteration is included to improve results and, as each change is documented, does not break the goals of structure and repeatability. The refinement of the specification for the shortlisting is expected to occur on most selection tasks. All of these iteration options can be seen on the expanded Process diagram (Figure 5.7).

5.8.1 A Pattern for Component Selection

Over the course of the investigation, the CdCE Process was made flexible to allow the exploration of specific techniques at various phases of the selection. This generic approach makes it possible to consider the high level Process as a pattern for component (and more general) software selection. In instantiating the pattern, users may adopt the CdCE techniques, create their own techniques or use a blend of the two.

This description of the CdCE Pattern for Component Selection (see Table 5.11) uses the template provided by Gnatz et al (2002). Following the related theory for living software development processes, this description is at the meta-model level (Gnatz et al, 2002), with the CdCE Process itself at the 'model' level.

Name:	**Context-driven Component Evaluation**
Also Known As:	N/A
Author:	Valerie Maxville
Intent:	As reuse of third party software increases, developers face the task of selecting between alternatives. This process provides a systematic, intuitive approach to the selection task.
Problem:	The scenario is that the project requires one or more third party software items. These may be standalone applications or software components to form part of the final system.
Context:	The selection should begin in the requirements or design phases. For component-intensive systems, the pattern can be used in tandem with the rest of the system development. Developers will need to have information on the requirements (ideal component specification) and access to a software repository.
Solution:	**Eight Step Process:** Step 1 : Specify ideal component: The desired component is described using XML and Z notation. Any priorities between attributes should be included as well as relationships between attributes. Step 2 : Shortlist candidates: The shortlisting step takes the ideal component specification and compares it to component information on repository websites. Step 3 : Generate test cases: The behavioural specification in Z notation is used to generate tests to be applied to all the components. These are based on the interfaces and behaviour that are required. Step 4 : Adapt tests: Given the shortlist of candidate components from Step 2 and the tests generated in Step 3, the tests are adapted ready to be run against each of the candidates. Step 5 : Execute tests: Using an appropriate testing environment, all of the tests are executed for each candidate, and their performance is recorded. Step 6 : Evaluate tests: This step takes the execution results from Step 5 and provides an evaluation against a set of metrics, ready for inclusion in the ranking classification. Step 7 : Rank components: Information from the preceding steps is combined to determine a ranking or comparison of the components. Step 8 : Report results: A report is generated to provide reasons for the decisions made and give information to assist with adapting the component to the target application.
Consequences:	The pattern provides an intuitive approach to selection which was developed to be open to automation and tool support.
Known Uses:	The pattern is instantiated as the CdCE Process. The Process includes a specification template, tool support and process guidelines and has been applied in case studies. The Process is also able to be applied manually.
See Also:	

Table 5.11: Pattern Definition: Context-driven Component Evaluation

5.9 Summary

This Chapter followed the development of the CdCE Process, initially through the work in Spiral 2 and with updates in the later Spirals fleshing out the detail of the Process. As a final product, the CdCE Process is structured and repeatable, with support for automation and self-documentation. Context is a repeating theme in the Process, which is supported through non-functional selection criteria, testing in the target environment and context-based tests. The Process developed met required criteria for stakeholders and review, either within Spiral 2 or in setting the path towards their implementation in later Spirals.

Contributions from Spiral 2 and the CdCE Process include: the definition of structured, repeatable process for software selection; the provision for context-based evaluation; and the pattern for software selection as described in Section 5.8.1.

In the next Chapter, Spiral 3 focusses on the shortlisting process. This considers existing techniques for comparison and their benefits and short-comings. A key goal is to increase the number of components that can be assessed through the use of automation.

Chapter 6

Shortlisting Candidates

This Chapter describes the approach taken for shortlisting or screening the components. This is the first Spiral to address **RE3** - the development of strategies and techniques for software component selection. Spiral 2 identified some areas of the CdCE Process to target in future Spirals, with shortlisting (Step 2) being the first one addressed.

Spiral 3 aimed to find an approach to provide automation for the selection process. A number of techniques were considered, including weighted sum method (WSM), Analytical Hierarchy Process (AHP), expert systems, artificial neural networks and machine learning classifiers. Experimental case studies led to the selection of C4.5 classifier as it outputs a decision tree. The Weka implementation provides an engine to the classification-based shortlisting approach that has been developed. This novel application of decision tree classifiers, and the supporting tools and procedures, is a contribution of the study (C4).

The goals for Spiral 3 are listed in Table 6.1. Key points were that the shortlisting be structured and repeatable (**quality**). Iteration and flexibility had been identified as important in Spiral 2 and the approach needed to minimise the effort involved in supporting the **dynamics** of selection. Automation and ease of use (**usability**) were to

SPIRAL 3	GOALS
Quality	Produce a structured, repeatable process for shortlisting components
Usability	The shortlisting process must be automated and understandable
Intelligence	Apply machine intelligence techniques and strategies to shortlisting
Innovation	Consider a wide range of options for shortlisting to find a novel solution
Dynamics	Allow for change and iteration
Reuse	Where possible make use of existing code and artefacts

Table 6.1: Goals for Spiral 3

be provided, with the expectation that machine intelligence techniques would form part of the solution. A range of techniques were under consideration with the expectation that this could be an area of **innovation** in the project. It was expected that an existing machine learning techniques could be applied in a novel way (**reuse**). These goals are revisited in the evaluation at the end of the Chapter.

6.1 Spiral 3 Overview

Spiral 3 addresses aspects of **RE3**: to select, implement and evaluate strategies to assist in component selection, based on the specification and process from Spirals 1 and 2. Specifically, Spiral 3 looks at the shortlisting of candidates. The focus from the user perspective are the **Select Component**, **Adapt Process**, **Revisit Selection** and **Assess Selection** use cases (Figure 6.1). This involves two actors, the application developer and the quality assurers. The application developer wants an effective mechanism for shortlisting within the CdCE Process. It should also be possible to repeat the shortlisting easily, whether within the initial system development, or as the system evolves. From a quality assurance perspective, the shortlisting needs to provide documentation and justification to support decisions.

Candidate screening (shortlisting) involves querying component brokers and repositories to seek out likely matches to requirements. Shortlisting uses a subset of the selection criteria, commonly the description, functionality, development language, component framework and platform. Most Internet repositories, such as Component Source, hold this information in easily accessible metadata. When using in-house repositories it is possible to use more knowledge of the documentation standards and even search to match on formal specifications (Atkinson, 1997).

Some of the problems identified with existing component selection techniques include issues with assumed independence of criteria, and the need to understand and simulate the reasoning used by experts when undertaking a manual selection. In an intuitive assessment, an expert gradually builds a picture of a components performance. This could be regarded as a simple checklist, but more likely the expert is mentally balancing up the positive and negative attributes of a component. An example would be that the developer has used software from a particular vendor in the past. This will colour

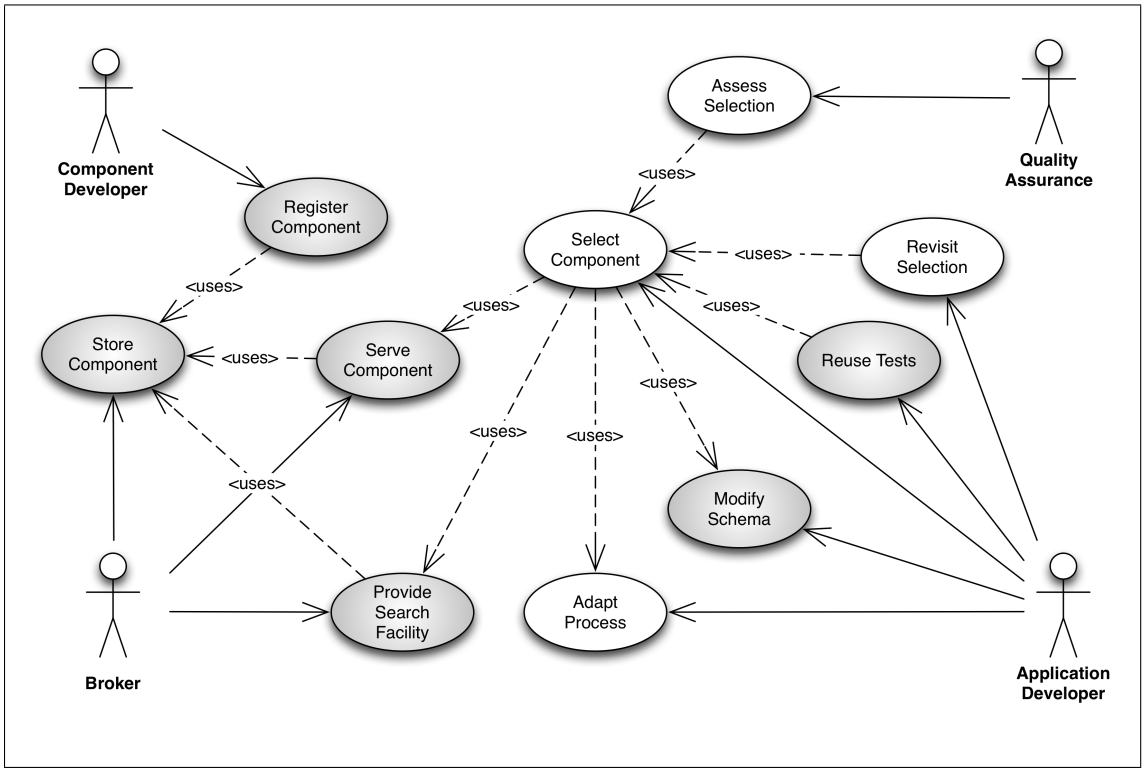


Figure 6.1: Use cases for component selection, the focus of Spiral 3 (those not in the scope for this Spiral are greyed)

their perception for assessing attributes such as trust, reliability, risk and maintenance. Unfortunately the intuitive assessment is limited in the number of candidates that can be considered, is not repeatable, and unlikely to be well documented.

The original CdCE approach (Spiral 2) to screening used a heuristic search, driven from the ideal specification. The search process goal was to determine a shortlist of components within a numeric range (e.g. 3-10 candidates). The heuristic search works in a series of passes, adding criteria and reducing the list of components on each pass. If it is found that mandatory criteria are causing too few candidates to survive a pass, those criteria can be loosened, or it can be reported that there are no components available.

This Spiral includes a series of investigations and evaluations of strategies and tools applicable to shortlisting. These are presented in Sections 6.3 to 6.5, along with the discussion of their suitability to the investigation. A discussion across the three approaches in Section 6.6 with the decision taken to make use of the C4.5 classifier for shortlisting. A case study was undertaken and analysed to identify ways to improve performance in terms of selection outcomes.

The evaluation criteria for the stakeholders are given in Table 6.2. When looking at

Stakeholder	Win Conditions
Application Developers	Strategies are beneficial. Justifiable results. Low overhead to use strategies.
Component Developers	Know how their component is assessed and compared.
Component Brokers	How to integrate with the repository. What information is required and when.
Quality Assurance	Documentation and justification of decisions.
Academia	Validity of strategies. Effectiveness of strategies. Peer reviewed. Flexible and extensible.

Table 6.2: Win conditions for stakeholders (Spiral 3)

shortlisting strategies, the application developers want them to be beneficial, with low overheads in the effort required. There should also be information available to allow the justification of results. Brokers and component developers are interested in how their components are assessed, and what information is required from the repository and when. Quality assurance needs to be able to track the process and have documentation of each step and decision. From the academic perspective, the strategies must be valid and effective. Peer review is important, as is the extensibility of the implementation, to allow for further research in the area.

6.2 Spiral 3 Context

The component selection task begins by defining the selection criteria. Most published selection approaches include a component model to describe the criteria or attributes to be used in the assessment. This is often implemented as a hierarchy to improve understandability. For example, the COTS characterisation model in Sassi et al (2003) groups the attributes into behavioural, architectural, quality of service, technical and usage categories. A discussion of component models is also given. Other schemes develop a custom hierarchy for the specific problem (Kontio, 1995, Ochs et al, 2009). The relative importance of the criteria must be determined if weightings are to be applied in the evaluation. These may be customised using a structured approach such as AHP (Saaty, 1990, Ochs et al, 2009). An assessment of each component against the criteria is then carried out, often as a manual process. Given the scores for all components on all criteria, a recommendation or ranking can then be determined. This often involves the aggregation

Stage	Output	CdCE Process
Specification	Criteria	Step 1
Customisation	Weights	Step 1
Screening	Candidates	Step 2
Assessment	Scores	Step 6
Comparison	Overall Scores	Step 7
Recommendation	Ranking	Step 8

Table 6.3: Generalised stages in component selection

of results using the WSM or the AHP (Saaty, 1990). In other cases, techniques such as Outranking are applied, as in Morisio and Tsoukiàs (1997).

The information gathering through the selection process is summarised in Table 6.3, including the mapping to generalised stages and to the CdCE Process. Shortlisting/screening can be seen as mini iteration of the overall selection process. The key difference is the number of products evaluated, and the depth of information used in the evaluation. Thus this chapter considers 'evaluation' techniques within screening/shortlisting.

In 2002, a paper by Ncube (Ncube and Dean, 2002) challenged the existing evaluation approaches used in component selection. Key issues were the use of WSM, AHP and similar tools. One criticism was in the aggregation of the scores across criteria (WSM) 'did it make sense to average across unrelated criteria?' For example, if the price of the software scored 4/10 and the user interface 20/30, averaging the scores would give 5.3333/10. By generating a number from the aggregation, much information was lost and the user may read more into the difference between results that is valid (is 5.3333/10 really much better than 5/10?). While the AHP was considered an improvement to WSM, it includes an assumption that all criteria are independent, which does not hold for component selection. Another issue with the AHP is the time required to evaluate the pairwise comparisons between each component on every criterion. Ruhe (2002) argued that automation was needed to reduce the expert's time required to evaluate components and support quality decision making.

Some selection research began to use AI techniques to address issues with assessing components, in particular the inherent problems with aggregating results. Neuro-fuzzy (Kuo et al, 1999) and Rough fuzzy sets (Rao and Sarma, 2003) have been used to deal with imprecision and uncertainty in component assessment, while overcoming some overheads of determining the original fuzzy sets. Most techniques are more applicable to

in-house repositories where the documentation of components can be standardised and detailed. For example, one paper described a repository with up to 1320 attributes for each component (Nakkrasae et al, 2004). This project is concerned with selecting third party components sourced from a range of repositories. Therefore there is a very large number of components to screen and information about them may be basic. This leads to an interest in AI to carry out both coarse screening and more in-depth analysis of the technical features of candidate components. The overheads for using the AI technique must be low as each selection process will have new requirements and is thus a new problem.

AI is a field that provides a range of techniques for representing and processing knowledge. When selecting an AI technique, it is important to consider the features that are needed, and which are more critical to the particular problem. In the component selection problem, the aim is to classify the components as being acceptable or rejected. Adjustable thresholds can be used to include or exclude more candidates, providing flexibility where criteria may have been too restrictive or lenient. When working with metadata from online repositories there may be incomplete data, so a tool with some tolerance for missing or uncertain data is preferred.

Feature	**Expert Systems**	**Fuzzy Systems**	**Artificial Neural Networks**	**Genetic Algorithms**	**C4.5**
Knowledge representation	+	++	–	-	+
Uncertainty tolerance	+	++	++	++	-
Imprecision tolerance	–	++	++	++	–
Adaptability	–	-	++	++	++
Learning ability	–	–	++	++	++
Explanation ability	++	++	–	-	++
Knowledge discovery and data mining	–	-	++	+	+
Maintainability	–	+	++	+	++

Table 6.4: Comparison of Traditional AI Techniques, adapted from Negnevitsky (2002). Techniques that deal well with a feature are indicated with + symbols, those which respond poorly have - symbols. ++ and – indicate stronger or weaker performance.

Tables 6.4 and 6.5 show how traditional and hybrid AI systems perform against eight criteria, all of which may be considered important for the automated selection

of components. For example, artificial neural networks (ANN) and genetic algorithms (Table 6.4) show strength (++) in dealing with uncertainty and imprecision. However they are not so strong on knowledge representation and explanation ability (-). Expert systems and fuzzy systems are stronger on knowledge representation and explanation, with weaknesses around adaptability and maintainability. The C4.5 classifier rates well in learning, providing an explanation and maintenance, and less well in uncertainty and imprecision. The hybrid approaches in Table 6.5 aim to address these weaknesses.

Feature	**Neural Expert Systems**	**Neuro-fuzzy Systems**	**Evol. Neural Networks**	**Fuzzy Evol. Systems**
Knowledge representation	+	++	–	++
Uncertainty tolerance	++	++	++	++
Imprecision tolerance	++	++	++	++
Adaptability	++	++	++	+
Learning ability	++	++	++	+
Explanation ability	++	++	–	++
Knowledge discovery and data mining	–	-	++	+
Maintainability	++	++	++	+

Table 6.5: Comparison of Hybrid AI Techniques (Maxville et al, 2004b)

Knowledge representation is important to component selection as it must utilise the metadata supplied by vendors and brokers, and communicate the results to users. As the information comes from diverse sources, there may be missing and uncertain data. While the selection criteria for components can be considered 'a match' or 'not a match', the facility to deal with imprecision may be more useful when looking at how well a description meets the user's needs (e.g. is the cost of the component close to the required cost).

An automated assessment is unlikely to be trusted unless it can explain the reasoning behind decisions. From Table 6.4, the traditional AI systems that perform well on explanation ability rate poorly on adaptability, learning and maintenance. This may lead to a trade off where the reasoning can be explained, but the expert must invest time to develop and tune rules - reducing the advantage of using AI. Neural-expert or neuro-fuzzy systems (Table 6.5) may overcome this, assuming data is available to train the neural network. An AI technique capable of knowledge discovery would be advantageous as it

may allow future data mining and self-update. Although not one of the criteria in the comparison tables, the ability to represent the interplay between attributes is another desirable feature. Common techniques for selecting components use weighted sums of scores against the selection criteria, losing the relationships between criteria and values. An expert system, fuzzy system or neural network would be capable of encoding these dependencies.

The perfect AI technique for component selection would ideally rate well in all of the above categories. Given the information in the tables, and the discussion, neural-expert systems and neuro-fuzzy systems may give the best balance between understandability (knowledge/explanation) and flexibility (adaptability/learning/maintenance). This investigation initially considered expert systems, moved on to a comparison of C4.5 and ANN and chose to use the C4.5 decision tree classifier. As can be seen in Table 6.4, C4.5 rates well in all features except uncertainty and imprecision tolerance. Exploration of the application of new technology, such as those in Table 6.5, to component selection has potential as future work for this line of research.

A deeper discussion of the exploration of approaches to shortlisting follows. Three sections consider the trials with: multicriteria assessment; expert systems and machine learning, followed by a section to discuss and reflect on the trials.

6.3 Approach 1: Multicriteria Assessment

Evaluation of components can be viewed as a Multi-Criteria Analysis (MCA) problem, forming a recommendation from a candidate's performance against a set of (weighted) criteria. This is intuitively what component selection involves, and is the premise for aggregation-based approaches. The terminology of MCA: criteria, alternatives, weights and scores, is given in Table 6.6. The relative importance of criteria are recorded as weights, impacting the influence of the criteria during aggregation of the results. A decision matrix (Table 6.7) is completed to record all results for each candidate for each criterion. This results in an m x n matrix, where there are m criteria and n candidates (alternatives). An overall score or ranking is calculated combining the score and weight for each criterion according to a formula.

As an example, a WSM method will calculate the overall score for a candidate by

multiplying the *Score* for each *Criterion* by the *Weight* and summing them together. This is the approach used in Alves and Castro (2001), calculated in Microsoft Excel. The formula for WSM is:

$$S_i = \sum_{j=1}^{n} s_{ij} w_j, \qquad i = 1,2,3...m.$$

Term	Description
Criteria (C_j)	The category being assessed, e.g. cost, development language.
Alternatives (A_i)	Alternatives being assessed, in our case the candidate components.
Weight (w_j)	A multiplier to increase or decrease the effect of individual criteria on the overall score.
Score (s_{ij})	The actual value given to a component against a particular criterion.

Table 6.6: Terminology

Criteria Weights Alternatives	**C_1 (w_1)**	**C_2 (w_2)**	**... ...**	**C_n (w_n)**
A_1	s_{11}	s_{12}	...	s_{1n}
A_2	s_{21}	s_{22}	...	s_{2n}
...	...	...	...	...
A_m	s_{m1}	s_{m2}	...	s_{mn}

Table 6.7: Decision Matrix

6.3.1 Customisation of the Selection Process

The selection process should be customised for each selection task by adjusting the weightings or priorities on each of the criteria, even if using a standardised specification. Weightings may be informed by intuition, previous experience, organisational guidelines, or apply a systematic approach such as the AHP (Saaty, 1990). Criteria weightings provide a mechanism to include the context of the selection task in the evaluation. For a given selection task, priority can be given to functionality, cost, risk factors through weightings on the related criteria. Emphasis and priority of criteria may vary between organisations, projects and components within a project.

The AHP approach to determining weightings requires a pairwise comparison of each of the criteria (m), creating an m x m matrix for all of the criteria being assessed. The pairwise comparison helps decision-makers to focus their attention on two criteria at a time, with the implication that this provides more accurate comparative weights (Saaty, 1990). The values used in the pairwise comparison come from a Scale of Relative Importance, with each criterion assessed relative to each of the other criteria. Saaty's scale is shown in Table 6.8, while Lootsma (1999) has an alternative, nonlinear scale. Table 6.9 gives an example of the Saaty or Judgement Matrix for calculating weights for criteria. Importance values are shown for C_1 and C_2. The Saaty Matrix is also used for generating values in the assessment stage.

Intensity of Importance	**Definition**
1	Equally important
3	Weakly more important
5	Essentially or strongly more important
7	Demonstrated importance
9	Absolutely more important
2,4,6,8	Intermediate values
Reciprocals	The counter-relationship is the reciprocal of the decided relationship between two alternatives.

Table 6.8: Saaty's Scale of Relative Importance

Criteria	**C_1**	**C_2**	**...**	**C_m**
C_1	1	p_{12} (5)	...	p_{1n}
C_2	p_{21} (1/5)	1	...	p_{2n}
...	...	...	1	...
C_m	p_{m1}	p_{m2}	...	1

Table 6.9: Saaty's Judgement Matrix: Generating Relative Weighting of Criteria

To provide a weighting for each criterion, we need to reduce the m x m matrix to a vector. Saaty's work advocates using the right principal eigenvector of the Judgement Matrix. It has also been approached as an error minimisation problem (Triantaphyllou, 1995). A simpler calculation of the weighting applies the normalised geometric mean as an approximation to finding the eigenvector. The AHP includes a Consistency Index: a calculation to give an indication of whether the scores are consistent within the matrix. A poor consistency result would indicate a revision of the importance values assigned to

each pair of criteria is required.

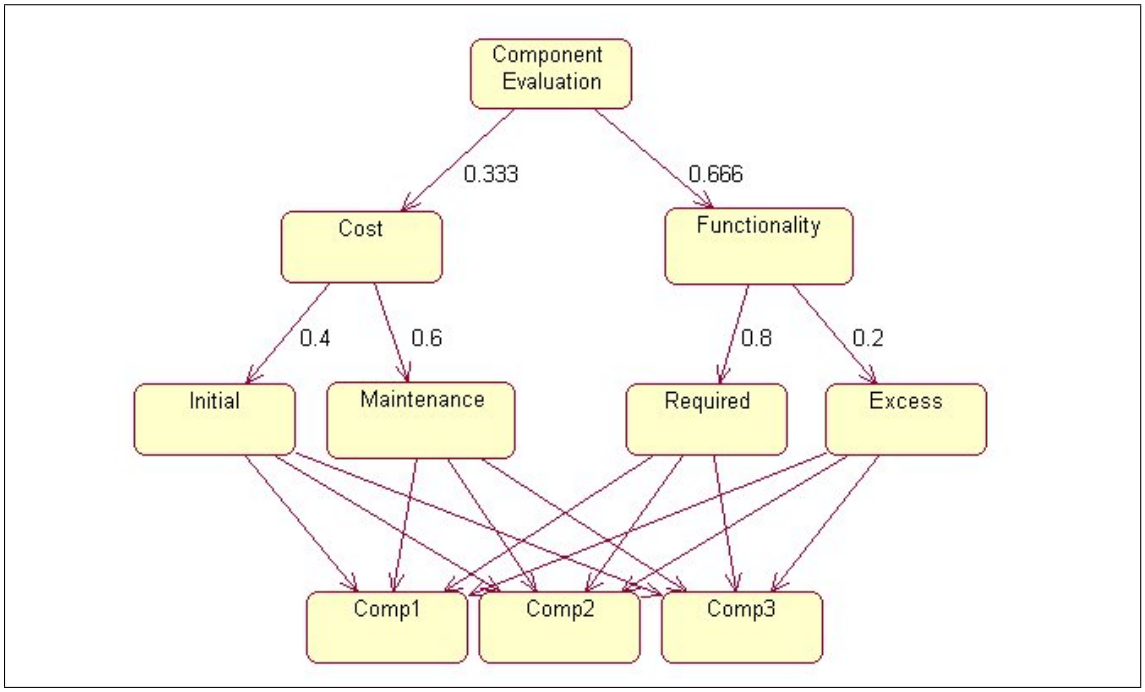


Figure 6.2: Example of Criteria Using the AHP

Another feature of the AHP is that it supports the organisation of criteria into a hierarchy (Figure 6.2). The weightings of sub-criteria must sum to one, with each subtree going through the same pairwise process as the top level criteria. In a real world situation, this may involve a large number of criteria on each level.

Determining weightings with the AHP approach may be a significant improvement over other techniques (or no technique). Alternatively, independent techniques may then be used for assessment and aggregation. In the COTS Acquisition Process (CAP) (Ochs et al, 2009) a 'Tailor and Weight Taxonomy' is used to organise the selection criteria, then the AHP is applied to customise the weightings on each criterion.

6.3.2 Assessment of Candidates

The assessment process requires a value for each candidate for each criterion. There are various ways of deciding on these values. One approach is the selection of a value from a scale or range. Guidelines can be provided to ensure consistency in grading between reviewers and across candidates. This would be the approach when using WSM.

An alternative is to apply the AHP approach, where a comparison is made between each pair of candidates for each of the criteria. The Judgement Matrix is used (as in

customising the weightings), but in this case each criterion has its own matrix, with candidates listed in the rows and columns.

An example calculation to assess Vendor Reliability is given in Table 6.10 for three components. The calculation of the geometric mean and normalisation is also shown. The result is a priority vector across the three candidates of (C_1=0.1963, C_2=0.6571, C_3=0.1466).

Vendor Reliability	**Component$_1$**	**Component$_2$**	**Component$_3$**
Component$_1$	1	1/5	2
Component$_2$	5	1	3
Component$_3$	1/2	1/3	1

*Component$_1$ = cube root (1 * 1/5 * 2) = 0.7368*
*Component$_2$ = cube root (5 * 1 * 3) = 2.4662*
*Component$_3$ = cube root (1/2 * 1/3 * 1) = 0.5503*
divide through by (0.7368 + 2.4662 + 0.5503 = 3.7533)
gives Priority vector
(C_1, C_2, C_3) = (0.1963, 0.6571, 0.1466)

Table 6.10: Saaty's Judgement Matrix: Assessing Candidates

Criteria Alternatives	**10**	**20**	**50**
2	55	210	1275
10	495	1090	3475
20	1945	3990	10725
50	12295	24690	62475
100	49545	99190	248725

Comparisons = n(n-1)/2 + n[m(m-1)/2]

Table 6.11: Pairwise Comparisons Required

Using pairwise comparisons helps to break a complex comparison into approachable small steps. There is a disadvantage as the number of comparisons increases quickly with larger numbers of criteria and alternatives. These comparisons are manual and often require teams or groups of people to each contribute their evaluations. If working with three alternatives and two criteria, six comparisons would be required; with ten alternatives and twenty criteria there are 1090 comparisons. In Table 6.11, the impact of the number of elements and criteria on the total number of comparisons is shown. Techniques for dealing with this blow-out include a duality approach (Triantaphyllou, 2001) or incomplete judgement matrices (Harker, 1987).

6.3.3 Comparison of Candidates

After all the scores/values have been collected, a comparison between components can be made. An aggregation-based approach will have a completed decision matrix containing the candidate components and their associated scores against each criterion. MCA techniques then combine all the data together to provide an overall score or ranking for each candidate.

The simplest MCA involves the direct analysis of the completed matrix to provide information about dominance of particular candidates. A dominant candidate performs at least as well as all others on all criteria, and better than the others on at least one criterion. Dominance analysis can be used to identify bias in criteria or to partition the candidates for more detailed analysis. More sophisticated MCA techniques use different approaches to evaluate data in the decision matrix. The main categories are: Multi-Attribute Utility Theory (MAUT); Outranking Methods; and, Mathematical Programming (Ncube and Dean, 2002).

MAUT techniques determine ordering of alternatives based on the decision-maker's preferences. They assume independence between criteria, which is not always true in component selection. MAUT includes Multi-Criteria Analysis Decision Making (MCDM), where an overall score is aggregated from criteria, weights and alternatives (e.g. WSM, AHP). Outranking techniques aim to find a subset or shortlist of candidates based on pairwise comparisons to determine the better performer. Concordance and discordance thresholds are defined and calculations across criteria for each candidate are used to determine if an alternative is in or out of the shortlist. Outranking is applied to software selection in Morisio and Tsoukiàs (1997). Mathematical programming techniques aim to identify alternatives that are closest to the ideal solution using some measure of distance. The measure takes into account the candidate's score on each criterion and the weighting attached to that criterion to create a function across all the criteria. Neubauer and Stummer (2007) use mathematical programming for multi-objective combinatorial optimisation, making it possible for users to interactively explore the solution space until they find the most appealing solution.

MCDM is the most commonly used technique in component selection, particularly the WSM and the AHP. As an exploration of the component selection process, WSM and AHP were trialled, with results in the following section. The alternative MAUT

techniques above were not trialled as they do not progress the CdCE Process towards automation or artificial intelligence.

6.3.4 Applying WSM and AHP

This investigation within Spiral 3 considers the WSM and AHP and variants. As the common approaches found in the literature, they were initially considered for the CdCE Process. Criticism of their applicability (e.g. Ncube and Dean (2002)) and experience of their application influenced the move to find alternative approaches.

The WSM is an intuitive and straight-forward approach, where values are combined by multiplying the weight for each criterion by the component's score and the sum of the weighted values becomes the overall result. To validly use this approach, the scores should be numerical, comparative and expressed in the same unit. In its favour is the simplicity of the calculation. Limitations of the WSM are that it does not assist the user in determining the weighting of the metrics and the criteria are treated as independent (Kontio, 1995). An additional theoretical issue is the summing of values with varying units of measurement. It is possible, but not correct, to sum scores in units of cost ($), maturity (years), and reliability (scale 1-10), for example. The assessment methodology can account for these differences by converting to a common scale (Solberg and Dahl, 2001).

The AHP deals well with qualitative data and provides facility for creating a hierarchy of criteria. It includes support for the determination of criteria and weightings, as well as the aggregation of the completed matrix. The approach is based on pairwise comparisons: firstly to determine the weightings for the criteria, then to provide values for the candidates against the criteria. The procedure for generating the weights and values is described in Sections 6.3.1 and 6.3.2. The overall calculation in AHP is similar to the WSM, with an additional calculation to normalise the columns in the decision matrix.

Issues with the AHP include the growth of the number of comparisons required for determining scores and the 'rank-reversal' problem. Comparisons increase quickly as the number of candidates and criteria increase. The rank-reversal problem can occur when the candidates (alternatives) under consideration are changed. In some situations, a new candidate can reverse the rankings of two unrelated candidates (Triantaphyllou, 2001).

Triantaphyllou (2001) suggests the Multiplicative AHP as a solution to this issue, using the following equation:

$$S_i = \prod_{j=1}^{n} \left(s_{ij}\right)^{w_j}, \qquad i = 1,2,3...m.$$

The WSM, the AHP and the Multiplicative AHP have been applied to an existing data set (selecting calculator software) to investigate the effort required and comparative results. Figure 6.3 provides a chart of the aggregated results from the second case study, including raw scores (before the weights were applied). The techniques used for each bar on the chart (in order) were:

- WSM without weights
- WSM with weights from the AHP
- AHP without weights
- AHP standard
- AHP using multiplicative formula.

It was found that there was a large effort involved when determining the raw AHP scores for each of the criteria, and the individual comparisons were less intuitive than allocating a value from an appropriate scale. The weights determined using the AHP were applied in a WSM calculation for comparison. An interesting result was that the weighting for cost was less than 3%, making it the least important criterion in this case study (i.e. cost would be sacrificed ahead of all other criteria). The true importance of cost was identified by going through the AHP process and comparing cost against each of the other attributes. The customisation process provided a way of making an objective decision. The comparison of attributes such as cost versus functionality or cost versus operating system showed that cost was consistently less important than the other attributes. Intuition and business decisions may allocate a much higher importance to price/cost.

Although there is variation in the scores in Figure 6.3 from each of the techniques, there was a similar overall ordering. The standard AHP, WSM with AHP weights and Multiplicative AHP gave almost the same rankings, with the middle candidates differing in standard AHP (C5 and C9 interchanged). This would indicate that the WSM may be adequate if using an aggregation approach, assuming a well-defined method of scoring,

and that scaling and weights are sufficient. Using the AHP for determining weights for a WSM approach may be a good compromise. The AHP has considerable overhead in terms of time. If an organisation is prepared to follow the method and devote expert time, it is the preferred aggregation option.

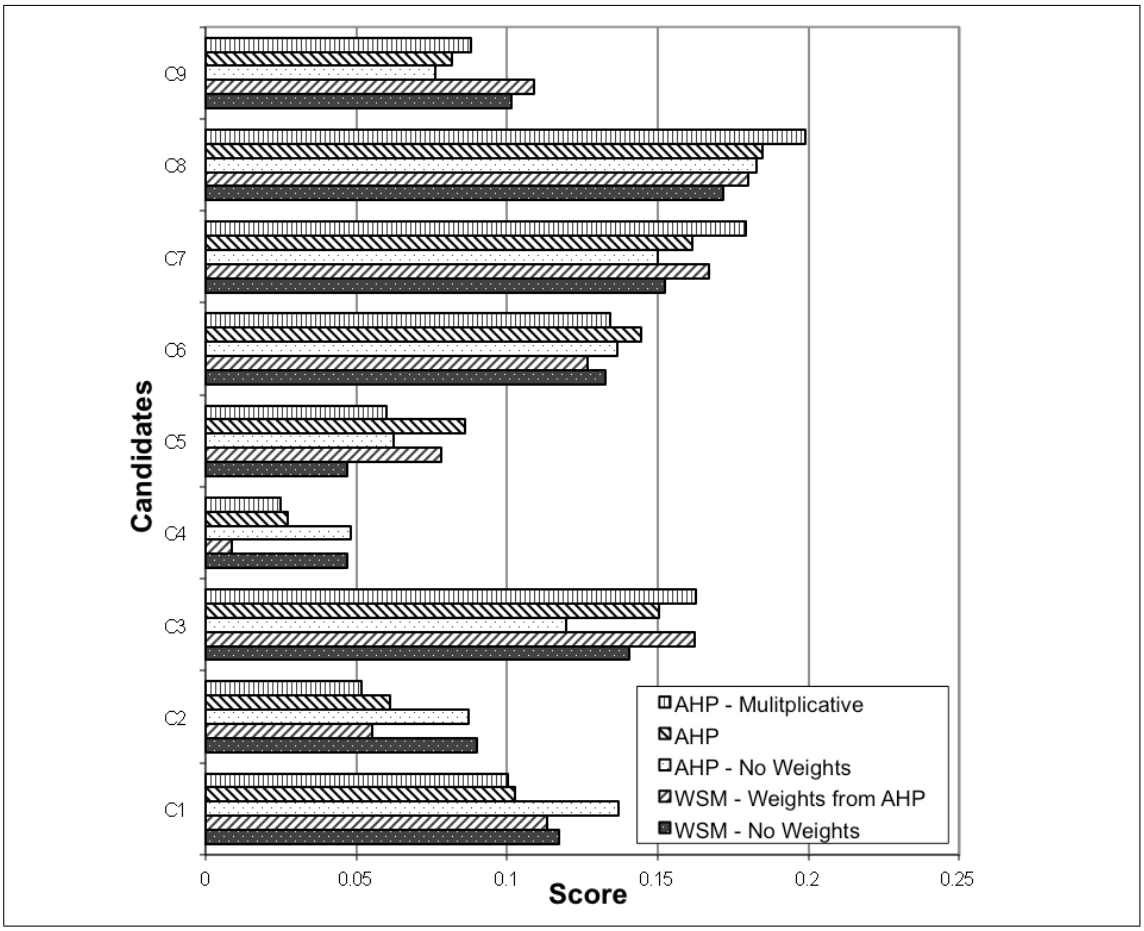


Figure 6.3: Comparison of Aggregation Techniques (Normalised)

6.3.5 Multicriteria Assessment Observations

This comparison of approaches finds no clear recommendation for aggregation-based techniques. The AHP provides a method for both determining weights and assessing candidates, and deals well with qualitative and quantitative data. However, it also requires considerable effort to carry out pairwise comparisons of each component on each criterion. The WSM is criticised for not providing a method for assessing weights. It is also often misused to aggregate data of varying types, resulting in values with little meaning, and potential to bias and mislead the user. To use the WSM safely, there should

be: guidelines for scoring; the conversion of scores to a uniform scale; and a reasoned approach to the allocation of weights. The AHP approach to customising weights can be utilised with the WSM. The final decision is made on the basis of the amount of time and expertise the organisation has available. For a quick decision with low risk, the WSM may be adequate. In a more critical situation it would be justified to use the AHP.

Less common techniques, such as outranking and mathematical programming address some of the issues with WSM and the AHP. However, they still work with a level of information loss where the scores against criteria are simple numbers. They also leave little room for the application of AI or knowledge-based techniques.

6.4 Approach 2: Expert Systems

The discussion of AI techniques in Table 6.4 highlighted the knowledge and explanation ability of expert systems. In response to some of the issues raised with multicriteria assessment approaches, the inclusion of knowledge, and the potential to clearly explain the logic of a decision was worthy of investigation.

An expert system is made up a series of rules which trigger under certain inputs, providing a path through the rules to any point in the decision-making process.

Similar to the previous approaches, an expert system approach requires customisation to the problem at hand. As a first step, the rules for accepting or rejecting a component must be determined, based on the selection criteria. These may be provided in a predetermined order, with consideration for any dependencies or priorities among criteria. These rules will then be applied to all candidates.

6.4.1 Exploration of an Expert System Approach

This work considered a simple selection problem, using the Jess expert system (Friedman-Hill, 2008). Jess provides a flexible interface that can be extended through Java programs, which has potential to be used for automating our process. The first stage assigned a mandatory or desirable classification to each of the attributes in the ideal specification (Description, Platform and Cost). A matrix was used to determine any dependencies between attributes (Figure 6.4). An example is that the cost factor for a component will be increased if the component does not match the required platform. This information

was used to create a flowchart for the screening and was translated into rules for Jess. Figure 6.5 gives an example of a flowchart with three attributes for assessing components. In this example, the observed values for a component are assigned letters as follows:

- Description: (A) Acceptable (N) Not Acceptable
- Platform: (M) Match (C) Close (N) No match
- Cost: (F) Free (R) Reasonable (H) High.

Case	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
Description	A	A	A	A	A	A	A	A	A	N	N	N	N	N	N	N	N	N
Platform	M	M	M	C	C	C	N	N	N	M	M	M	C	C	C	N	N	N
Cost	F	R	H	F	R	H	F	R	H	F	R	H	F	R	H	F	R	H
Accept		X	X		X													
Reject	X			X		X	X	X	X	X	X	X	X	X	X	X	X	X
Values:	**Description** (A) Acceptable, (N) Not Acceptable **Platform** (M) Match, (C) Close, (N) No Match **Cost** (F) Free, (R) Reasonable, (H) High																	

Figure 6.4: Decision table for Expert System Case Study

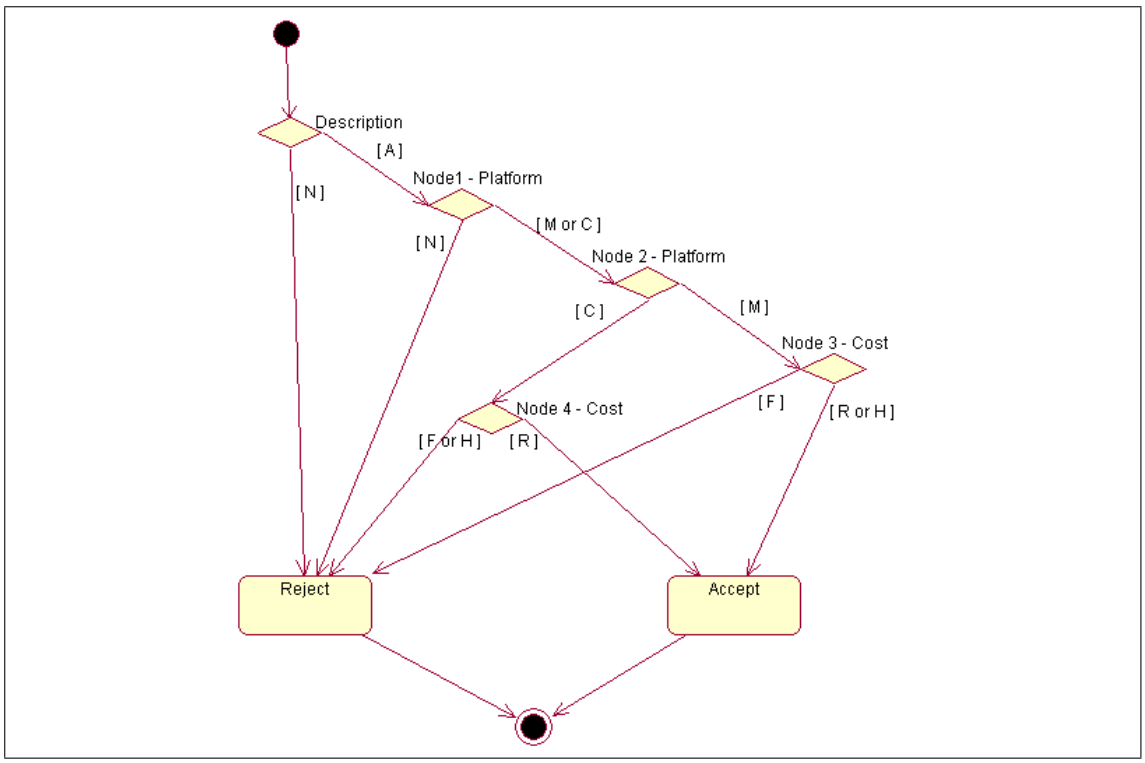


Figure 6.5: Example of reasoning for component selection

The description attribute is mandatory, while the platform and cost attributes do not require an exact match to the requirements. This gives priority to the component exhibiting a match on the description. The flowchart also shows the relationship between platform and cost, and in this case, free software is not acceptable. Generating rules is not a simple task, and would require an easy to use interface for recording priorities and dependencies from which to generate rules.

The sample output is generated from the implementation of the flowchart in Figure 6.5 in Jess. The output (Figure 6.6) shows rules being triggered to activate the nodes corresponding to the flowchart. The first example is accepted, while the second is rejected due to the cost. The reasoning for the decision is clearly outlined. If there were missing values, the Accept/Reject nodes would not be activated, and may be viewed as an intermediate result ('soft' reject).

```
Jess> (assert(candidate(description "A")
      (platform "C")(cost "R")))
<Fact-1>
Jess> (run)
Description acceptable - activate node 1
Platform acceptable - activate node 2
Platform acceptable - activate node 4
Cost acceptable - accept
CANDIDATE ACCEPTED
5
Jess> (reset)
TRUE
Jess> (assert(candidate(description "A")
      (platform "C")(cost "H")))
<Fact-1>
Jess> (run)
Description acceptable - activate node 1
Platform acceptable - activate node 2
Platform acceptable - activate node 4
Cost not acceptable - reject
CANDIDATE REJECTED
5
Jess>
```

Figure 6.6: Output of Jess expert system

The customisation stage of the expert system approach is responsible for the assessment of components. The base result is a simple accept/reject status. One measure that may be used for ranking results is to look at the reasoning for each candidate to get an indication of how many rules were triggered. For example, in Figure 6.6, a rejected candidate may have matched none of the rules, or it may have matched up to three of the rules. If three rules are matched, the candidate may be deemed closer to being accepted than one with two rules.

In more complex examples, there would be far more criteria and nodes on the flowchart. This may result in rules being triggered that do not directly take the candidate through to

being accepted (or rejected). For example, the shortest accept path may require six rules to be triggered. One accepted component may trigger more rules than another accepted component, which may indicate that more triggered rules implies a better match. Thus the comparison of accepted components could use the total or excess facts to generate a ranking. This implies that all rules are equal, which raises issues as that is unlikely to be the case.

The expert system approach gives the reasoning behind a selection or rejection, and can highlight issues in the rules themselves. The recommendation may state that better results could be returned if changes were made to the rules and the process rerun. For example, there could be missing data, or customisation may be too loose or too restrictive.

6.4.2 Expert System Observations

Expert systems were considered, where a rating or classification could be determined via a knowledge base of rules. The ability to capture the reasoning used when assessing components and provide reasoning for decisions is a valuable benefit of using expert systems: the mass of information used is not reduced to a number. The difficulty with expert systems is in determining the rules for assessment. In a component selection situation, each selection task will have different rules. The overhead of developing rules for each project may prohibit the use of expert systems. Where there may be repetition of the selection process, the time required for creation of rules for an expert system would pay off over time.

6.5 Approach 3: Machine Learning

Previous exploration of approaches were found wanting in terms of loss of knowledge and potential for automation. The expert system was able to take data about criteria in its native form, and apply rules suiting that data, showing the potential of an approach exhibiting knowledge representation and explanation ability. Unfortunately, the rules need to be manually created and cover all possibilities to produce meaningful results. Considering other approaches in Table 6.4 and the experience from the expert system, an improved result may come from a technique with learning ability and maintainability. These include ANN, genetic algorithms and C4.5 (decision tree classifiers). Another

lesson from using the expert system was that the selection task could be considered a classification problem - accept/reject. This led the investigation in the direction of ANN and C4.5. It also applied the concept of mandatory and preferred criteria, which was considered when loosening criteria in the case study in Spiral 2.

With both ANN and C4.5, a predictive model is used to classify the data. This model is the result of training the network or classifier on a real or created dataset. The predictive model can then be used on test data to assign a class to each instance in the dataset. If the test data is not part of the training dataset, it is considered 'unseen', which is preferred to avoid biased results. If the training data includes the class for each instance, this can be used for supervised learning, the approach used for this study.

To explore the potential of machine learning for selection of components, another exploratory case study was undertaken. The two machine learning techniques were available through the Weka application (Hall et al, 2009), which is used in the following case study.

6.5.1 Applying Machine Learning

The scenario for this case study is the selection of a component to provide scientific calculator functionality. As it is an exploratory case study, the selection criteria are simple. Four of the criteria are *mandatory* and six are *preferred*. Attributes not in those categories remain as the default priority *other*. Adjustable thresholds were assigned to require four out of four *mandatory* and three out of six of the *preferred* criteria for a component to be accepted. The ideal component specification is given in Figure 6.7.

Training Data

When working with classifiers, data is required to train the predictive model. In some scenarios, this would be historical data. As each selection task is quite different in terms of criteria, history from the previous tasks are not helpful in this case. Instead, training data is used to capture the characteristics of interest. The approach of using the ideal specification to create the training data developed from this perspective.

The two classifiers were trained on the same data, generated from the ideal specification. The data generator was developed to create a data distribution that captures the complexity of the criteria used in the assessment, while avoiding an internal bias.

```
<?xml version="1.0"?>
<Description xmlns="http://www.scis.ecu.edu.au/swvML/1.0/"
      xmlns:dc="http://purl.org/dc/elements/1.0/"
      xmlns:swv="http://www.scis.ecu.edu.au/swvML/1.0/" >
  <dc:description type="mandatory">scientific</dc:description>
  <dc:description>calculator</dc:description>
  <swv:devStatus type="mandatory">mature</swv:devStatus>
  <swv:licence type="preferred">GPL</swv:licence>
  <swv:price type="preferred" min="25" max="50">40</swv:price>
  <swv:technical>
    <swv:devLanguage type="preferred">Java</swv:devLanguage>
    <swv:devLanguage>C++</swv:devLanguage>
    <swv:operatingSystem type="mandatory">Linux
    </swv:operatingSystem>
    <swv:systemRequirements>
      <swv:memory type="preferred" min="15" max="50">20
      </swv:memory>
      <swv:diskSpace type="preferred" min="30" max="50">40
      </swv:diskSpace>
    </swv:systemRequirements>
  </swv:technical>
</Description>
```

Figure 6.7: Ideal Component Specification

More detail on the training data generation is in Section 6.7.1. Each training dataset is validated using Weka's implementation of 10-fold cross-validation, with those used in this case study scoring over 96%. The data generator application automates the labelling of the data into output classes, allowing the use of supervised learning techniques to train the classifiers[1].

Early tests with data generation showed that the classifier was highly sensitive to the distribution of the data[2]. If too high a proportion of the instances were rejected (e.g. 82%), then the classifier created an overpruned model that rejected all instances. The neural network did not oversimplify the classification, but also saw performance improvements with the regenerated training data.

The data is generated using a technique similar to boundary value analysis in test case generation (Myers, 1979). The focus is on including discriminating values close to the border between acceptance and rejection. Incremental experiments provided useful feedback on the distribution of data required for the classifiers to learn patterns with the greatest (or least) (such as identifying *mandatory* and *other*) attributes. Validation and

[1]Supervised learning relies on a manual labelling of the training data for the system to learn the patterns for each classification. Unsupervised learning works with the patterns formed within the training data and attempts to group them into clusters. Data falling inside a cluster can them be labelled according the closest cluster.

[2]An alternative approach to dealing with imbalance in the distribution of data is the use of cost matrices. This would have the side effect of lowering the overall performance of the classifier and is therefore not appropriate for this work.

training results indicated that this approach gives adequate results, but has potential for improvement.

C4.5

The most widely used decision tree classifier is the C4.5 algorithm (Quinlan, 1993). C4.5 generates a tree incorporating all instances within the dataset and their associated classifications. The algorithm groups the data to allow 'pruning' to create a smaller, more manageable decision tree. When working with the generated training sets, the trees were overpruned as the input data matched the real world distribution of data (a low percentage of acceptable instances) and the tree became a simple 'reject all'. Improvements to the training dataset were rewarded with a larger and more detailed tree. The decision tree for the case study dataset has 147 nodes, including 74 leaves (classification points).

Table 6.12 presents the results for the C4.5 classifier against the case study training and test sets (both unseen). The results for the training dataset included twelve misclassifications out of 2736 (below 0.5% error rate). These errors were all in situations where the mandatory requirements were met and the assessment of the preferable criteria failed. Additional improvements to the generator algorithm may be developed to resolve these problems and move closer to 100% correct classification.

Dataset	**% Correctly Classified**	**Total Instances**
Training	99.5614%	2736
10-fold Cross-validation	96.3085%	2736
Unseen Dataset 1	100%	96
Unseen Dataset 2	98.1183%	744

Table 6.12: C4.5 Performance

Unseen data (not used for training) was then processed to further evaluate the classifier. The datasets represent all combinations of attribute values that are acceptable (Unseen Dataset 1) and those that should be rejected (Unseen Dataset 2). Attributes that do not affect the decision (*other*) are randomised. Both datasets were classified with an acceptable level of accuracy. Unseen data will generally have more classification errors than the training data. Dataset 1 performed better than the training due to the

distribution of the values and the size of the dataset. The training sets are focussed on the combinations close to acceptance/rejection, whereas many of the combinations in the unseen datasets were clearly in one of these classes (e.g. reject with no matches of mandatory attributes).

Artificial Neural Network

The concept of a neural network classifier is to simulate the neurons of the human brain, organised into interconnected layers. In Weka's implementation, a backwards propagation algorithm updates the weights that connect the neurons and reinforce those that result in a correct classification.

Many parameters are available to tune an ANN. An empirical approach was used to explore the effect of the parameters on the classification of the case study data. The default parameters were: 500 epochs; a learning rate of 0.3; the momentum value of 0.2; and a network with 18 nodes in the hidden layer. Sensitivity analysis indicated that changing the number of epochs had little effect on the classification: the network had converged before 250 epochs. Applying a learning rate of 0.1 gave poor results with the test sets, while each of 0.2, 0.3 and 0.4 had similar results. Increasing the momentum produced a downward trend in performance, although all of the training results were above 98%.

There are an infinite number of possible configurations of an ANN. A network with two nodes in a hidden layer is sufficient for most problems (Abbass and Sarker, 2001). This was used as the baseline and a variety of configurations was explored. Sensitivity analysis between sample data and the network configuration indicated that a network with eighteen nodes in a single hidden layer was suitable. This provided similar performance to a ten node hidden layer and was over 10% better at classifying the training data than a two node configuration. There is potential for further exploration of the configuration as networks with two hidden layers also performed well (10,5 nodes and 10,2 nodes) and may be more suitable when working with more complex selection tasks.

Table 6.13 gives results for the eighteen node, single hidden layer ANN. The model correctly classified over 99.5% of the instances in the dataset. Investigation of the misclassified instances showed that they were on the boundaries of the ranges for numeric values. For example, a value of $14 for price may have been classed as 'accept', when it

should strictly have been rejected as being below $15. All errors found in classification resulted from similar borderline cases. Modification of the data generator algorithm may reduce these errors. The classification of unseen datasets produced good results, with potential for improvement.

Dataset	**% Correctly Classified**	**Total Instances**
Training	99.5249%	2736
10-fold Cross-validation	96.6009%	2736
Unseen Dataset 1	90.625%	96
Unseen Dataset 2	90.3274%	744

Table 6.13: Neural Network Performance

Interplay

One of the issues in aggregation-based approaches is the assumed independence of criteria. This is seen as a limitation, with a high likelihood that at least some criteria will be related in component selection tasks. An investigation of the potential for supporting the representation of interplay was undertaken in the machine learning experiments.

In a simple example of interplay, three interrelated attributes were considered: development language, framework and operating system. The scenario is that an organisation has expertise in Java, C++ and C#, ActiveX, Enterprise JavaBeans (EJB) and acceptable platforms are Windows and Linux. Certain combinations of the three attributes will be preferred, for example, (Java, EJB, Linux) and (C#, ActiveX, Windows). Combinations to avoid would include ('all', ActiveX, Linux) and (C#, EJB, 'all'). In a more realistic situation, a particular stack of software may be supported, or there may be three supported stacks. In this case, the software dependencies would need all of the stack, with unsupported mixed stacks/version being undesirable.

The training data for this scenario is given an output class ranging from 0 (not acceptable) to 5 (recommended). Using this data to generate C4.5 and ANN classifiers resulted in 100% correct classification of the data for both techniques. 10-fold cross-validation on the data was also 100%. The resulting C4.5 decision tree (Figure 6.8) shows the reasoning used by the classifier to allocate a 'score' to each component based on the attribute values. The different treatment of the attribute in each branch of the

tree is the key to the correct assessment of the attribute interplay. In an aggregation based approach, a component with conflicting attribute values would still score highly as each attribute is independently valid. For example (Java, EJB, Linux) would score (5,5,5)=15/15 as would the inadvisable (C#, EJB, Linux). Weightings on the attributes could not differentiate these results.

```
devLanguage = Java
|   operatingSystem = Linux
|   |   framework = EJB: 5
|   |   framework = ActiveX: 0
|   operatingSystem = Windows: 5
devLanguage = C++
|   framework = EJB: 0
|   framework = ActiveX
|   |   operatingSystem = Linux: 0
|   |   operatingSystem = Windows: 3
devLanguage = C#
|   operatingSystem = Linux: 0
|   operatingSystem = Windows
|   |   framework = EJB: 0
|   |   framework = ActiveX: 5
```

Figure 6.8: Decision tree for interplay dataset

6.5.2 Machine Learning Observations

This work developed a technique for training machine learning classifiers in selecting software components for development projects. The training data is generated from an ideal specification of the required component, using an XML Schema as a generalised template. Using the XML Schema and the instance document for the ideal component, the data generator creates an internal model of the component. The training data is automatically labelled into classes, overcoming one of the difficulties with supervised learning. Case study results for both classifiers gave a high degree of accuracy in identifying suitable components. A similar percentage of components were classified incorrectly (by ANN and C4.5) during training, although there was no overlap in the instances that caused confusion. Improvements to the generator algorithm should correct most, if not all of these classification problems. Both classifiers performed well when classifying unseen data (over 98% and 90%). This indicates the classifiers are able to correctly identify suitable components with high accuracy, based on the ideal specification of the component. Results of over 96% in 10-fold cross-validation of the training data gives confidence that

the data itself did not bias the training of the classifiers.

The investigation into the generation of classifiers to recognise attribute interplay were also successful. A small study was carried out to ensure that the classifiers are capable of dealing with this more complex combination of data. Both C4.5 and ANN correctly classified all instances in the data.

6.6 Evaluation of Approaches

At this point, five approaches to assessing components had been investigated. These were considered in terms of the desire for automation, and the inclusion of knowledge-based and intelligent techniques. Aggregation and multicriteria analysis were considered valuable in some scenarios, however they did not match the direction that was required for this project. The range of AI approaches was considered. An expert system approach showed potential, particularly in the representation of knowledge and understandability of results. Unfortunately, the effort involved in the development and maintenance of rules for the expert system made it unsuitable for this investigation. A change of focus to learning ability and ease of maintenance led to machine learning techniques for classifying data. An approach for generating training data was developed, and allowed the trial of two classifiers, C4.5 and an ANN. Both gave good results when classifying unseen data, with little difference in accuracy. The decision tree produced by C4.5 can provide reasoning for decisions that are made, allowing confidence and trust in the recommendations. For this reason, C4.5 was chosen as the approach for the remainder of the investigation.

6.7 Spiral 3 Implementation

The implementation of the C4.5 decision tree classifier approach to component selection extends the exploratory work described in Section 6.5. The approach begins with the ideal specification, with additional information on the mandatory/preferred status of the attributes and the thresholds on the number of preferred required for an acceptable candidate. This is used to generate training and test data using a new program, Intelligent. The training data is used with Weka and C4.5 (J48 implementation) to create a predictive model of the selection, to match the ideal specification. This model can then be

used on the repository data to classify the components and generate a shortlist. The ideal specification, the training data and the predictive model are all artefacts that help document the shortlisting process and can be reused if repeating the selection task.

More detail on the implementation of the shortlisting approach is provided in the following sections.

6.7.1 Generation of Training Data

Inputs	Outputs
ideal component specification (CdCE XML) thresholds for attribute priority	training data set (ARFF) test data set(s) (ARFF)

Table 6.14: Generation of training data - inputs and outputs

The first step towards classifying the components is to create a training dataset for the classifier. This takes the ideal specification and the thresholds as inputs (Table 6.14). The output from the generation is training and test data sets.

The training data generator (Intelligent) output is determined by the number of attributes and which combinations of values will be acceptable, based on the ideal specification. The attributes are classified as being 'mandatory', 'preferred' or 'other'. Mandatory attributes must all be met for the candidate to be accepted. The user provides a threshold for each group to indicate the proportion of those attributes that must be matched. For example, the threshold on mandatory would be 1.0 (all required) and for preferred it may be 0.5 (at least half of the attribute values must be satisfied). The 'other' attributes do not affect the assessment. This provides three equivalence classes for the data generation. Test generation uses equivalence classes to reduce the number of test cases by having one value represent the whole class of values. For training the system, equivalence classes are used to enumerate the combinations of attribute values inside and between classes, and the corresponding classification for that component.

The generation of the training data is based on permuting the attributes through all possible values. At this stage, the attributes can be matched or not matched, represented by 'Y' and 'N'. The permutations are made by grouping the attributes by priority (mandatory/preferred). The thresholds (0-1) are converted to be the number of matching attributes required for that priority group. An example would be that 3 of 3 mandatory attributes must be matched and 2 from 4 of the preferred. The generator then cycles

through all acceptable permutations of attribute values, attaching the label 'accept' as the 'result' class. The same cycling is done for all permutations that should be rejected. For the previous example, with seven attributes, the 23 permutations generated are shown in Figure 6.9. To go through all permutations would need 128 instances.

```
Y,Y,Y,Y,Y,Y,Y,accept
Y,Y,Y,N,Y,Y,Y,accept
Y,Y,Y,Y,N,Y,Y,accept
Y,Y,Y,Y,Y,N,Y,accept
Y,Y,Y,Y,Y,Y,N,accept
Y,Y,Y,N,N,Y,Y,accept
Y,Y,Y,N,Y,N,Y,accept
Y,Y,Y,N,Y,Y,N,accept
Y,Y,Y,Y,N,N,Y,accept
Y,Y,Y,Y,N,Y,N,accept
Y,Y,Y,Y,Y,N,N,accept
Y,Y,Y,N,N,N,Y,reject
Y,Y,Y,N,N,Y,N,reject
Y,Y,Y,N,Y,N,N,reject
Y,Y,Y,Y,N,N,N,reject
Y,Y,Y,N,N,N,N,reject
N,Y,Y,Y,Y,Y,Y,reject
Y,N,Y,Y,Y,Y,Y,reject
Y,Y,N,Y,Y,Y,Y,reject
N,N,Y,Y,Y,Y,Y,reject
N,Y,N,Y,Y,Y,Y,reject
Y,N,N,Y,Y,Y,Y,reject
N,N,N,Y,Y,Y,Y,reject
```

Figure 6.9: Generator permutations

For supervised learning, the last attribute denotes the classification of the entry, in this case result=accept/reject. The generated training data is grouped into lessons. The initial lessons focus on acceptable attribute values, then the values leading to rejection. Parameters on the generation can adjust the number and size of lessons. The lessons focus on the patterns of attribute values that are near the border of acceptable/unacceptable. Random selection of training data would almost certainly result in all candidates being classified as rejected. Our solution is to apply Boundary Value Analysis (BVA) techniques. Training data is selected that sits close to the boundary between acceptance and rejection, along with some more straight-forward entries. This has prevented the classifier from over-simplifying its decision tree and allows us to work with relatively small training sets. In a small example, the optimisation of the training set seems insignificant, however with 37 attributes, the number of permutations with just two values (Y/N) would be 2^{37} = 137,438,953,472.

6.7.2 Training the Classifier

Inputs	Outputs
training data set (ARFF) test data set (ARFF) classifier type and parameters	predictive model

Table 6.15: Training the classifier - inputs and outputs

Given the training and test datasets (see Table 6.7.2), it is possible to run Weka to generate the predictive model for the classification. The inputs are the training and test data sets, along with the parameters for classifier type and options (these are detailed in Table 7.11). The output from the training is the predictive model.

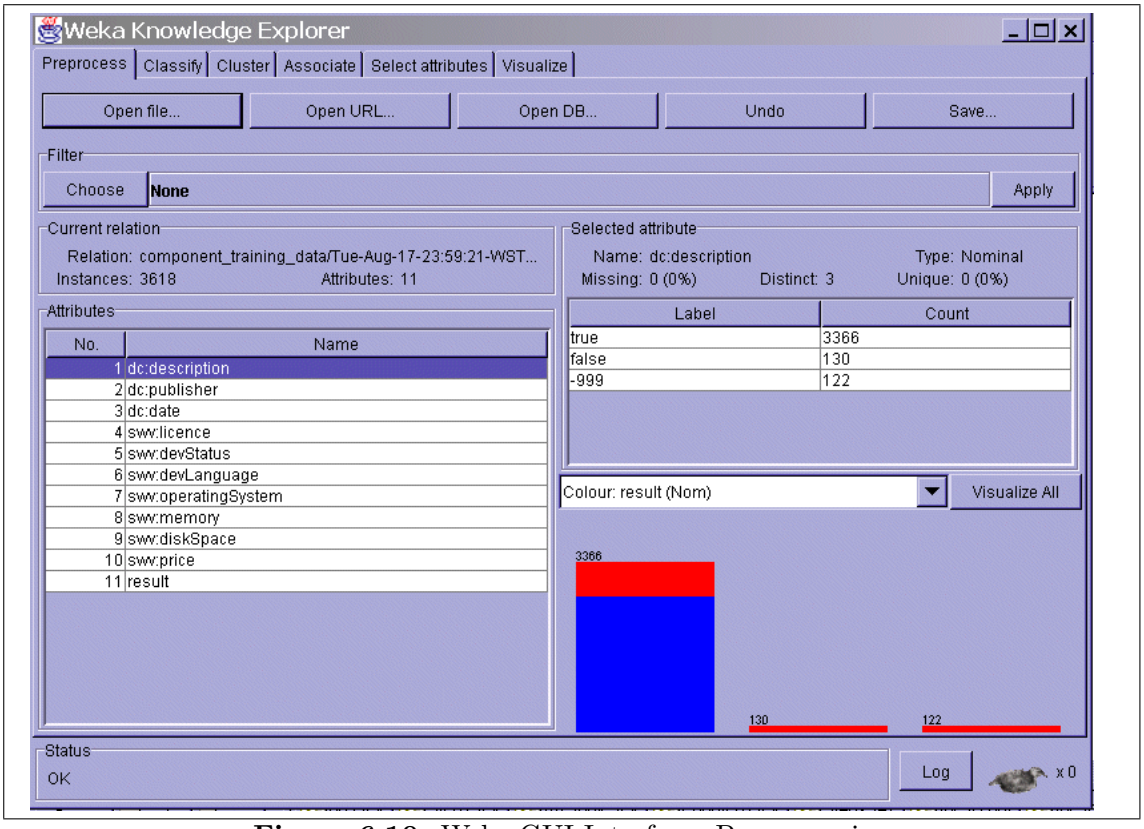


Figure 6.10: Weka GUI Interface: Preprocessing

When using the GUI interface to Weka, the Explorer tool is selected. This starts up the Weka Knowledge Explorer which provides access to tools for classification, clustering and association rules. The first tab is for preprocessing the data (see Figure 6.10)[3]. This is where the training data file is loaded. During the load, the headers, attribute types and

[3]Screenshots are from an older version of Weka as the GUI has not been required since these initial investigations

file structure are checked. Clicking on any of the attributes brings up a histogram of the number of values in each category, with colours representing the different output classes. This functionality helps in exploring the entire dataset for issues such as widespread missing data on a particular field.

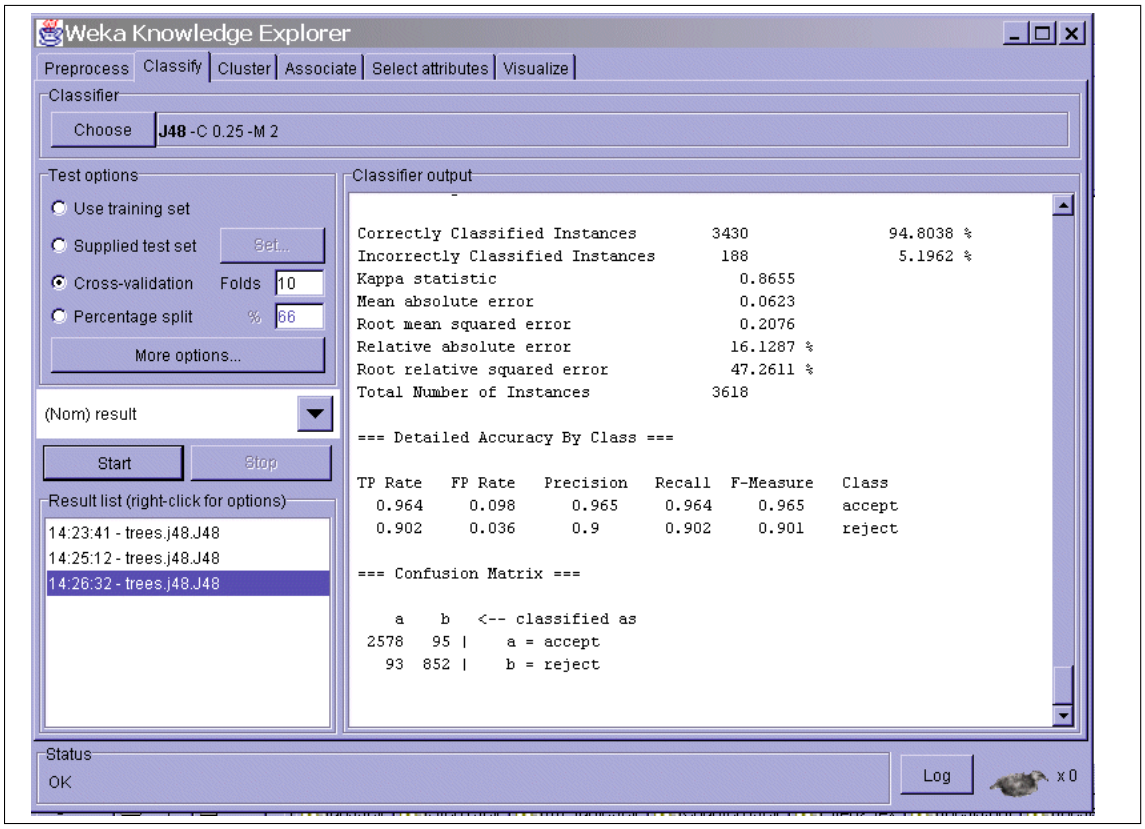


Figure 6.11: Weka GUI Interface: Classify

The Classify tab (Figure 6.11) is used to select a classifier, in this case J48 (for a C4.5 clone), input parameters (default: -C 0.25 -M 2) and whether to use cross validation. Input parameter C relates to the confidence factor on pruning the tree: the default of 0.25 is used. Parameter M sets the minimum number of objects in a leaf node, where 2 is the default. Default parameters were found to give the required results, and are not changed. The results of the classification are shown in the Classifier Output window. A shortened version of this output is given in Figures 6.12 and 6.13.

Reading through the output (Figure 6.12), the first items listed are the classifier used and the parameters. It then shows the `relation`, a description of the data taken from the first line of the ARFF file. The number of instances (data points) is given, followed by the number of attributes and their labels. In this case the mode is 10-fold cross

```
=== Run information ===

Scheme:       weka.classifiers.trees.j48.J48 -C 0.25 -M 2
Relation:     component_training_data/Tue-Aug-17-23:59:21-WST-2004
Instances:    3618
Attributes:   11
              dc:description
              dc:publisher
              dc:date
              swv:licence
              swv:devStatus
              swv:devLanguage
              swv:operatingSystem
              swv:memory
              swv:diskSpace
              swv:price
              result
Test mode:    10-fold cross-validation
```

Figure 6.12: Results of training the classifier (Part 1/2)

validation, as selected in Figure 6.11.

Cross validation is a method for predicting the fit of a model by splitting the sample into training and validation data. The model is created based on the training data and then tested on unseen data for validation. Using unseen data helps identify overfitting to the training set. Stratified cross validation endeavours to select sets of data to preserve the mean response value (or to have a similar distribution of nominal attributes). Taking this a step further, 10-fold cross validation splits the data into ten subsamples. In the first pass, nine of these are used for training the model and the other is used for validation. This is rotated through all the subsamples. The validation results are combined to provide the overall statistics. 10-fold cross validation is widely used (Witten et al, 2011) and the stratified form is used in all training in this project.

With processing complete, the model is available, in this case the pruned C4.5 tree generated by the classifier for the data (Figure 6.13, shortened to fit page). The accuracy of the model is assessed by running the three data sets through the model: the training data and two unseen sets - Test1 and Test2. Test1 includes all 'acceptable' permutations of attribute values as data points, while Test2 is made up of 'reject' permutations. This allowed for clearer diagnosis of issues in the training data generation and the subsequent predictive model. The remainder of this Section uses the training data set for illustration.

```
=== Classifier model (full training set) ===

J48 pruned tree
------------------
swv:operatingSystem = true
|   dc:description = true
|   |   swv:devLanguage = true
|   |   |   swv:devStatus = true
<--- snip --->
|   dc:description = false: reject (42.0)
|   dc:description = -999: reject (42.0)
swv:operatingSystem = false: reject (134.0)
swv:operatingSystem = -999: reject (118.0)

Number of Leaves  :  151
Size of the tree :  226

Time taken to build model: 0.24 seconds

=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances        3430               94.8038 %
Incorrectly Classified Instances       188                5.1962 %
Kappa statistic                          0.8655
Mean absolute error                      0.0623
Root mean squared error                  0.2076
Relative absolute error                 16.1287 %
Root relative squared error             47.2611 %
Total Number of Instances             3618

=== Detailed Accuracy By Class ===
TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.964     0.098      0.965     0.964     0.965    accept
  0.902     0.036      0.9       0.902     0.901    reject

=== Confusion Matrix ===
    a    b   <-- classified as
 2578   95 |    a = accept
   93  852 |    b = reject
```

Figure 6.13: Results of training the classifier (Part 2/2)

The classifier in this example is assigning data points to one of two groups: a = accept, b = reject. Correctly accepted values are True Positives (TP) and correctly rejected are True Negatives (TN). The aim is to minimise incorrectly accepted False Positives (FP) and incorrectly rejected False Negatives (FN). In relation to the confusion matrix from Figure 6.13, these map to the following values:

```
=== Confusion Matrix ===
     a               b   <-- classified as
 TP = 2578          FN =  95  |    a = accept
 FP =   93          TN = 852  |    b = reject
```

These results indicate a poor performance in classification, although it shows 94% of instances were correctly classified under stratified cross-validation. The number of False Negatives and False Positives shows the impact of unclear classification, which may be addressed through improvements in the data representation.

6.7.3 Classifying the Data

Inputs	**Outputs**
real data set (ARFF) predictive model classifier type and parameters	classified real data

Table 6.16: Classifying the data - inputs and outputs

Previous steps have provided a predictive model as input to this final stage, to be used with Weka, C4.5 (J48) and a set of parameters (see Table 6.16). The repository data then needs to be transformed into the format required by Weka to match the training and test data sets. At this time, a manual transformation was carried out to create an ARFF file with the matching number and type of attributes. The class for all entries was set to 'reject'. When the predictive model was run against this repository dataset, any acceptable candidates show up as their class does not match what is in the input file. These instances can then be matched to the source component, which is added to the shortlist.

6.8 Spiral 3 Results

In this case study a manual selection exercise is revisited, updating it to use machine learning. The scenario for the case study is the selection of a software component to provide scientific calculator functionality. The attributes used in the case study are shown in Table 6.17. Potential component information was taken from four online sites. These had been assessed previously with a manual application of the CdCE Process, summarised in Table 6.18.

Attribute	**Type**	**Importance**	**Values**
Description	Multi-String	Mandatory	Scientific Calculator
Development Status	String	Mandatory	Mature
Licence	String	Preferred	GPL
Price	Numeric	Preferred	$0-$75
Development Language	Multi-String	Preferred	Java/C++
Operating System	Multi-String	Mandatory	Linux
Memory	Numeric	Preferred	5-70Mb
Disk Space	Numeric	Preferred	10-90Mb

Table 6.17: Case study ideal specification

Site	**Number of Entries**	**Number of Candidates**
I	8,000+	1
II	12,000+	7
III	36,000+	4
IV	30,000+	0
Total	**86,000+**	**9** (3 duplicates)

Table 6.18: Case study manual assessment

For this study, three of the attributes are mandatory and five are preferred, with the remaining attributes categorised as other. A threshold of 0.5 was selected, which rounds down to two out of five preferred attributes for acceptance. Thus acceptable instances will have a match (Y) on all three mandatory attributes, and match between two and five of the preferred attributes. If the threshold was 0.6, between three and five preferred attributes would need to be matched.

It is then input into Weka's implementation of the C4.5 classifier which outputs a decision tree-based predictive model. Weka also provides an analysis of the resulting tree's performance against the training and test data. The derived decision tree matched the model of the candidate selection criteria and when applied to the training data, correctly classified 100% entries. Another test of the classifier was run against simulated data and correctly classified all the components and selected 27 out of 2000 components

as potential candidates.

The predictive model was applied to real component data where it identified 17 suitable components for the 578 that were considered. Although the four repositories offered over 86,000 entries, a subset of those matching the search criterion 'calculator' was used as manual conversion of all entries to XML was impractical. Incorrect results were given for less than 7% of the data, in situations where values for attributes were missing. Classification involving instances with missing values is one of the limitations of C4.5. If it has not seen a particular value for an attribute, it will still try to classify the instance according to its decision tree, with unpredictable results. In our data, missing information was replaced with '-' for text attributes and -1 or 1000 for numeric attributes. There is more that can be done to address missing data, including the substitution of average or default values for missing values, which is future work for this project.

At this point, the user can consider updating or tuning the ideal specification. Using the facilities provided by Weka, it is possible to look at the component data as individual attributes or as groups of attributes. Statistical information about individual attributes helps to adjust ranges for numeric values. Clustering tools help us to find components that have a similar profile to our ideal specification. The ideal specification may then be adjusted, the classifier retrained for a new predictive model, and the component data reclassified to get a tighter match on suitable components.

6.9 Spiral 3 Evaluation

Spiral 3 investigated strategies to support shortlisting as part of **RE3**. Approaches trialled included WSM, AHP, expert systems, ANN and C4.5. On consideration of each approach, it was decided to work with C4.5. Further experiments were able to show the effectiveness of using the classifier, along with indications of what future work was required to improve performance in terms of relevance and recall.

The Spiral can be evaluated with respect to the stakeholder Win conditions. For the application developer, all strategies had benefits. However for the purposes of this project, utilising a C4.5 classifier is considered the most beneficial in that it produces a decision tree which can be used to justify results. This is also useful for quality assurance. The classifier has been used through the Weka GUI, which also can be run from the command

line to simplify interaction. This use of Weka and utilising the ideal specification for inputs of requirements information results in low overheads for the user.

Component developers and brokers would be able to see what information is required for their components to compete for selection, and can see how the selection process is implemented. Academic stakeholders can consider the trials and discussion of approaches in determining their validity and effectiveness. The work in Spiral 3 was reviewed through publication at SEKE (Maxville et al, 2004b), COMPSAC (Maxville et al, 2004c) and the Postgraduate Electrical Engineering & Computing Symposium (PEECS) (Maxville et al, 2004a). The use of five techniques for shortlisting in this Spiral supports the premise of flexibility in the CdCE Process. While each technique could have had further extensions, in this investigation it is C4.5 that will be further explored. From the discussion above, Win conditions have been satisfied for Spiral 3.

6.9.1 Spiral 3 Goals

The following discussion refers to the goals listed in Table 6.19.

Quality

The shortlisting task is described step by step in Section 7.4. Each stage has well-defined inputs and outputs.

Given the requirements as inputs for Step 2, the process can be repeated, manually or through re-running the tools provided. This was a key aim for the shortlisting as it is likely to be revisited for iteration within a selection task or during system evolution.

Usability

The inputs and outputs for the shortlisting are easy to understand. One of the reasons that C4.5 was chosen was for its human readable decision tree. Building the training data and transforming the repository are a bit more complex, but do not need to be understood in detail to use the shortlisting process. One of the reasons that the user does not need deep understanding of the classification is the provision of scripts. Automation is provided in the processing of the data and the scripts for running the various utilities. These input and output XML files, which are the interfaces between the tools, as well as providing much of the documentation of the process.

SPIRAL 3	**Purpose** **Issue** **Object** **Context**	**Evaluate** **effectiveness of** **strategies for shortlisting** **Spiral 3**	**Results**
Goal 3A	Focus	**Quality**: *Produce a structured, repeatable process for shortlisting components*	
	Viewpoint	Quality Assurance personnel	
	Q3A1	Is the shortlisting task well defined?	YES
	Q3A3	Is the process repeatable?	YES
Goal 3B	Focus	**Usability**: *The shortlisting process must be automated and understandable*	
	Viewpoint	Application developer	
	Q3B1	Is the shortlisting easy for the user to understand?	YES
	Q3B2	Does the shortlisting include automation?	YES
	Q3B3	Has the work been tested on real world examples?	YES
Goal 3C	Focus	**Intelligence**: *Apply intelligent techniques and strategies to shortlisting*	
	Viewpoint	Application developer	
	Q3C1	Have intelligent strategies been utilised?	YES
Goal 3D	Focus	**Innovation**: *Consider a wide range of options for shortlisting to find a novel solution*	
	Viewpoint	Academia	
	Q3D1	Were a range of options considered?	YES
	Q3D2	Have innovations been developed?	YES
Goal 3E	Focus	**Dynamics**: *Allow for change and iteration*	
	Viewpoint	Application developer	
	Q3E1	Does the shortlisting allow for iteration?	YES
	Q3E2	Does the shortlisting allow for change?	YES
Goal 3F	Focus	**Reuse**: *Where possible make use of existing code and artefacts*	
	Viewpoint	Application developer	
	Q3F1	Has the work reused external resources?	YES

Table 6.19: GQM Summary - Spiral 3

The initial data was collected manually, harvested from real world repository data. Once the tools and approaches pass development tests on small datasets, small case studies are used to evaluate and proof work.

Intelligence

Artificial intelligence techniques have been considered the focus of this Spiral. These included the Jess expert system, artificial neural networks (ANN) and the C4.5 classifier. The C4.5 machine learning classifier was considered the best fit for this approach to component selection, with the training data for the classifier generated from the ideal specification.

Innovation

The search for an intelligent solution began with a literature review of potential techniques (Tables 6.4 and 6.5). The initial choice was to use an expert system, investigated through the Jess tool. Results were acceptable, but there was an issue in the manual creation of rules. The approachability of rules was still preferred, particularly when other solution were bringing back the issue of aggregation. Generation of the training data from the ideal specification opened up options and C4.5 and ANN were trialled, with C4.5 being selected for its decision tree as compared to the ANN black box.

The innovations are in the use of C4.5 in generating a predictive model for component selection, and in the generation of training data from the ideal specification to allow supervised learning.

Dynamics

The shortlisting supports iteration through the automation of selection, giving faster results across a larger set of candidates. The application developer can consider the results and quickly explore options by adjusting the criteria.

Changes in the requirements can be accommodated easily in a similar way to iteration. It is also possible to select different options in Weka, or adapt the training data for another tool.

Reuse

New reuse in this Spiral was the Jess Expert System and the Weka machine learning environment. Scripts were written to support the work, but there was little application development for this Spiral.

6.10 Spiral 3 Review and Planning

This Spiral is the first of four which address Research Element 3: strategies for the evaluation of software components and how an intelligent approach can be taken. Spiral 3 considered various approaches, including the AHP. The key points were to have an automated, repeatable, justifiable approach that retained the information and detail of

the software under evaluation. Contributions of the work are in the novel use of a classifier and the mechanism for training the classifier for component selection.

The discussion in this Chapter documents a key decision in the project - the use of C4.5 classifier for shortlisting. This is a novel approach and the fourth contribution of the research (C4). Although it was considered the preferred technique for subsequent investigation, some issues have been identified. The first is that the current implementation loses much of the information in the ideal specification and is providing the equivalent of a simple database query (match/no match). This could be enhanced through better data representation. Another issue has been in the high rate of missing data in some fields in online repositories. These have created some unpredictable results, which indicates that better handling of missing data is required. A third enhancement would be greater automation for the experiments and, in turn, the selection process as a whole. At the end of Spiral 3, the plan for Spiral 4 is to improve data representation, missing data handling and to increase automation.

6.11 Post-Spiral Updates

The work in Spiral 3 is most impacted by Spirals 4 and 6 (see Figure 7.26). In Spiral 4, changes are made to the data representation, affecting the attributes and their handling. This, in turn, requires changes to the training (and test) data generator. Also in Spiral 4, the repository data was sourced from freshmeat as an RDF file, then converted into CdCE format (swvML schema). Using real world data creates issues with incomplete entries and specific handling of missing data has been implemented. The Spiral 4 version of the shortlisting has been automated via scripts and uses the command line version of Weka.

In Spiral 6 (Chapter 9) this automation led to the development of the ClassifierSuite. The suite is a decision support tool, providing a graphical view of the process of selecting attributes for the shortlist. This impacts the selection by taking the mandatory and preferred criteria and showing the user all permutations of loosening the selection set. The user can then consider 16, 32 or more different sets (and resulting shortlists) instead of being limited to a few.

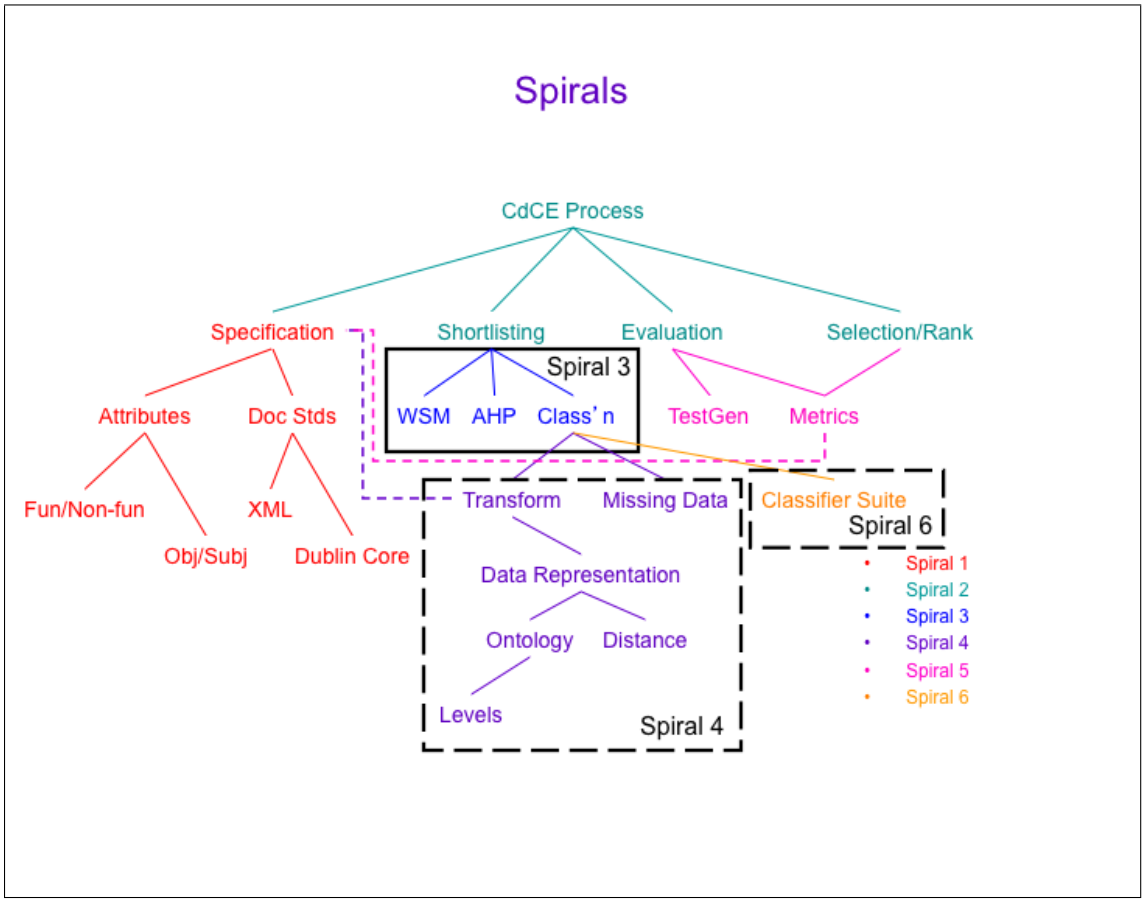


Figure 6.14: Spiral 4 and later updates made to the Spiral outcomes

6.12 Summary

This chapter has presented the work of Spiral 3, focusing on techniques for shortlisting as part of **RE3**. The goals for the Spiral were to implement a usable, effective shortlisting approach which could provide documentation to justify the selections made. Within the Spiral, a number of techniques were explored including WSM, AHP, expert systems, ANN and C4.5. The decision was made to choose C4.5 as the approach to selection for the CdCE Process.

The contribution of this Spiral, and its later updates, is a novel shortlisting approach. The approach takes the ideal specification and uses it to generate training data for the C4.5 classifier. The resulting predictive model is used to classify components from a repository to create a shortlist. The shortlist is then used for the rest of the CdCE Process. C4.5 provides a decision tree which gives a human-understandable justification of the selections that were made. Evaluation of shortlisting with C4.5 identified areas

which could be improved: data representation, missing data handling and automation. The commitment was made to address these issues in Spiral 4.

The following chapter presents the work of Spiral 4. This continues the exploration of **RE3** and the approach to shortlisting in the CdCE Process.

Chapter 7

Data Representation

This Chapter presents the work of Spiral 4, concentrating on enhancing the data representation used for the shortlisting. As part of the overall investigation, Spiral 4 is targeting **RE3**, strategies to assist component selection. The Spiral extends and refines the approach taken in Spiral 3, where a machine learning classifier is used for filtering candidate components.

The main aim of the Spiral is to improve the data representation for the selection process. A range of transformations is used to take advantage of specific characteristics of the base attribute types. A suite of tools and procedures has been developed to provide automation of this part of the CdCE Process. The result is improved relevance and recall in the shortlisted candidates. The contribution of this Spiral is the enhanced data representation, integrated with the Process and supported by tools.

The goals for Spiral 4 are noted in Table 7.1. The primary goal of this Spiral is enhanced data representation (**quality**). To make this usable, there will be tools and an underlying knowledge base (**usability**). The **intelligence** goal is to be realised through the use of ontologies and knowledge management, combined with the classifier approach from Spiral 3. Existing code and artefacts will be used where possible (**reuse**), and

SPIRAL 4	GOALS
Quality	Enhance shortlisting for more accurate results
Usability	Provide tools and knowledge base for users
Intelligence	Apply ontologies and knowledge management to shortlisting
Innovation	Include innovative knowledge management and missing data treatment
Dynamics	Allow for update and substitution of knowledge base
Reuse	Where possible make use of existing code and artefacts

Table 7.1: Goals for Spiral 4

be easily updated and interchanged (**dynamics**). The solutions developed are expected to include **innovation** in the combination of classification and knowledge management. These are revisited in the evaluation at the end of the Chapter.

7.1 Spiral 4 Overview

The problem being approached in this Spiral is to improve results and be able to get more out of the data via knowledge management techniques. Following successful work with the classifier, this Spiral deals with extending and refining the shortlisting strategies. A requirement of this work is to take the raw data from component and COTS repositories and use them in Step 2 of the CdCE selection process.

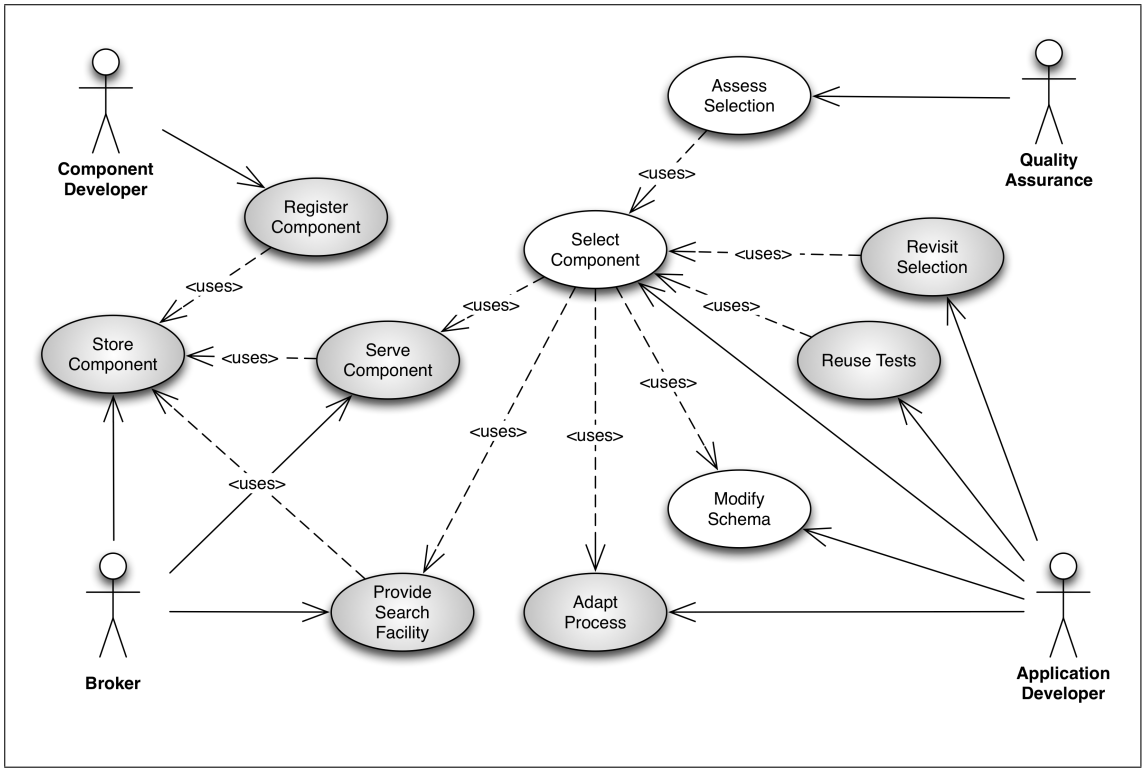


Figure 7.1: Use cases for component selection, the focus of Spiral 4 (those not in the scope for this Spiral are greyed)

There are two actors involved in Spiral 4: application developers and quality assurers (Figure 7.1). The use cases of interest are **Select Component**, **Assess Selection** and **Modify Schema**. Application developers will make use of the data representation through the ideal specification and on any parameters or settings required for using the related tools. They would also assess the external repositories to utilise in the selection

Stakeholder	Win Conditions
Application Developers	Strategies are beneficial. Justifiable results. Low overhead to use strategies. More than database queries.
Component Developers	Know how their component is assessed and compared.
Component Brokers	How to integrate with the repository. What information is required and when. May affect their templates and use of ontologies.
Quality Assurance	Documentation of results and decision-making process.
Academia	Choice of data representations. Handling of data. Peer reviewed. Flexible and extensible.

Table 7.2: Win conditions for stakeholders (Spiral 4)

task. In an assessment of a completed selection process, quality assurers will need to understand the impact of the data representation and the options provided. If a change is made to the underlying schema, the application developer needs to consider the impact and how to ensure it fits with the data representation approach adopted.

The stakeholder Win conditions for this Spiral are listed in Table 7.2. Application developers will need to see that the strategies can be beneficial to them, and enhance the process. They will also need to see the difference between these techniques and standard database queries. Quality assurers need to know that the knowledge management techniques are providing the correct information and the results can be justified. Component developers and brokers will be interested in how the assessment will be impacted by the enhanced data representation. It may affect how and where they list their software in repositories. From the academic perspective, the handling of the data and the choice of data representation must be valid. Peer review is again important, as is the extensibility of the implementation.

7.2 Spiral 4 Context

The first step is to use the ideal specification to generate training data, then use this data to train the classifier. As the datasets are created, a variety of data transformations is available for representing the values for each attribute.

The shortlisting of candidates can draw on literature for similar tasks and also on the broader fields of searching, classification and knowledge management. In many areas

an individual is in a situation of trying to find items matching a set of criteria. Perhaps the most common applications of shortlisting are web search engines. In these general-purpose search engines, unstructured text documents are matched/ranked according to search criteria. Algorithms for text searching continue to be improved; however at a basic level the relevance of a document is rated by whether all the search terms appear and by counting the number of occurrences of each term (Sparck Jones, 1972, Wu et al, 2008). Engines, such as Google, combine the relevance of a page (based on its internal content) with 'pagerank', an indicator of the importance of a page determined by the number of inward links (Page et al, 1999). The indexing and searching of text pre-dates search engines, therefore Information Management and Library Science techniques such as distance calculations (Lee et al, 1993), stopwords (Luhn, 1960), word stemming (Lovins, 1968), classification (Sparck Jones, 1970) and ontologies (Masterman, 1957) are also applicable.

There are other ways to represent data than unstructured text. In particular, the component metadata in this project includes dates and numeric values, along with terms that have inherent relationships and structure. In these situations, classification schemes and ontologies can be applied to encode semantic relationships and can enhance matching. Large ontologies exist for research areas, such as the Protein Ontology[1], as well as organisational ones such as freshmeat troves.

Another source of theory for search and selection are the related fields of data mining and machine learning. Machine learning is a sub-area of AI which aims to find and classify patterns in text and images. Data mining techniques are often applied in machine learning for discovering patterns in large datasets, an increasing issue as the amount of stored data increases (Benoît, 2002).

Moving from text searching to machine learning and data mining, issues arise with the representation of data. It is possible to transform data between types (Table 7.3) and to refine the representation of the 'match' between required and observed values. This assumes expertise and often an intuition of where patterns will lie. It also benefits from attribute analysis to cull attributes that are unlikely to have a bearing on the result and could cloud the results. Encoding can create new attributes or a sequence of values or events can be encoded in one attribute.

[1]Protein Ontology Project: http://proteinontology.org.au/

Input Data	Output Data
Numeric	Scaled numeric Nominal
Date	Numeric
Nominal	Numeric
Text (short)	Nominal (Boolean) Numeric Nominal Text (short)
Text (long)	Surrogate Auxiliary

Table 7.3: Data transformations

In the specific area of component and COTS selection, basic filtering is supplied by repositories to search the metadata for each item. These vary from a flat search of all fields, to some which allow the fixing of an attribute to a value (freshmeat, 2007). Component Source categorises their components into domains and values for the attributes. This provides the equivalent of an SQL query on a number of fields and a Boolean result of the comparison.

In most cases the components are given a score against each of the selection criteria and weights applied to each before a weighted sum is calculated. This was discussed in Section 6.3. Central in that discussion are the limitations of using WSM and AHP for component selection (Ncube and Dean, 2002), a key driver for the exploration of alternative evaluation approaches in this study.

7.3 Spiral 4 Approach to Data Representation

Based on the work in Spiral 3, the approach is to take the ideal specification and generate training data to create a predictive model for shortlisting. The repository data is transformed to CdCE format for compatibility. Another transformation of the repository information takes place using the ideal specification, creating data ready for classification. The classifier is then used on the transformed data, resulting in each instance being assigned to a class (accept/reject) which provides the shortlist of candidates.

The CdCE data model (swvML) is based on the Spiral 1 investigation of characterisation: describing resources; useful information for component selection; and, common information provided by repositories. As discussed in Chapter 4, the CdCE data model

is described in an XML schema with instances of the schema able to describe single or multiple components. While the CdCE Process and tools have been developed around this data model, the option to use a different model has been maintained throughout the project. This allows an organisation to substitute their own data model.

Each repository also has its own data model and format for providing data. For example, freshmeat has 38 attributes in a DTD with the data supplied as RDF files. SourceForge provides a copy of its relational database or free text summary files. To be able to automatically process COTS information it needs to be in a form that can be interpreted with consistency. Field mappings and an ontology are used to transform data from the various sources into the CdCE data model. The freshmeat repository is built on an ontology 'trove' and this has been used as a starting point for the CdCE ontology.

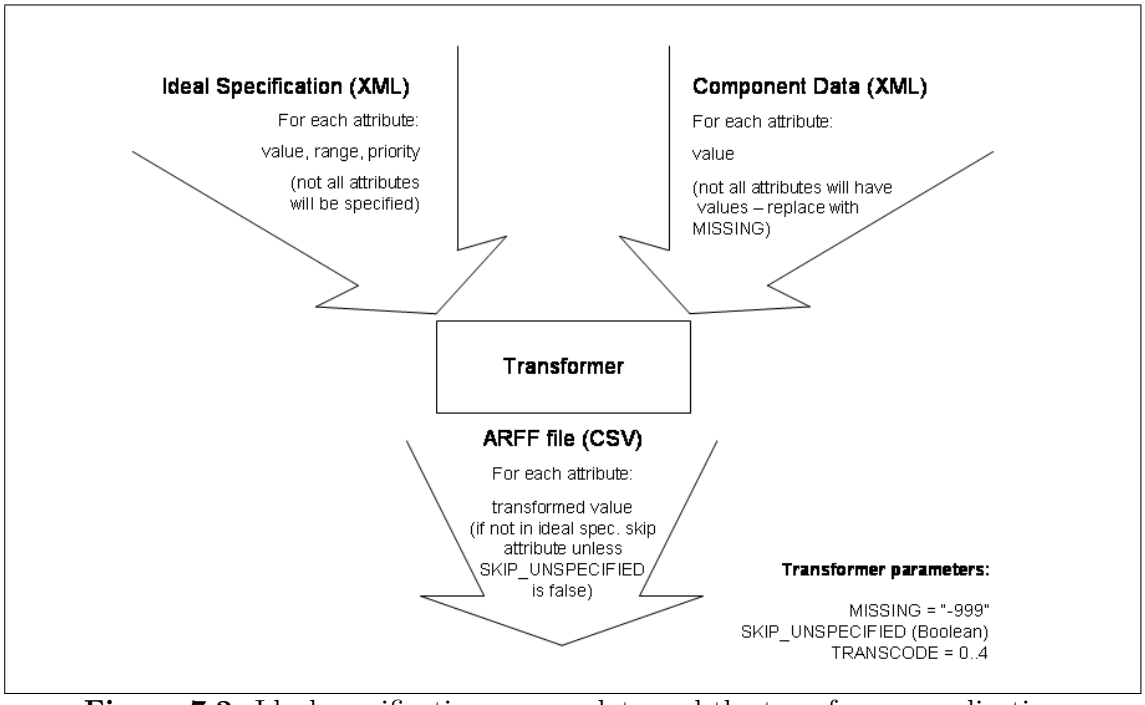


Figure 7.2: Ideal specification, source data and the transformer application

Figure 7.2 shows the contributions of the ideal specification, source data, transformer and the resulting ARFF output ready for classification using the Weka Data Mining Software. The ideal specification supplies the required values and the priority level for each attributes. The component data is provided in an XML file which will include the values for each attribute for each of the components. The transformer application takes these two inputs and creates an ARFF file. Each line in the file corresponds to a particular component, with comma separated entries to represent each attribute value

on that component.

The conversion process has been implemented as a series of filters using the SAX[2] parser. XSLT would have been appropriate for a simple transformation; however SAX allows more complex conversions, including the use of ontologies to map data values. Another alternative would have been to use a parser to build a DOM tree of XML documents. The reasons for choosing to use the SAX parser were:

- Ability to work with very large files, e.g. the freshmeat data file is over 118Mb
- Simple implementation of transformation between XML documents
- The ability to input and output from various file formats and databases using the same program structure as for XML files
- Simplification of repetitive processing using UNIX pipes and filters and scripts.

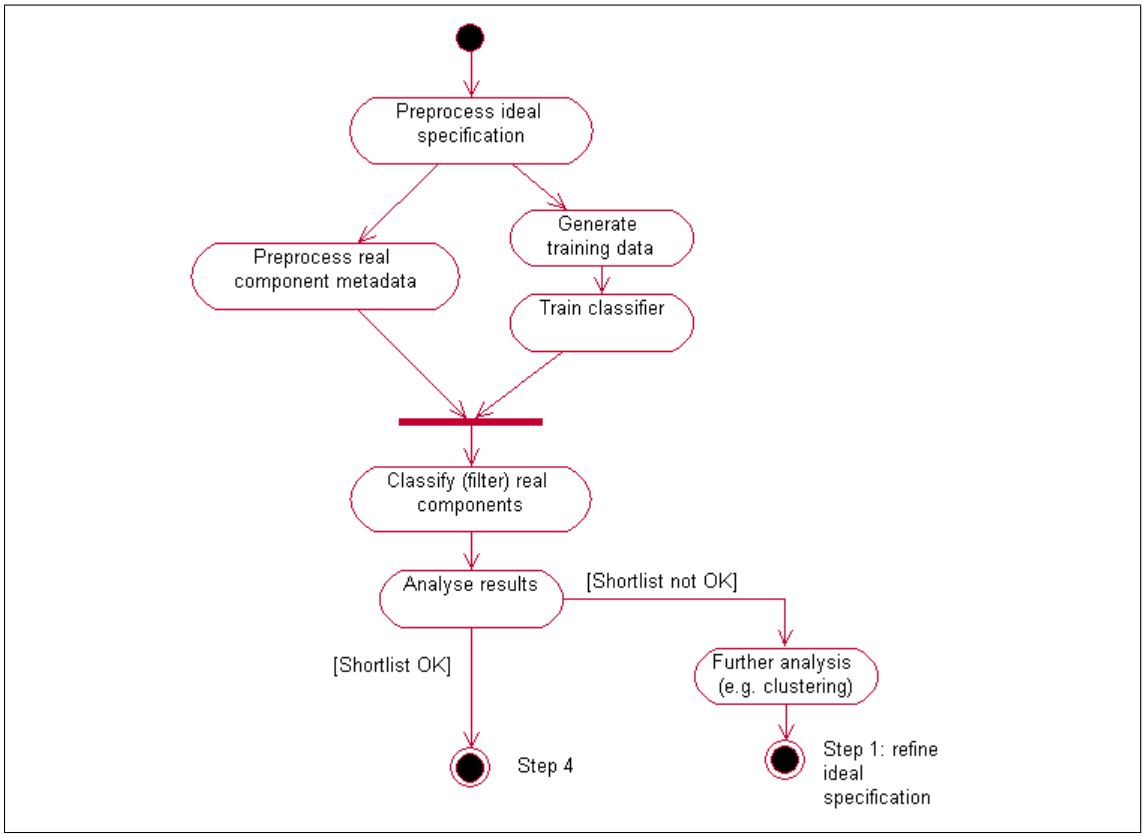


Figure 7.3: Activity diagram for Step 2

The activity diagram in Figure 7.3 shows the approach taken to generating training data and to converting the repository data. On the left branch, the repository data

[2]http://download.oracle.com/javase/1.4.2/docs/api/javax/xml/parsers/SAXParser.html

is converted into CdCE format. This includes working with a thesaurus (part of the ontology) to standardise the terminology used. In experiments the repository data has been held in CdCE format, however in practice it would need to be refreshed to give a new snapshot of the available components. The right branch shows the generation of training data from the ideal specification, which is then used to train the classifier.

The second part of the processing takes the CdCE version of the data and converts it with the same transformation as that used for the training data. The transformed data is run through the trained classifier, resulting in a shortlist of components. This list is then analysed to determine if the number of candidates is in the right range, and if they are a good match to requirements. If the list is acceptable, the user moves to Step 4 of the process, otherwise the ideal specification can be refined by iterating to Step 1. Chapter 8 describes the ClassifierSuite tool to assist the selection of criteria for shortlisting.

7.3.1 Data Model

The choice of attributes for component selection has been discussed and defined in Chapter 4. When implementing the data processing for the component information, more specific information was needed on the types of data held in the attributes. To give maximum information the conversion to CdCE format retains as much of the raw information as possible. For example, the long text description of the component is kept intact and a number of transformations suited to textual data are provided. With other text fields, such as operating systems, there are clear semantic relationships between terms (e.g. Java and J2ME, Windows and Win2000) which may form a hierarchy . These are classified as ontology fields and their transformation can make use of a knowledge base of related terms.

The CdCE data model has attributes of varying types, given in Table 7.4. There are `numeric` attributes for the price, memory required and disk space required. A `date` attribute is used for the release date. Attributes including descriptive text are categorised as `longText` and are brought into the CdCE data model unchanged. These attributes include the description, detail and technical description. Other text attributes include terms that represent values, or have relationships between values (also includes enumerated numeric attributes), referred to as `ontology` attributes. In these cases an extendable

ontology is used to hold all values and map them to the CdCE representation. The ontology is also used to compare values in the selection process. The third group of text attributes has more variance and little scope for processing or comparison. These are `freeText` attributes and are stored unchanged.

Attribute	Base type	Attribute	Base type
Title	FreeText	Rel_Id	FreeText
Version	FreeText	Rel_Value	FreeText
Creator	FreeText	Rights	FreeText
Subject	Ontology	Licence	Ontology
Description	LongText	Demo	FreeText
Detail	LongText	Documentation	FreeText
Publisher	FreeText	sourceCode	Ontology
Support	FreeText	SupportLevel	FreeText
devStatus	Ontology	Zspec	LongText
Date	Date	techDescription	LongText
Type	Ontology	devLanguage	Ontology
Format	Ontology	Framework	Ontology
Identifier	FreeText	Standard	Ontology
Source	FreeText	operatingSystem	Ontology
Language	Ontology	Platform	Ontology
Relation	FreeText	Processor	Ontology
Rel_Type	Ontology	Memory	Numeric
Rel_Source	FreeText	diskSpace	Numeric
Rel_Version	FreeText	Price	Numeric

Table 7.4: CdCE Data Model

Ideal Specification

The ideal specification is read in after the list of attributes is created as a component template. A sample ideal specification for a game renderer and/or browser is shown in Figure 7.4. The ideal specification includes values for each required attribute and indicates the attribute's priority (mandatory, preferred or other). `numeric` and `date` attributes can have minimum and maximum values, as well as the optimal value. For example, the preferred price in this case is between $25 and $50, with an optimum of $40. `freeText` and `ontology` tags are taken as entered, while the value in `longText` fields are interpreted as keywords for matching throughout the descriptive text.

Attribute Creation

An XML file is used to provide extra information about the attributes (Figure 7.5). This is mainly to allow the attribute type to be easily determined, creating an object to match the base type of each attribute as part of the component object. It is also the connection

```
<?xml version="1.0"?>
<Description xmlns="http://www.scis.ecu.edu.au/swvML/1.0/"
       xmlns:dc="http://purl.org/dc/elements/1.0/"
       xmlns:swv="http://www.scis.ecu.edu.au/swvML/1.0/" >
<dc:description type="mandatory">game renderer browser</dc:description>    <====== longText
<swv:detail type="mandatory">game renderer browser</swv:detail>
<dc:publisher>Freshmeat</dc:publisher>                                      <====== freeText
<swv:devStatus type="mandatory">mature</swv:devStatus>
<swv:licence type="preferred">GNU General Public License (GPL)</swv:licence>
<swv:price type="preferred" min="25" max="50">40</swv:price>               <====== numeric
<swv:technical>
<swv:devLanguage type="preferred">Java</swv:devLanguage>                   <====== ontology
<swv:devLanguage>C++</swv:devLanguage>
  <swv:operatingSystem type="mandatory">Linux</swv:operatingSystem>
<swv:systemRequirements>
<swv:memory type="preferred" min="15" max="50">20</swv:memory>
<swv:diskSpace type="preferred" min="30" max="50">40</swv:diskSpace>
</swv:systemRequirements>
</swv:technical>
<dc:date type="preferred" min="2002-01-01" max="2004-01-01">2003-10-10</dc:date> <=== date
</Description>
</xml>
```

Figure 7.4: Ideal specification Game Renderer/Browser in CdCE XML Format, highlighting attribute types

```
<attributes>
  ...
  <attribute>
    <name>creator</name>
    <tag>dc:creator</tag>
    <type>freeText</type>
  </attribute>
  <attribute>
    <name>subject</name>
    <tag>swv:subject</tag>
    <type>ontology</type>
  </attribute>
  ...
</attributes>
```

Figure 7.5: Attribute Creation File in XML

between the subsequent tools that have been developed and the specific data model being used. As this file is read in, a list of attributes is created with the tags used as indices. To substitute a different schema with the same attribute types, the user would change the reference to this file and supply their attribute file.

Ontology

The ontology is stored as an XML file with six attributes for each ontology entry, referred to as a discriminator. One of the main purposes of the ontology is to provide a mapping between equivalent terms, for example: *Windows 98*, *W98* and *Win98*. A portion of

```
<ontology>
  ...
  <descriminator>
    <id>9</id>
    <name>Development Status :: 3 - Alpha</name>
    <mapsTo>3 - Alpha</mapsTo>
    <parent_id>6</parent_id>
    <root_id>6</root_id>
    <field>devStatus</field>
  </descriminator>
  <descriminator>
    <id>10</id>
    <name>Development Status :: 4 - Beta</name>
    <mapsTo>4 - Beta</mapsTo>
    <parent_id>6</parent_id>
    <root_id>6</root_id>
    <field>devStatus</field>
  </descriminator>
  ...
</ontology>
```

Figure 7.6: XML Ontology File

the ontology referring to project maturity/development status is displayed in Figure 7.6. This functionality uses the `<name>` value as the observed description and replaces it with the text given in the `<mapsTo>` tags. The `<parent_id>` and `<root_id>` fields help to create a hierarchy of related terms. As there may be some overlap in the terms used in various attributes, the `<field>` tag is included to match the ontology entry to the CdCE tag.

The OntologyCheck program tries to match an observed value against the `<name>` value for the matching `<field>`. If no match exists, a segment of XML is produced to use as a template for adding an entry for that value. Thus in the preprocessing stage, the ontology is used to ensure that any terminology from the source has been included and that it has all information required for comparisons.

Once the data is converted to CdCE format and the terms used are included in the ontology, the component data can be compared and analysed. All programs have been written in Java and make use of object-oriented techniques. One that is particularly relevant is polymorphism, where the five attribute types are implemented as objects with methods including: comparing their values with the ideal value for that attribute; and, giving a response in a number of formats.

7.3.2 Data Transformations

Even though the source data have been converted into CdCE format, they are still not in a format that can be processed by machine learning tools. Weka provides an implementation of a wide range of machine learning algorithms and requires the input in specific formats. For this work the format chosen is ARFF. ARFF files are text files with a strict structure and can take two types of attribute, real and multi-value. Figure 7.7 includes part of an ARFF file.

```
@relation component_training_data/Tue-Aug-03-20:46:49-WST-2004

@attribute dc:description {true,false,-999}
@attribute swv:detail {true,false,-999}
@attribute dc:publisher {true,false,-999}
@attribute dc:date {true,false,-999}
@attribute swv:licence {true,false,-999}
@attribute swv:devStatus {true,false,-999}
@attribute swv:devLanguage {true,false,-999}
@attribute swv:operatingSystem {true,false,-999}
@attribute swv:memory {true,false,-999}
@attribute swv:diskSpace {true,false,-999}
@attribute swv:price {true,false,-999}
@attribute result {accept,reject}

@data
false,false,true,false,true,-999,-999,-999,false,-999,-999,accept
false,false,true,true,true,-999,-999,-999,-999,-999,-999,accept
false,false,true,false,false,-999,-999,-999,false,-999,-999,accept
false,false,true,false,true,-999,-999,-999,false,-999,-999,accept
false,false,true,false,false,-999,-999,-999,false,-999,-999,accept
false,false,true,true,false,-999,-999,-999,-999,-999,-999,accept
false,false,true,true,true,-999,-999,-999,-999,-999,-999,accept
```

Figure 7.7: Sample ARFF File

The ARFF file structure begins with a header indicated by the relation declaration, which is useful for identifying the task and parameters being used. This is followed by a list of attributes, which are the columns/fields which will be given for each data point. Attributes are listed in order to match the data and indicate the datatype. In this project, two datatypes have been used: nominal and numeric. Nominal attributes require a list of all possible values to be included in the braces e.g. {true, false, -999}. Weka also includes string and date type attributes. The data declaration indicates that the rest of the file will be comma separated values (csv), each line representing a different datapoint.

Thus any data to be processed through Weka needs to be converted in strict types and, if not numeric, needs to have a finite number of values. This particularly affects

`freeText` and `longText` attributes. Weka also includes clustering and association rules algorithms that require their input data to be in a particular format, which can be used if the various attribute types are converted to suit the processing requirements.

Independent of the requirements of Weka, an examination of a variety of ways to represent the data was undertaken to determine how the representation affects the ability to give an overall picture of a component's suitability. Spiral 3 of this project gave results as matching or not matching the requirements (from the ideal specification). Other representations are to use a number of values (e.g. true, false and borderline) or to represent the comparison as a number. In the literature there are two real options for input data for analysis tools, numeric and multi-valued text - which needs to be considered when targeting these tools as part of a process.

Missing Data

An issue that users have little control over is missing data values in the repository dataset. Due to differences in data models, there can be up to 100% of data points missing a particular attribute value. An example from the freshmeat dataset is price. The Open Source nature of freshmeat results in no equivalent attribute for price. Iteration to refine the ideal specification allows us to deal with these types of issues. Missing data is not uncommon in commercial repositories (Yoon et al, 1999), and having to deal with missing values makes our dataset more realistic and tests the robustness of our approach when faced with such issues.

Previous work in this project had problems with the representation and handling of missing values, described in Chapter 6. The ARFF format requires a valid value in each field (attribute) and it was considered preferable to have a predictable response to missing data. For example, in missing numeric values, -1 and -100 were trialled. Both were problematic as a requirement for a value being <50 would evaluate to true, while >50 would be false - whereas either both should be true, or both false. A particular problem with using -1 to represent missing values is that it was so close to the (possibly) acceptable value 0, that it gave regular false positive results.

Methods for dealing with missing data include ignoring/deleting data points, replacing the missing value with a symbol, replacing the value with the mean or mode (often

with restriction to similar data points), replacing with a predicted value (e.g. via clustering) or to use or modify an algorithm to deal with the missing data (Batista and Monard, 2003, Grzymala-Busse and Hu, 2001).[3] Missing values are frequently indicated by out of range entries, such as negative numbers in numeric fields (Witten et al, 2011). For example, -999 is unlikely to be a valid value for any fields in the CdCE data model. After consideration of the attribute and their likely values, -999 was chosen as the missing value and the classifier trained accordingly. When the classifier was not specifically trained on missing values, there were unexpected results to a -999 in the test data (e.g. apparently random classification). Specific handling of missing values was added to the training data, based on a parameter to allow the missing value to be taken as a match, a mismatch or borderline. This resulted in consistent classification results. -999 is also used to represent missing string/text values in textual fields.

Transformation

To allow for systematic experimentation, a set of equivalent transformations was developed across the five attribute types. They begin with a two-value (Boolean) result as to whether the observed value matched the required - Transformation 1 (T1) {true, false}. As values close to matching may be preferred over those that are complete mismatches, Transformation 2 (T2) introduces a third value to represent borderline cases {true, false, border}. A numeric result is generated for the comparison as Transformation 3 (T3), with a 4th transformation, T4, for those attributes that lend themselves to more than one calculation. Transformation 5 (T5) allows levels and abstraction of concepts when dealing with ontology attributes.[4] The details of each of the transformations is described below in terms of each attribute type.

Not every attribute will be specified in the ideal specification. Unspecified attributes may be ignored or there is a facility for a default value to be substituted. The transformations include a default output that has been chosen for unspecified attributes as they do not have a value in the ideal specification to which to be compared. This is flexible and may be switched off and on via parameters. These alternate outputs are indicated in

[3]Data cleansing could also be used to insert values into attributes, based on the given data and associations between attributes. That approach has not been adopted in this work.

[4]An additional transformation (T0) is included in the code to output the raw data ('LongText' is output for LongText attributes for practical reasons). This is for development purposes and not intended for input into a classifier.

the 'Specified' and 'Unspecified' columns in the tables that follow. For text based fields, all comparisons are done on the text after converting it to lowercase. Thus matches are not sensitive to the case in the original data.

The selected transformation is applied to all attributes. The programs also have facility for the transformation to be defined at the individual attribute level, requiring alteration of the initialisation sections of the code.

Numeric Attributes

Any scaling or unit conversions have taken place during the conversion of the data to CdCE, so a comparison of `numeric` values is a combination of the *observed* value and the *optimal*, *minimum* and *maximum* values from the ideal specification. Table 7.5 describes the output of each numeric attribute transformation, based on the transformation type, the values being compared, and whether the attribute is in the ideal specification.

Trans.	**Specified**	**Not Specified**	**Output Type**
T1	if (minimum < value < maximum) then = true else = false	false	(true,false,-999)
T2	if (minimum < value < maximum) then = true else if ((min - delta1) < value < (maximum + delta2)) then = border else = false	false	(true, false, border, -999)
T3	if (minimum < value < maximum) then = 1 else if ((min - delta1) < value < (maximum + delta2)) then = f(min, max, value) else = 0	false	real
T4	as in Trans. 3	as in Trans. 3	as in Trans. 3
T5	as in Trans. 3	as in Trans. 3	as in Trans. 3

Table 7.5: Numeric Attribute Transformation

Table 7.5 and Figure 7.8 show the difference between the numeric attribute transformations. In T1, the result is true or false, depending on whether the observed value is within the *min* - *max* range. In T2 the *border* areas are calculated using the distance between the *optimal* and *minimum* and *maximum* values. These are represented in the table as *delta1*, *optimal-minimum*, and *delta2*, *maximum-optimal*. The borders are indicated in the Figure as the areas between *min_b* and *min*, and *max* and *max_b*.

In the T2 transformation, a result is true if within the specified range, border if in the border areas, and false otherwise. In T3, the result is 1 if within the specified range, and

0 if beyond the border areas. If the value falls within the border areas, and outside of the specified range, the result will be a number between 0 and 1. For `numeric` attributes, T4 and T5 are the same as T3, summarised in Table 7.5.

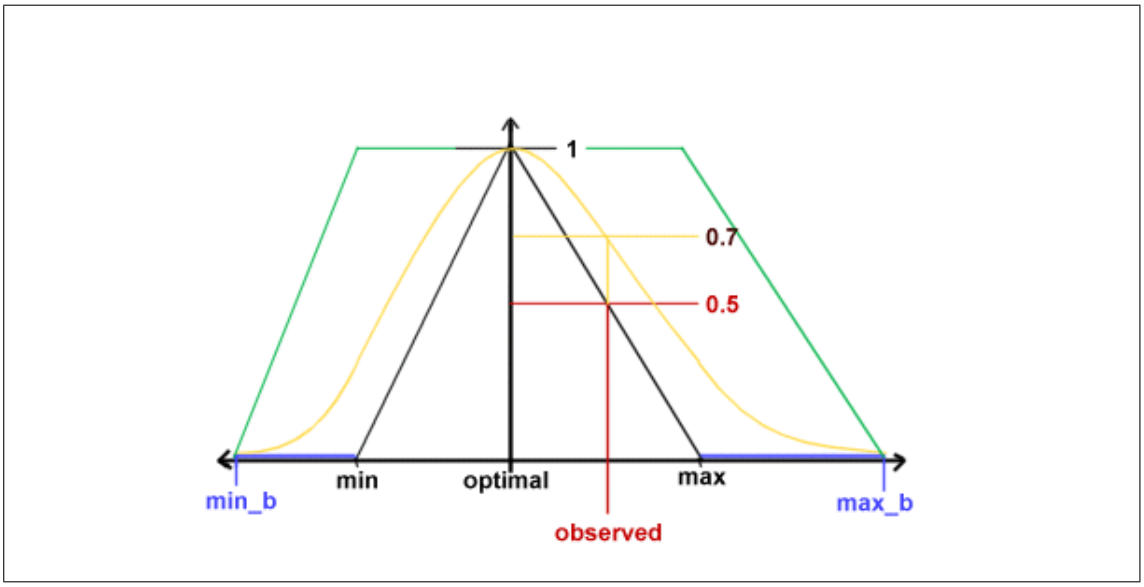


Figure 7.8: Transforming Numeric Attributes

Date Attributes

As `date` attributes can be required to be within a range of two dates, they are processed in the same way as numeric values. Although they are stored internally as dates, the transformations output them as multi-value or numeric. T2 and T3 use the same technique for calculating the delta values as for the numeric attributes. T4 and T5 are the same as T3, as shown in Table 7.6.

FreeText Attributes

`freeText` attributes have a limited range of comparisons available. T1 uses basic string comparisons and gives a boolean result. It is possible that the value has more information than expected (e.g. 'Sun Microsystems' where the ideal specification value is 'Sun'). In that situation T2 would class the result of the comparison as a borderline match as a substring can be found. The third comparison is a technique to generate a numeric result representing the similarity between two text strings. The values in each position in the strings is compared and the number of matches counted.

Trans.	Specified	Not Specified	Output Type
T1	if (minimum < value < maximum) then = true else = false	false	(true,false,-999)
T2	if (minimum < value < maximum) then = true else if ((min - delta1) < value < (maximum + delta2)) then = border else = false	false	(true, false, border, -999)
T3	if (minimum < value < maximum) then = 1 else = 0 else if ((min - delta1) < value < (maximum + delta2)) then = f(min, max, value) else = 0	false	real
T4	as in Trans. 3	as in Trans. 3	as in Trans. 3
T5	as in Trans. 3	as in Trans. 3	as in Trans. 3

Table 7.6: Date Attribute Transformation

Trans.	Specified	Not Specified	Output Type
T1	if matched = true else false	false	(true,false,-999)
T2	if matched = true else = false else if substring matched = border else = false	false	(true, false, border, -999)
T3	#matching letters/#letters in shorter word	0	real
T4	as in Trans. 3	as in Trans. 3	as in Trans. 3
T5	as in Trans. 3	as in Trans. 3	as in Trans. 3

Table 7.7: FreeText Attribute Transformation

The matching process stops with the shorter of the two strings.[5]

For `freeText` attributes, T4 and T5 are the same as T3. The `freeText` transformations are summarised in Table 7.7.

LongText Attributes

`longText` attributes are an unrestricted description which is internally converted into a list of keywords and counts. An extendable list of stop words is included which are ignored by the keyword counting process. The ideal specification includes a list of required keywords for `longText` attributes. T1 will return 'true' if all of the keywords have at least one occurrence, 'false' otherwise. To extend this, T2 returns 'true' if all the keywords are matched, 'false' if none are matched, and is borderline if some of the keywords are matched. Missing 1 of 4 keywords is thus the same as missing 3 of 4. T3 gives more information about the closeness of the match by returning the number of keywords

[5]There may be better or alternative techniques to give a more meaningful measure of similarity, but these were not followed through on as there was a low expected value in terms of results.

Trans.	Specified	Not Specified	Output Type
T1	if all matched = true else false	false	(true,false,-999)
T2	if all matched = true else if some matched = border else = false	false	(true, false, border, -999)
T3	# keywords matched/# keywords required	0	real
T4	# keywords matches/# keywords in text	0	real
T5	as in Trans. 3	as in Trans. 3	as in Trans. 3

Table 7.8: LongText Attribute Transformation

Want / Have	Plan	Pre-Alpha	Alpha	Beta	Prod'n /Stable	Mature
Planning	10	8	6	4	2	0
Pre-Alpha	10	10	8	6	4	2
Alpha	10	10	10	8	6	4
Beta	10	10	10	10	8	6
Prod/Stable	10	10	10	10	10	8
Mature	10	10	10	10	10	10

Table 7.9: Distance matrix for maturity attribute

matched, divided by the total number of keywords required. So a full match = 10, 3 from 4 is 7.5 and 1 from 4 is 2.5 (scaled from 0-10). Continuing the numeric representation of the keyword match, T4 takes the length of the passage of text into account, so takes a count of all the matches, divided by the total number of keywords in the text. For `longText` attributes, T5 is the same as T3. All transformations for `longText` are listed in Table 7.8.

Ontology Attributes

`ontology` attributes are treated in the same way as `freeText` attributes for T1. As the `ontology` attributes have been mapped to the consistent representation of concepts, they are more likely to match than `freeText`. T2 and T3 make use of the ontology knowledge base to determine the similarity of terms. T2 will be able to match similar concepts and return a 'true' result if they are closely related. If there is some relation between the concepts, a 'border' result will be given, while unrelated or conflicting concepts will return 'false'. The third transformation calculates the distance between the ideal and observed terms and will return 10 if they are very close, and 0 if distant, and a value between 0 and 10 if they are in the middle ground. This calculation is facilitated by a distance matrix, stored in XML. Each value for each attribute is listed on the axes for

Trans.	Specified	Not Specified	Output Type
T1	if matched = true else = false	false	(true,false,-999)
T2	if very close in ontology = true else if nearby in ontology = border else = false	false	(true, false, border, -999)
T3	ontology calculation of distance (0-10) within ontology	0	real
T4	as in Trans. 3	as in Trans. 3	as in Trans. 3
T5	if level <= ideal_level then value else parent(value,level)	same as spec.	('list of all ontology values at that level', -999)

Table 7.10: Ontology Attribute Transformation

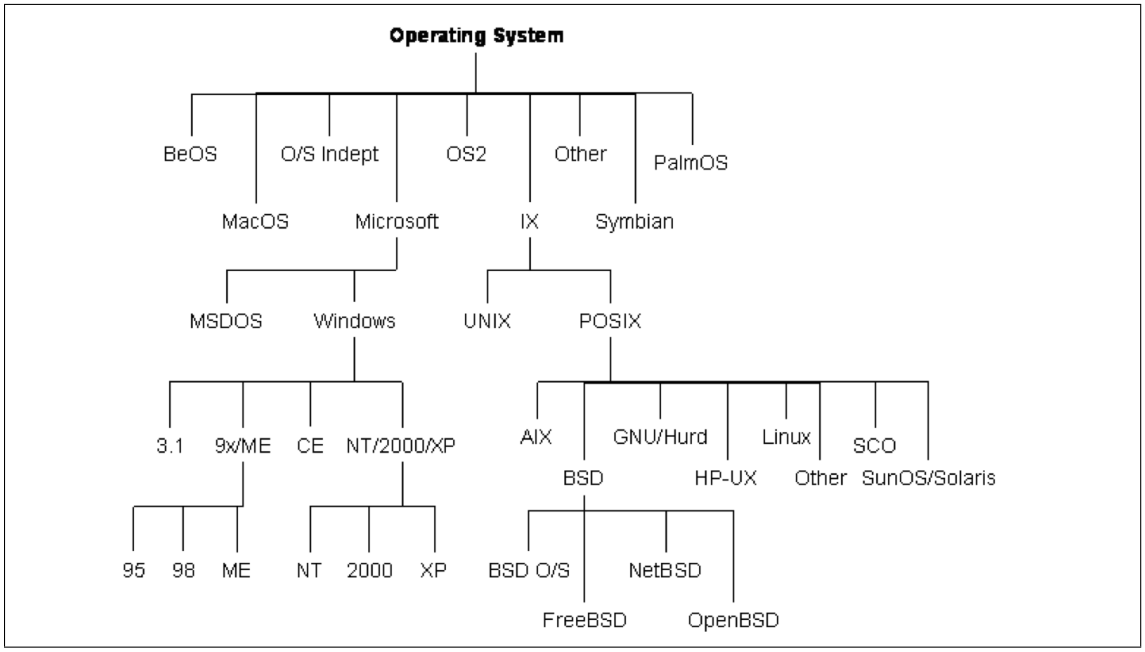


Figure 7.9: Ontology entries for Operating System

the matrix. The distances are filled in by expert opinion and can be localised or updated by substituting a matrix file. An example is the development status, shown in Table 7.9. In this case, if the requirement is for at least `Beta`, then `Production`, `Stable` or `Beta` would match. The core reduces as the observed value moves further from the required range. For `ontology` attributes, T4 is the same as T3 to allow the observation of impact of T4 for `longText`. All `ontology` transformations are in Table 7.10.

Problems can occur in training the classifier if there are a large number of possible values for a particular attribute. For example, the Operating Systems part of the ontology has 31 entries (Figure 7.9). There are nine children at the first level, with only two of them branching to further levels. When viewing and processing data, a restriction of the hierarchy to a level (e.g. level 2) may be a better indicator of the suitability of a piece

of software. The system offers the flexibility to set a cut-off level, so the user can choose to use the entire tree, or prune it to a specified level. This approach is applied to all ontology attributes. Thus using T5 will prune the tree to a level (depth) and replace any values with their direct ancestor at the required level. For example, in Figure 7.9 a level 3 cut-off would replace `Win98` with `9x/ME` and `NetBSD` with `BSD`.

7.4 Spiral 4 Implementation

The generation and transformation software has been developed as a series of Java applications. Each of these uses the SAX parser to read XML data files. The ideal specification, attribute information, ontology and transformed component data are held in XML files. Files for processing by Weka (training, test and repository data) are stored in ARFF format.

Inputs	**Outputs**
ideal component specification (CdCE XML) attribute information (XML) thresholds for attribute priority `MISSING` value representation (default -999) `SKIP_UNSPECIFIED` directive `TRANSCODE` (and level if applicable)	training data set (ARFF) test data set(s) (ARFF)

Table 7.11: Generation of Training Data

The generation of training data has been updated for the new data representation. The underlying work is the same, with input parameters and outputs shown in Table 7.11. New parameters to the generator are the missing value, skip directive and transformation code. The `MISSING` value string can be specified to any user value. The default used in this work is '-999' as it can be interpreted as a real or as a string - allowing consistent missing value replacement across all attributes and transformations. Not all attributes are required for all classification tasks. The `SKIP_UNSPECIFIED` directive is used to toggle whether to include attributes that are not in the ideal specification, or to skip them. The final parameter is the `TRANSCODE`. This value should be an integer, currently ranging from 1-5. The data transformations matching each transformation were described in Section 7.3.2. For transformations utilising the ontology (i.e. T5), a `level` should also be specified. The default level is 2, and the value is ignored if other transformations are used.

The training data generation begins similarly to that described in Chapter 6, with

a permutation of Y/N values on the attributes in the ideal specification. This is the first pass of training generation. To match the transformations for each attribute, the generator must consider the attribute type and all possible values on that attribute.

The second pass in the generation of the data is to substitute values in place of the Y/N placeholders. This step needs to take into account the `TRANSCODE` value and the ideal specification to determine valid (matching) or invalid (not matching) values. For T1, there is a simple substitution of `true` for `Y` and `false` for `N`. This creates a strict matching of attributes against criteria. The experiments were run with all missing values set as not matching, so in all transformations, '-999' (`MISSING`) can be substituted for `N`. In T2, the criteria are loosened to allow the 'borderline' cases to be accepted, thus `true` and `border` are substituted for `Y` and `false` and `-999` for `N`. T3 involves converting all of the data to numeric values. The generator can determine these values from the ideal specification and can then substitute acceptable numeric values for `Y` and unacceptable values (and `-999`) for `N`. T4 is similar, with differences in the values used for some attribute types. Values for T5 are more closely matched to the raw data. `ontology` attributes are the focus of this transformation, with all other attributes treated as in T3. T5 works on a manageable subset of the ontology for each attribute. This can be split into acceptable and unacceptable values according to the ideal specification. Using these two groups of values, the generator can substitute values for `Y` and `N`. More detail on the transformations is in Section 7.3.2.

Using the transformation information, the generator can output an ARFF header to include the attribute names, types and all possible values. This is followed by the training data which retains the label (accept/reject) from the first pass of the generation. An example of training data which uses T1, skips unspecified attributes and has -999 as the missing value is in Figure 7.10.

```
@relation component_training_data/Tue-Aug-24-16:49:40-WST-2004

@attribute dc:description {true,false,-999}
@attribute dc:publisher {true,false,-999}
@attribute dc:date {true,false,-999}
@attribute swv:licence {true,false,-999}
@attribute swv:devStatus {true,false,-999}
@attribute swv:devLanguage {true,false,-999}
@attribute swv:operatingSystem {true,false,-999}
@attribute swv:memory {true,false,-999}
@attribute swv:diskSpace {true,false,-999}
@attribute swv:price {true,false,-999}
@attribute result {accept,reject}

@data
false,true,false,true,true,-999,false,-999,-999,-999,accept
false,true,false,true,false,false,-999,-999,-999,-999,accept
false,true,false,false,false,false,true,-999,-999,-999,accept
false,true,false,true,-999,-999,-999,-999,-999,-999,accept
false,true,false,false,-999,-999,-999,-999,-999,-999,accept
false,true,false,false,-999,-999,-999,-999,-999,-999,accept
false,true,false,true,true,false,false,-999,-999,-999,accept
false,true,false,true,-999,-999,-999,-999,-999,-999,accept
false,true,false,false,-999,-999,-999,-999,-999,-999,accept
false,true,false,true,-999,-999,-999,-999,-999,-999,accept
false,true,false,true,-999,-999,-999,-999,-999,-999,accept
```

Figure 7.10: Training Data (Transformation 1)

7.4.1 Transformation of Repository Data

Inputs	Outputs
repository data (original form) ontology (XML)	repository data (CdCE XML) repository data (ARFF)

Table 7.12: Transformation of repository data

Table 7.12 and Figure 7.11 show the main elements in the transformation process. This begins with the data file at the tag/attribute level using a datasource specific program, in this case FM2CdCE. A first pass is run to check that the values observed within the ontology attributes exist in the CdCE ontology. If not, the user is given templates to add new terms to the ontology. The updated ontology and the converted data file are run through CdCEOntology to convert the ontology attributes using the mappings described in the ontology. Maintenance of the ontology is currently a manual process. Once the ontology mappings are complete, the data can then be transformed based on the ideal specification values.

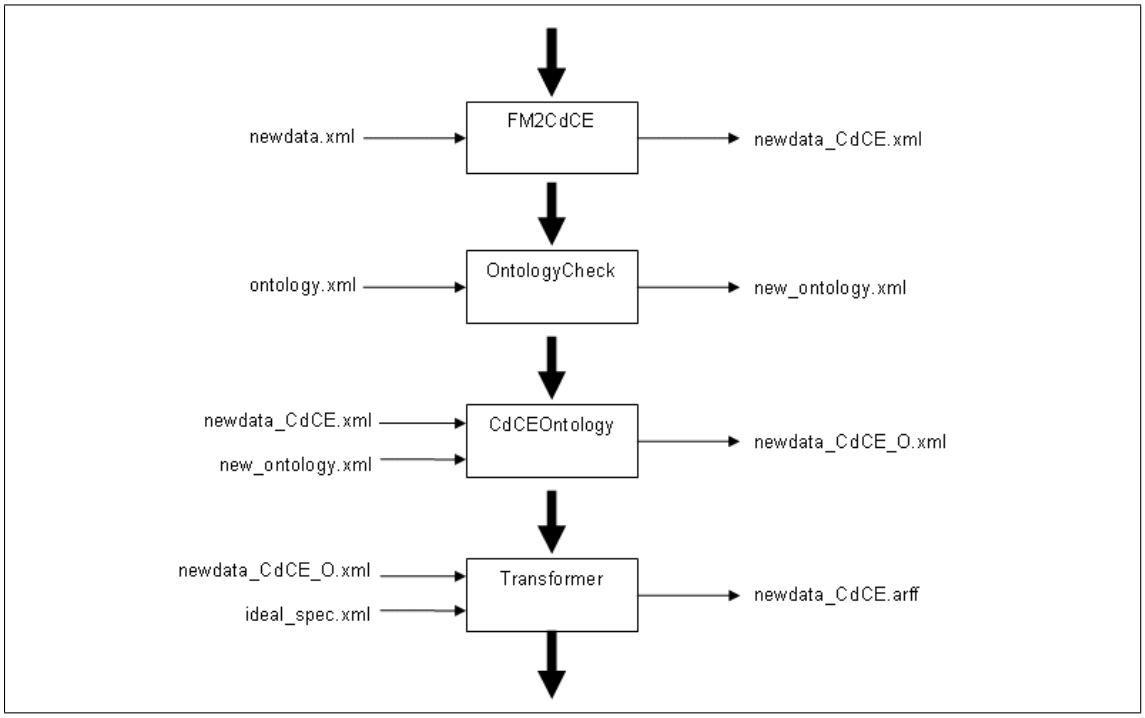


Figure 7.11: Transformation process

Conversion to CdCE

Each conversion from data source to CdCE is written as a separate program coded as a SAX filter. The initial step is to get the full data model for the data source and investigate any conventions they have for recording details.

Table 7.13 shows the mapping between tags in the freshmeat RDF file and the CdCE output file. Although many of the omitted fields hold interesting information, they are unlikely to be available from other sources. Where there are multiple fields from which to choose an attribute, the logic used for the conversion is shown in the right hand column of the table. For example, there are at least four date fields in the freshmeat file for each piece of software. The logic to choose the date to use in the CdCE transformation is:

```
if no latest_release_date then
        date = date_added
else
        date = latest_release_date
```

The `date_updated` was a candidate for use as the 'date'; however it is triggered when comments are added to freshmeat project records. This is not useful when looking for an indication of the most recent update to the software, but may help to indicate activity or vitality of the project.

freshmeat	**CdCE**
-	Publisher = Freshmeat
project_id	Identifier = project_id
date_added	If no latest_release_date date = date_added Else date = latest_release_date
date_updated	omitted
projectname_short	If no projectname_full title = projectname_short Else title = projectname_full
projectname_full	omitted
desc_short	description = desc_short
desc_full	detail = desc_full
vitality_score	omitted
vitality_percent	omitted
vitality_rank	omitted
popularity_score	omitted
popularity_percent	omitted
popularity_rank	omitted
rating	omitted
rating_count	omitted
rating_rank	omitted
subscriptions	omitted
branch_name	omitted
url_homepage	omitted
url_tgz	omitted
url_changelog	omitted
url_rpm	omitted
url_deb	omitted
url_bz2	omitted
url_cvs	omitted
url_list	omitted
url_zip	omitted
url_osx	omitted
url_bsdport	omitted
url_purchase	omitted
url_mirror	omitted
url_demo	Demo = url_demo
url_project_page	Support = url_project_page Creator = url_project_page Source = url_project_page
license	Licence = license
latest_release	**Not a repeated field
latest_release_version	Version = latest_release_version
latest_release_id	omitted
latest_release_date	See 'date'
screenshot_thumb	omitted
descriminators (trove_id*)	Dev status, licence, O/S, prog lang and topic all linked through trove_ids
trove_id	Link through to: Development Status, Platform = Environment (dependencies), Operating System, Licence, Framework = network environment, Programming Language, Subject = Topic , Language = translations
dependencies (dependency*)	Multiple dependencies allowed
dependency_project_id	Relation.id = dependency_project_id
dependency_branch_id	omitted
dependency_release_id	Relation.version = dependency_release_id
dependency_project_title	Relation.value = dependency_project_title
-	Relation.source = Freshmeat
dependency(type)	Relation.rel_type = dependency(type)

Table 7.13: freshmeat conversion

7.4.2 Classifying the Data

Inputs	Outputs
repository data (ARFF) classifier model (Weka binary .mod)	shortlist from repository data (XML)

Table 7.14: Classifying the data

At this point the training data has been used to create the predictive model of the selection task. This is stored as a Weka binary model with a .mod extension. Weka can now be run with the same parameters as used in training the model, adding the model file and the repository data file as inputs. The parameters for classifying the data are given in Table 7.14. The output of this run is the dataset annotated with the class that the model predicted for each component. As all instances in the repository data file have 'reject' in the output class, entries showing that 'reject' was not the correct class are put onto the shortlist. In the example shown in Figure 7.12, one entry out of 41885 is rejected - which would provide a shortlist with one entry.

The command used for classifying the data is:

```
java -Xint -classpath "${WEKAHOME}" weka.classifiers.trees.J48  \
       -l model.mod-T $REAL
```

The raw output from Weka is post-processed to create a new XML file holding the shortlisted component information.

```
=== Classifier model (full training set) ===
J48 pruned tree
------------------

dc:description <= 4: reject (9900.0)
dc:description > 4
|   swv:detail <= 4: reject (2970.0)
|   swv:detail > 4
|   |   swv:devLanguage = Ada: reject (30.0)
|   |   swv:devLanguage = APL: reject (30.0)
<--- snip --->
|   |   swv:devLanguage = Zope: reject (33.0)
|   |   swv:devLanguage = error: reject (33.0)
|   |   swv:devLanguage = -999: reject (66.0)

Number of Leaves  :  138

Size of the tree :  144

=== Error on test data ===

Correctly Classified Instances       41884               99.9976 %
Incorrectly Classified Instances         1                0.0024 %
Kappa statistic                          0
Mean absolute error                      0
Root mean squared error                  0.0049
Total Number of Instances            41885

=== Confusion Matrix ===

     a     b   <-- classified as
     0     0 |     a = accept
     1 41884 |     b = reject

{Experiment run: calc_2Jan08t5s23_2008-12-31_21_01}
```

Figure 7.12: Results of classification of the component data

7.4.3 Supporting Code and Scripts

The code developed across Spirals 3 and 4 is held in CdCETransformer and Intelligent - two Java applications. Both require xmlwriter and xerces classes for handling XML. The parameters for both programs are held in XML files, such as the one shown in Figure 7.13. The file provides settings for transformations, missing data handling, skip unspecified setting and gives filenames for the ontology, attribute list and the ideal specification.

The output of CdCETransformer and Intelligent write to files and to standard out (the screen) and standard error (defaults to the screen). These are both captured using redirections on the command line. CdCETransformer provides the component repository data in ARFF and Intelligent creates the training and test data files in ARFF. The names of the files output from CdCETransformer and Intelligent include information on the parameters and time it was run.

```
<PARAMETERS>
<TEMPLATE>CdCETemplate.xml</TEMPLATE>
<FILE_BASE>fm_projects06</FILE_BASE>
<IDEAL_SPEC>ideal_calc_2Jan08t5s23.xml</IDEAL_SPEC>
<VERSION>calc_2Jan08t5s23</VERSION>
<TRANSCODE>5</TRANSCODE>
<ONTOLOGY_LEVEL>2</ONTOLOGY_LEVEL>
<SKIP_UNSPECIFIED>true</SKIP_UNSPECIFIED>
<MISSING>-999</MISSING>
</PARAMETERS>
```

Figure 7.13: Parameter file for ideal calculator case study

In Spiral 3, Weka was used in interactive mode, through the GUI interface. A series of scripts have been developed to facilitate the testing and experimentation with the shortlisting tools (see Appendix B). These allow multiple sets of parameters and/or ideal specifications to be processed in one command. They also ensure that each run has all the input and output files bundled into identifiable directories.

The first script, xml_exp_search, runs a set of related scenarios, based on a common prefix entered at the command line. A scenario will have an ideal specification and a matching parameter file. For each scenario, a second script, process, is called. This script creates directories and stores files which are at risk of being overwritten. The script then runs Intelligent and CdCETransformer to create the training, test and component data files for the scenario. The training and classification of component data is carried out using the weka_train script. The output of weka_train is a series of output files from processing the training, test and component input files. The last step of the process script moves the scenario files into the created directory, restores the saved files and deletes temporary directories.

After all of the scenarios have run through in the process script, a final script, grab_predict, is called to take the output files and create XML shortlist files for each of the scenarios. It also counts the number of matches in each scenario as a guide for the user. Using these scripts, the user can consider a range of ideal specifications in a systematic manner, rather than picking those they expect to give good results. This may lead to unexpected choices which result in shortlists that are more suitable to the selection task.

7.5 Spiral 4 Results

Output of Transformations

To illustrate the data transformation process a sample file of COTS data is followed as it is processed using transformations T1-T5. The process begins with the source data in freshmeat RDF format (Figure 7.14). Table 7.13 details the mapping of tags and values from freshmeat to CdCE. After running the source file through the FM2CdCE filter, the same data is represented in CdCE format (Figure 7.15). The most involved conversions are those of the `<trove>` elements. They use the ontology to map the `<trove_id>` to an ontology element which indicates the tags and values to be added to the CdCE file.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project-listing>
  <project>
    <project_id>2</project_id>
    <date_added>2000-07-23 16:44:50</date_added>
    <date_updated>2003-12-17 19:04:05</date_updated>
    <projectname_short>Overkill</projectname_short>
    <projectname_full>Overkill</projectname_full>
    <desc_short>A bloody action 2D deathmatch game in ASCII art.</desc_short>
    <desc_full>Overkill is a client-server 2D deathmatch-like game ... editor.</desc_full>
    <vitality_score>15.70</vitality_score>
    <vitality_percent>0.01</vitality_percent>
    <vitality_rank>6312</vitality_rank>
    <popularity_score>870.70</popularity_score>
    <popularity_percent>2.03</popularity_percent>
    <popularity_rank>1865</popularity_rank>
    <rating>7.69</rating>
    <rating_count>12</rating_count>
    <rating_rank>237</rating_rank>
    <subscriptions>14</subscriptions>
    <branch_name>Default</branch_name>
<--- snip --->
    <url_demo></url_demo>
    <license>GNU General Public License (GPL)</license>
    <latest_release>
      <latest_release_version>0.16</latest_release_version>
      <latest_release_id>64935</latest_release_id>
      <latest_release_date>2001-12-16 08:52:36</latest_release_date>
    </latest_release>
    <screenshot_thumb>http://images.freshmeat.net/screenshots/2_thumb.jpg</screenshot_thumb>
    <descriminators>
      <trove_id>15</trove_id>
      <trove_id>82</trove_id>
<--- snip --->
      <trove_id>2</trove_id>
    </descriminators>
    <dependencies>
    </dependencies>
  </project>
  ...
</project-listing>
```

Figure 7.14: freshmeat source file

```
<?xml version="1.0"?>
<swv:components
  xmlns="http://scis.ecu.edu.au/research/PhD/vmaxvill/swvMLap/1.0/"
  xmlns:dc="http://purl.org/dc" >
  <swv:component>
    <dc:publisher>Freshmeat</dc:publisher>
    <dc:identifier>2</dc:identifier>
    <dc:description>A bloody action 2D deathmatch game in ASCII art.</dc:description>
    <swv:detail>0verkill is a client-server 2D deathmatch-like game
    in ASCII art. It supports free connecting/disconnecting during the game, and
    runs well on modem lines. Graphics are in 16-color ASCII art with elaborate hero
    animations. 0verkill features 4 different weapons, grenades, invisibility, and armor.
    The package also contains reaperbot clients, a simple graphics editor, and a level
    editor.</swv:detail>
    <dc:creator>http://freshmeat.net/projects/0verkill/</dc:creator>
    <dc:source>http://freshmeat.net/projects/0verkill/</dc:source>
    <swv:support>http://freshmeat.net/projects/0verkill/</swv:support>
    <swv:demo></swv:demo>
    <swv:licence>GNU General Public License (GPL)</swv:licence>
    <swv:version>0.16</swv:version>
    <swv:licence>GNU General Public License (GPL)</swv:licence>
    <swv:subject>First Person Shooters</swv:subject>
    <swv:standard>Console (Text Based)</swv:standard>
    <swv:standard>X11 Applications</swv:standard>
    <swv:operatingSystem>OS/2</swv:operatingSystem>
    <swv:operatingSystem>Windows</swv:operatingSystem>
    <swv:operatingSystem>Windows 95/98/ME</swv:operatingSystem>
    <swv:operatingSystem>Windows NT/2000/XP</swv:operatingSystem>
    <swv:operatingSystem>POSIX</swv:operatingSystem>
    <swv:operatingSystem>FreeBSD</swv:operatingSystem>
    <swv:operatingSystem>IRIX</swv:operatingSystem>
    <swv:operatingSystem>Linux</swv:operatingSystem>
    <swv:operatingSystem>Other</swv:operatingSystem>
    <swv:operatingSystem>SunOS/Solaris</swv:operatingSystem>
    <swv:subject>Arcade</swv:subject>
    <swv:devStatus>5 - Production/Stable</swv:devStatus>
    <swv:operatingSystem>HP-UX</swv:operatingSystem>
    <swv:subject>Games/Entertainment</swv:subject>
    <audience>End Users/Desktop</audience>
    <dc:date>2001-12-16 08:52:36</dc:date>
    <dc:title>0verkill</dc:title>
  </swv:component>
  ...
</swv:components>
```

Figure 7.15: freshmeat data in CdCE format

Transforming the file ready for classification is done by combining the information in the ideal specification with the observed values for each component on each attribute. Figure 7.16 shows the ideal specification used for this example. The details of the transformations for each attribute type are in Section 7.3.2. T0 (Figure 7.17) is intended to assist in understanding the data and is not valid ARFF. `LongText` attributes are substituted with 'LongText' while all other attributes pass through their basic text or numeric representation.

```
<?xml version="1.0"?>
<Description xmlns="http://www.scis.ecu.edu.au/swvML/1.0/"
xmlns:dc="http://purl.org/dc/elements/1.0/" xmlns:swv="http://www.scis.ecu.edu.au/swvML/1.0/">
  <dc:description type="mandatory">Internet email application</dc:description>
  <swv:detail type="mandatory">Internet email application</swv:detail>
  <dc:publisher>Freshmeat</dc:publisher>
  <swv:devStatus type="mandatory">5 - Production/Stable</swv:devStatus>
  <swv:licence type="preferred">GNU General Public License (GPL)</swv:licence>
  <swv:price type="preferred" min="25" max="50">40</swv:price>
  <swv:technical>
     <swv:devLanguage type="preferred">Java</swv:devLanguage>
     <swv:devLanguage>C++</swv:devLanguage>
     <swv:operatingSystem type="mandatory">Linux</swv:operatingSystem>
     <swv:systemRequirements>
         <swv:memory type="preferred" min="15" max="50">20</swv:memory>
         <swv:diskSpace type="preferred" min="30" max="50">40</swv:diskSpace>
     </swv:systemRequirements>
     <Zspec> ... </Zspec>
  </swv:technical>
  <dc:date type="preferred" min="2002-01-01" max="2004-01-01">2003-10-10</dc:date>
</Description>
</xml>
```

Figure 7.16: Ideal specification for Internet email application

```
@relation component_training_data/Fri-Aug-13-15:24:06-WST-2004

@attribute dc:description {longText,-999}
@attribute swv:detail {longText,-999}
@attribute dc:publisher {freeText,-999}
@attribute dc:date real
@attribute swv:licence {ontology,-999}
@attribute swv:devStatus {ontology,-999}
@attribute swv:devLanguage {ontology,-999}
@attribute swv:operatingSystem {ontology,-999}
@attribute swv:memory real
@attribute swv:diskSpace real
@attribute swv:price real
@attribute result {accept,reject}

@data
longText,longText,Freshmeat,2001-12-16,GNU General Public License (GPL),
      5 - Production/Stable,-999,OS/2,-999,-999,-999,accept
longText,longText,Freshmeat,2003-5-30,GNU General Public License (GPL),
      4 - Beta,C,-999,-999,-999,-999,accept
...
```

Figure 7.17: Transformation 0 of freshmeat data to illustrate the raw data for the scenario (with instances word-wrapped for formatting reasons)

T1 (Figure 7.18) provides a Boolean result of matching each attribute value with the ideal specification for that attribute. All possible values (including 'missing') must be listed in the ARFF header, unless the attribute is numeric. T2 (Figure 7.19) adds a third value to the output allowing for less strict comparisons between observed and required values. This allows the classifier the option of treating near misses differently to extreme values.

```
@relation component_training_data/Fri-Aug-13-15:30:13-WST-2004

@attribute dc:description {true,false,-999}
@attribute swv:detail {true,false,-999}
@attribute dc:publisher {true,false,-999}
@attribute dc:date {true,false,-999}
@attribute swv:licence {true,false,-999}
@attribute swv:devStatus {true,false,-999}
@attribute swv:devLanguage {true,false,-999}
@attribute swv:operatingSystem {true,false,-999}
@attribute swv:memory {true,false,-999}
@attribute swv:diskSpace {true,false,-999}
@attribute swv:price {true,false,-999}
@attribute result {accept,reject}

@data
false,false,true,false,true,true,-999,false,-999,-999,-999,accept
false,false,true,false,true,false,false,-999,-999,-999,-999,accept
false,false,true,false,false,false,false,true,-999,-999,-999,accept
false,false,true,false,true,-999,-999,-999,-999,-999,-999,accept
false,false,true,false,false,-999,-999,-999,-999,-999,-999,accept
false,false,true,false,false,-999,-999,-999,-999,-999,-999,accept
false,false,true,false,true,true,false,false,-999,-999,-999,accept
...
```

Figure 7.18: Transformation 1 of freshmeat data

T3 and T4 (Figures 7.20 and 7.21) use numeric values to represent the closeness of a match. The final example is the output for T5 (Figure 7.22), where the ontology attributes: `operatingSystem`, `developmentLanguage` and `developmentStatus`, show the results of the abstraction of the included information to the second level. Not all ontology attributes have a tree hierarchy, but this facility has been shown to improve returned results as compared to T1 to T4.

It is clear that there are many missing values (-999) in this data set. This is indicative of real world data issues and will result in smaller than expected shortlists if any of those attributes are considered mandatory.

```
@relation component_training_data/Fri-Aug-13-15:40:45-WST-2004

@attribute dc:description {true,false,border,-999}
@attribute swv:detail {true,false,border,-999}
@attribute dc:publisher {true,false,subString,-999}
@attribute dc:date {true,false,border,-999}
@attribute swv:licence {true,false,border,-999}
@attribute swv:devStatus {true,false,border,-999}
@attribute swv:devLanguage {true,false,border,-999}
@attribute swv:operatingSystem {true,false,border,-999}
@attribute swv:memory {true,false,border,-999}
@attribute swv:diskSpace {true,false,border,-999}
@attribute swv:price {true,false,border,-999}
@attribute result {accept,reject}

@data
border,false,true,border,true,true,-999,false,-999,-999,-999,accept
false,false,true,border,true,false,false,-999,-999,-999,-999,accept
false,border,true,border,false,false,false,true,-999,-999,-999,accept
false,false,true,border,true,-999,-999,-999,-999,-999,-999,accept
border,border,true,false,false,-999,-999,-999,-999,-999,-999,accept
false,false,true,border,false,-999,-999,-999,-999,-999,-999,accept
false,false,true,border,true,true,false,false,-999,-999,-999,accept
...
```

Figure 7.19: Transformation 2 of freshmeat data

```
@relation component_training_data/Fri-Aug-13-15:50:12-WST-2004

@attribute dc:description real
@attribute swv:detail real
@attribute dc:publisher real
@attribute dc:date real
@attribute swv:licence real
@attribute swv:devStatus real
@attribute swv:devLanguage real
@attribute swv:operatingSystem real
@attribute swv:memory real
@attribute swv:diskSpace real
@attribute swv:price real
@attribute result {accept,reject}

@data
10,0,10,5,10,10,-999,0,-999,-999,-999,accept
0,0,10,5,10,0,0,-999,-999,-999,-999,accept
0,5,10,5,0,0,0,10,-999,-999,-999,accept
0,0,10,5,10,-999,-999,-999,-999,-999,-999,accept
5,5,10,0,0,-999,-999,-999,-999,-999,-999,accept
0,0,10,5,0,-999,-999,-999,-999,-999,-999,accept
0,0,10,5,10,10,0,0,-999,-999,-999,accept
...
```

Figure 7.20: Transformation 3 of freshmeat data

```
@attribute swv:detail real
@attribute dc:publisher real
@attribute dc:date real
@attribute swv:licence real
@attribute swv:devStatus real
@attribute swv:devLanguage real
@attribute swv:operatingSystem real
@attribute swv:memory real
@attribute swv:diskSpace real
@attribute swv:price real
@attribute result {accept,reject}

@data
50,0,10,5,10,10,-999,0,-999,-999,-999,accept
0,0,10,5,10,0,0,-999,-999,-999,-999,accept
0,33,10,5,0,0,0,10,-999,-999,-999,accept
0,0,10,5,10,-999,-999,-999,-999,-999,-999,accept
25,33,10,0,0,-999,-999,-999,-999,-999,-999,accept
...
```

Figure 7.21: Transformation 4 of freshmeat data

```
@relation component_real_data/Thu_Jan_01_10:52:52_WST_2009

@attribute dc:description real
@attribute swv:detail real
@attribute swv:licence {Affero_General_Public_License,Aladdin_Free_Public_License_(AFPL),Apple_
Public_Source_License_(APSL),Copyback_License,DFSG_approved,Eclipse_Public_License,Eiffel_
Forum_License_(EFL),Free_For_Educational_Use,Free_For_Home_Use,Free_for_non-
commercial_use,Free_To_Use_But_Restricted,Freely_Distributable,Freeware,Netscape_Public_
License_(NPL),Nokia_Open_Source_License_(NOKOS),Academic_Free_License_(AFL),Adaptive_
Public_License_(APL),Artistic_License,BSD_License_(original),BSD_License_(revised),Common_
Development_and_Distribution_License_(CDDL),Common_Public_License,GNAT_Modified_GPL_
(GMGPL),GNU_Free_Documentation_License_(FDL),GNU_General_Public_License_(GPL),GNU_
Lesser_General_Public_License_(LGPL),Guile_license,IBM_Public_License,MIT/X_Consortium_
License,MITRE_Collaborative_Virtual_Workspace_License_(CVW),Mozilla_Public_License_(MPL),
Open_Software_License,Perl_License,Python_License,Q_Public_License_(QPL),Ricoh_Source_Code_
Public_License,SUN_Public_License,W3C_License,zlib/libpng_
License,OSI_Approved,Other/Proprietary_License,Other/Proprietary_License_with_Free_Trial,
Other/Proprietary_License_with_Source,Public_Domain,Shareware,SUN_Binary_Code_License,
SUN_Community_Source_License,The_Apache_License,The_Apache_License_2.0,The_CeCILL_
License,The_Clarified_Artistic_License,The_Latex_Project_Public_License_(LPPL),The_Open_Content_
License,The_PHP_License,Voxel_Public_License_(VPL),WTFPL,Zope_Public_License_(ZPL),error,-999}
@attribute swv:devStatus {1_-_Planning_(disabled_category),2_-_Pre-Alpha,3_-_Alpha,
4_-_Beta,5_-_Production/Stable,6_-_Mature,error,-999}
@attribute swv:devLanguage {Ada,APL,ASP,Assembly,Awk,Basic,C,C#,C++,Clipper,Cold_Fusion,Common_
Lisp,Delphi,Dylan,Eiffel,Emacs-Lisp,Erlang,Euler,Euphoria,Forth,Fortran,Gambas,Groovy,Haskell,J2ME,
Java,JavaScript,Lisp,Logo,Lua,ML,Modula,Object_Pascal,Objective_C,OCaml,Other,Other_Scripting_
Engines,Pascal,Perl,PHP,Pike,PL/SQL,Pliant,PROGRESS,Prolog,Python,REALbasic,Rebol,Rexx,Ruby,
Scheme,Simula,Smalltalk,SQL,Tcl,Bash,TCSH,Unix_Shell,Visual_Basic,XBasic,YACC,Zope,error,-999}
@attribute swv:operatingSystem {BeOS,MacOS,MacOS_X,MS-DOS,Windows,Microsoft,OS_
Independent,OS/2,Other_OS,PalmOS,AIX,BSD,GNU/Hurd,HP-UX,IRIX,Linux,Other,QNX,SCO,
SunOS/Solaris,POSIX,SymbianOS,Unix,error,-999}
@attribute result {accept,reject}

@data
0,2,GNU_General_Public_License_(GPL),5_-_Production/Stable,-999,OS/2,reject
0,0,GNU_Lesser_General_Public_License_(LGPL),4_-_Beta,C,Linux,reject
0,0,GNU_General_Public_License_(GPL),-999,-999,-999,reject
0,0,Other/Proprietary_License_with_Free_Trial,-999,-999,-999,reject
0,0,Free_for_non-commercial_use,-999,-999,-999,reject
...
```

Figure 7.22: Transformation 5 of freshmeat data

Comparison of Transformations

Development of the transformations was an iterative/exploratory process. The hypothesis was that each successive transformation would improve recall and relevance. A range of scenarios were used as cases for testing the transformations. The following results are representative of the results seen across the scenarios and come from the emailer software scenario.

Trans.	# matches	Description
T1	1	Simple match on value or range, equivalent to original work Maxville et al (2004b)
T2	-	As above, with a loosening to include 'close' values as acceptable
T3	2	Converts the comparison to a similarity score between 0 and 10.
T4	-	Similar to T3, but uses a different calculation for longText
T5	5	Same as T3, but ontology attributes can be abstracted

Table 7.15: Transformations used

The case study scenario is to source email software, written in Java, to run on a Linux platform. There are a total of nine attributes specified, given in the ideal specification (Figure 7.23). The results of using transformations T1, T3 and T5 are compared. The first iteration produced no matches, due to missing data in the OSS repository (e.g. price). Iteration 2 uses a loosened ideal specification, and produced one match. Attributes removed in iteration 2 were `price`, `memory` and `diskSpace`. A third iteration was undertaken, as this was not a large enough shortlist. On iteration 3 the ideal specification in Figure 7.24 was used and produced five candidates.

At this point there were five items to choose between with some overlap in what each transformation identified. The original XML specification for each of these items shows that T5 found five good possibilities, most of which were missed by T1 and T3. T1 matched one item, and T3 found two. As the only difference between T3 and T5 is ontology abstraction, this supports the use of ontologies as a way to enhance the selection process. T1 is equivalent to a database search and clearly misses many suitable items.

```
<?xml version="1.0"?>
<Description xmlns="http://www.scis.ecu.edu.au/swvML/"
        xmlns:dc="http://purl.org/dc/elements/1.0/"
      xmlns:swv="http://www.scis.ecu.edu.au/swvML/" >
 <dc:detail type="mandatory">Internet email</dc:detail>
 <swv:devStatus type="mandatory">5 - Production/Stable
 </swv:devStatus>
 <swv:licence type="mandatory">GNU General Public License
  (GPL)</swv:licence>
 <swv:price type="mandatory" min="25" max="50">40
 </swv:price>
 <swv:technical>
 <swv:devLanguage type="mandatory">Java</swv:devLanguage>
  <swv:operatingSystem type="mandatory">Linux
  </swv:operatingSystem>
 <swv:systemRequirements>
 <swv:memory type="mandatory" min="15" max="50">20
 </swv:memory>
 <swv:diskSpace type="mandatory" min="30" max="50">40
 </swv:diskSpace>
 </swv:systemRequirements>
 </swv:technical>
 <dc:date type="mandatory" min="2003-01-01"
 max="2005-03-01">2005-01-10</dc:date>
</Description>
```

Figure 7.23: Initial ideal specification

```
<?xml version="1.0"?>
<Description xmlns="http://www.scis.ecu.edu.au/swvML/"
        xmlns:dc="http://purl.org/dc/elements/1.0/"
     xmlns:swv="http://www.scis.ecu.edu.au/swvML/" >
<dc:detail type="mandatory">Internet email</dc:detail>
<swv:devStatus type="mandatory">5 - Production/Stable
</swv:devStatus>
<swv:licence type="mandatory">GNU General Public License
(GPL)</swv:licence>
<swv:technical>
<swv:devLanguage type="mandatory">Java</swv:devLanguage>
  <swv:operatingSystem type="mandatory">Linux
  </swv:operatingSystem>
</swv:technical>
<dc:date type="mandatory" min="2003-01-01"
max="2005-03-01">2005-01-10</dc:date>
</Description>
```

Figure 7.24: Loosened ideal specification

7.6 Spiral 4 Evaluation

Beginning with the application developers, the Win conditions are to see benefit of the strategies, justifiable results and low overheads. Although the processing involved in the CdCE shortlisting is complex, this is not what the user sees. From a user perspective, they need to create the ideal specification and consider the choice of transformation. At this point there has been improvement in terms of usability through the provision of scripts and post-processing of the data. In all cases, the files produced provide full documentation of the rules applied to generate the shortlists, an issue for both the application developers and for quality assurance.

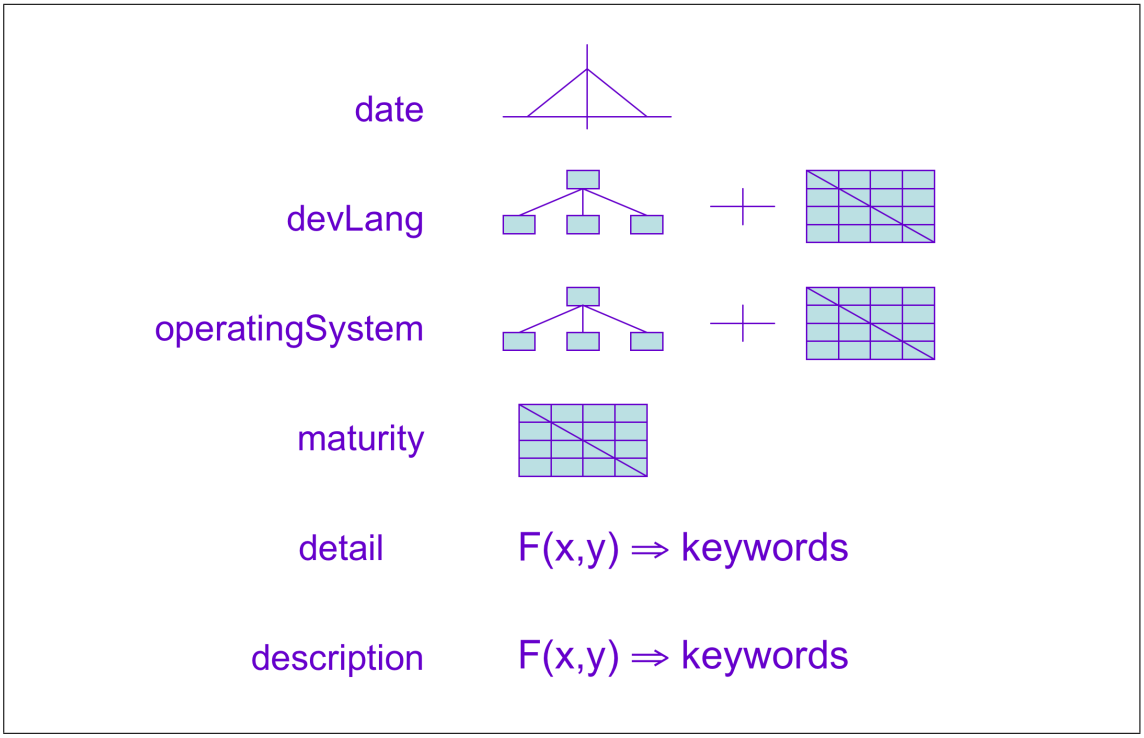


Figure 7.25: Representation of the different calculations for matches based on attribute type

A condition on Spiral 4 is that the approach provide more than a 'database query'. Figure 7.25 shows the various types of comparisons made, depending on the attribute type. These include the formulae for `numeric` and `date` attributes, and keyword matching for `longText`. `ontology` attributes utilise the ontology, the thesaurus, a distance matrix (in some cases) and level abstraction. Each of these helps to draw out maximum information based on the attribute type.

The description of the shortlisting provides information for developers and brokers

to understand how their products are filtered by the Process, and may see more benefit in giving complete metadata. For the brokers, it is clear that openness about their internal schemata and providing back-end access to the metadata is a way to improve their interaction with automated tools.

From the academic perspective, the discussion in this Chapter has justified the data representation used, and the refinement process that was undertaken. The flexibility of the shortlisting and tools has been exercised during the development as most aspects have evolved. Interfaces between the programs developed are all XML, ARFF (Weka input) or text (Weka output) using a pipe/filter model for processing. This provides flexibility and performance.

The work in Spiral 4 was published in PEECS (Maxville, 2005) and, through its inclusion in all later publications, has received multiple peer reviews. From the discussion above, the Win conditions have been satisfied for Spiral 4.

7.6.1 Spiral 4 Goals

The following discussion refers to the goals and questions listed in Table 7.16. A discussion of the performance of the work with respect to the questions is provided under each goal heading.

Quality

Comparisons between the five transformations indicate better recall and relevance in T3 and T5 over T1, which is equivalent to the representation used in Spiral 3. Section 7.5 provided an example of the transformations and how each performed on the same data. The handling of missing data has improved training accuracy and eliminated the false positives introduced by the way missing values were handled in Spiral 3. Clear descriptions of all the transformations and the missing data handling have been provided, along with information on the related ontology and distance matrix. Scripts have been developed to handle all processing, including a convention for unique naming of results and storing all files from each run. This ensures all key data is available for analysis and re-running if required.

SPIRAL 4	**Purpose** **Issue** **Object** **Context**	**Evaluate** **effectiveness of** **strategies for data representation** **Spiral 4**	**Results**
Goal 4A	Focus	**Quality**: *Enhance shortlisting for more accurate results*	
	Viewpoint	Quality Assurance personnel	
	Q4A1	Has Spiral 4 improved results?	YES
	Q4A2	Are the updates well documented?	YES
	Q4A3	Is the process repeatable?	YES
Goal 4B	Focus	**Usability**: *Provide tools and knowledge base for users*	
	Viewpoint	Application developer	
	Q4B1	Is the shortlisting easy for the user to understand?	YES
	Q4B2	Has the work been tested on real world examples?	YES
Goal 4C	Focus	**Intelligence**: *Apply ontologies and knowledge management to shortlisting*	
	Viewpoint	Application developer	
	Q4C1	Has an ontology been used?	YES
	Q4C2	Has knowledge management been used?	YES
	Q4C2	Has missing data been handled?	YES
Goal 4D	Focus	**Innovation**: *Include innovative knowledge management and missing data treatment*	
	Viewpoint	Academia	
	Q4D1	Have innovations been developed?	PART
Goal 4E	Focus	**Dynamics**: *Allow for update and substitution of knowledge base*	
	Viewpoint	Application developer	
	Q4E1	Can alternative knowledge bases be used?	YES
	Q4E2	Can the knowledge base be easily updated or modified?	YES
Goal 4F	Focus	**Reuse**: *Where possible make use of existing code and artefacts*	
	Viewpoint	Application developer	
	Q4F1	Has the work reused external resources?	YES

Table 7.16: GQM Summary - Spiral 4

Usability

As with Spiral 3, the user is insulated from the complexity of the software. From the user perspective, there is the ideal specification, the parameters for missing data, ontology levels and transformation (or use defaults) and the output shortlist in XML. As the user is an application developer, some familiarity with scripts, parameters and XML is likely.

The shortlisting in Spiral 4 uses the freshmeat dataset, with an initial load of 33,262 projects, later updated to 41,886 projects. After initial testing of the modifications, all work was carried out in case study mode - utilising real world scenarios.

Intelligence

The freshmeat trove was adopted as the ontology for this project. It includes nine categories which in five cases form a hierarchy (e.g. OperatingSystem). The ontology implementation is described in Section 7.3.1.

The representation of data has been enhanced by the re-alignment of attribute types into `numeric`, `date`, `freeText`, `longText` and `ontology`. A corresponding set of comparison routines were developed to apply the appropriate transformations based on attribute type. These transformations can be selected as parameters to explore and select the most appropriate.

Missing data is handled by substituting the value -999 for all attribute types. This value can be changed as a parameter. The benefit of using the missing value is that the classifier can be trained to respond to it appropriately. The user can choose for this to be as a match, a mismatch or a borderline case.

Innovation

Each of the approaches applied in Spiral 4 is based on existing techniques. However, the combination of approaches into such a system is novel. The data representation builds on the novel work from the previous spiral.

Dynamics

To use a different knowledge base (ontology) it needs to be supplied in XML to match the structure of the provided ontology file. This structure is displayed in Figure 7.6.

The XML files for ontology and distance metric can be updated easily using a text editor. As described in Section 7.3.1, new entries can be added to the ontology by adding a block of XML and linking to the existing data. While specific tools for this manual updating have not been developed, the OntologyCheck tool outputs blocks of pre-populated XML to assist with updates to the ontology.

Reuse

The freshmeat project list and trove are used as the repository and ontology, respectively. Most of the code developed for Spiral 3 was able to be easily reused for Spiral 4 as a result of the use of object-orientation, pipe/filter architecture and external files for input/output.

7.7 Spiral 4 Review and Planning

This Chapter addresses data representation as dealt with in Spiral 4. The Win conditions were satisfied by the strategies applied to the shortlist of candidates. The GQM was positive for all questions except that Question Q4D1 (**innovation**) was not fully satisfied as the AI techniques were not innovative in themselves. However, this application of existing techniques is regarded as a positive in Q4F1 (**reuse**).

The approaches in Spiral 4 were peer reviewed for the PEECS Symposium (Maxville, 2005).

Future work would include the conversion of additional repositories. During this Spiral, some time was spent trying to work with SourceForge. The SourceForge repository has been provided in two forms: a relational database and a text file. The text file is a summary of the projects and does not include some of the useful data to be seen on the SourceForge site. The PostgreSQL database is over 12Gb, which brought its own issues to handle file size. Unfortunately, the data model for the SourceForge database was difficult to work with so this task was abandoned.

Planning for Spiral 5 turned to the test generation and the remaining steps of the CdCE Process. A commitment was made to continue with the classifier-based approach and for T5 (Level 2) as the most effective of the five transformations.

7.8 Post-Spiral Update

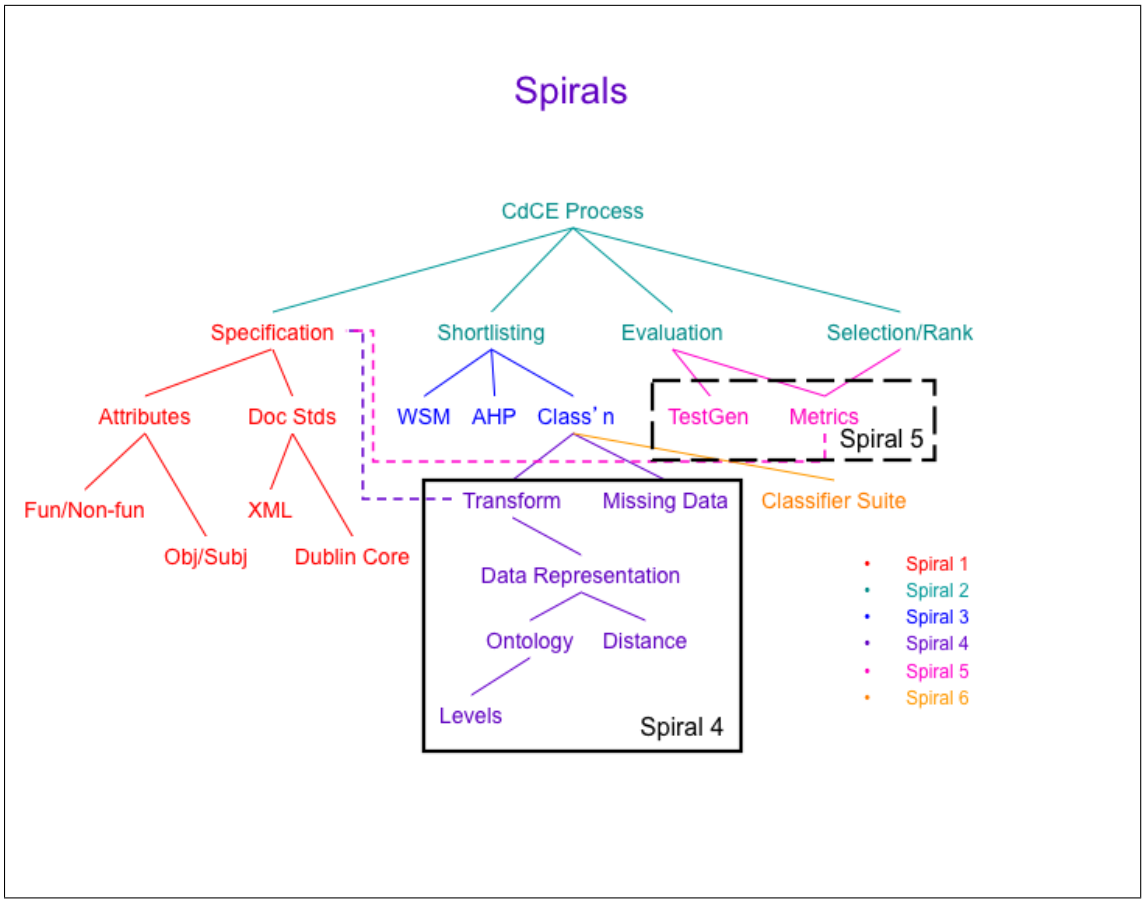


Figure 7.26: Spiral 4 and later updates made to the Spiral outcomes

Figure 7.26 provides context for topics addressed in Spirals throughout the investigation. The solid border groups aspects of the investigation which were the target of, or resulted from, Spiral 4. The dashed outline box on Spiral 5 indicates the updates to the data representation subsequent to Spiral 4. This was principally the side effects of the testing and evaluation approaches developed in Spiral 5. Work on the evaluation fleshed out the metrics and related attributes in the specification, resulting in minor changes to the specification and the related files from this Spiral. Scripts and workflow for generating and applying the classifier were reused for ranking in Spiral 5.

7.9 Summary

This chapter has provided a description of Spiral 4, focussing on data representation and the development and evaluation of five transformations on five base attribute types. It has

discussed what was aimed to be achieved, and the approaches used. The implementation was illustrated with examples of running the tools that have been developed. Results and strategies were evaluated and found to be effective and met stakeholder Win conditions. The project will now build on this work through the evaluation and testing in Spiral 5 (Chapter 8).

Key contributions of Spiral 4 were the integrated and automated use of a range of data representations to improve recall and relevance in shortlisting. The approach provides flexibility in the application of the transformations and in the handling of missing data. The data representation is driven through five datatypes, keeping to simple and type-appropriate comparisons. In text-based fields, the transformations increasingly make use of deeper information including a supporting knowledge base (ontology) and distance matrix for comparisons.

Chapter 8

Testing and Evaluating Candidates

This chapter discusses the approach used for the functional evaluation and ranking of shortlisted candidates in the CdCE Process. Steps 1 and 2 have been the focus of the investigation thus far (see Figure 8.1). The scope of this Spiral is Steps 3-8. The evaluation takes place in Steps 3 to 6 and includes the generation, adaptation and execution of tests based on the ideal specification. In Steps 7 and 8, the results are used to rank the candidates according to metrics included in the ideal specification in order to report on the complete CdCE evaluation.

There are multiple products of this Spiral. The first is in the evaluation metrics, their definition and application. A second product is the process, tools and supporting documentation for test generation from Z specification. Adaptation and execution depend on the testing environment, and are left to the user to implement. Finally, the ranking is implemented via the reuse of the classifier approach used for Step 2. This work is the basis for contribution C6, a metrics-driven evaluation featuring context-based testing.

The goals for Spiral 5 are given in Table 8.1. As with the overall Process, this Spiral

SPIRAL 5	GOALS
Quality	Provide structured, repeatable approach to testing and evaluation, drawing on literature
Usability	Provide tools to suit user needs and automate testing and evaluation
Intelligence	Use AI and knowledge management for testing and evaluation
Innovation	Consider novel approaches to testing and evaluation
Dynamics	Provide flexibility for testing and evaluation
Reuse	Where possible make use of existing code and artefacts

Table 8.1: Goals for Spiral 5

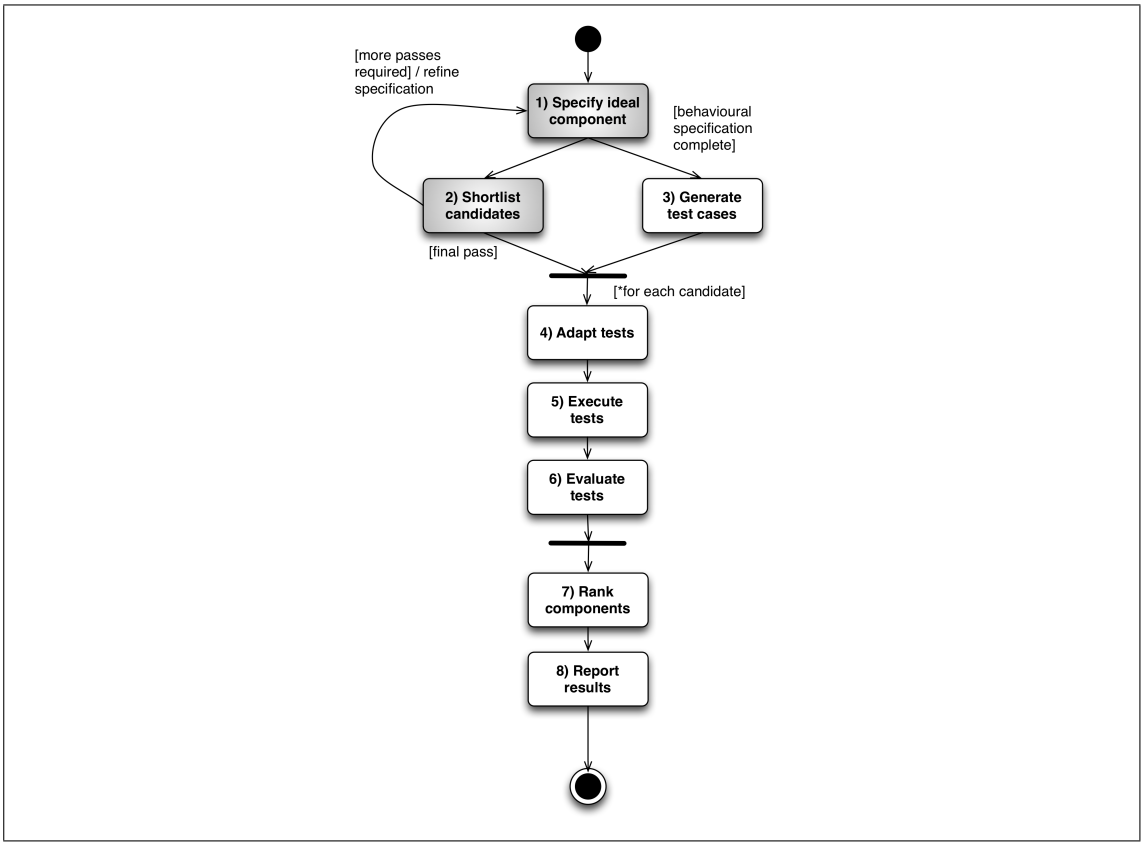


Figure 8.1: Steps implemented in Spiral 5, with out of focus steps greyed

aims to support **quality** through a formal, structured approach and allows flexibility for selection tasks and local conventions (**dynamics**). Tool support for the repetitious aspects of the Process will be developed to aid **usability**. Where possible, machine **intelligence** will be applied in Steps 3 to 8, utilising artificial intelligence and knowledge-based techniques. A balance between **reuse** of existing and creating new software is required, with novelty around new strategies, building from existing theory and software (**innovation**). These goals are revisited in the Spiral evaluation described in Section 8.6.

This chapter is structured with separate sections grouping related items that have been implemented:

- Section 8.2: Metrics (Step 3)
- Section 8.3: Test Generation (Step 3)
- Section 8.4: Adaptation and Execution (Steps 4-6)
- Section 8.5: Ranking and Reporting (Steps 7-8).

After providing the overview to Spiral 5, the Approach, Implementation and Examples

(results) are covered for the four subtopics. The implementations for Steps 3-8 of the CdCE Process are then evaluated and reviewed in the final sections. The evaluation is as a whole, with attention given to each subtopic.

8.1 Spiral 5 Overview

In this Spiral, the focus is on the testing and evaluation of the candidates. The use cases addressed in this Spiral are: **Select Component**, **Revisit Selection**, **Reuse Tests** and **Assess Selection** (Figure 8.2). Application developers will use the products of this Spiral for initial selection of components, and if repeating the selection, as the target system evolves. There is also the possibility of reusing the tests and adaptation in other parts of the system development, for example, integration and regression testing. As in the other Spirals, there is an awareness of the information required for assessing the quality of the development process, and that should be automatically generated.

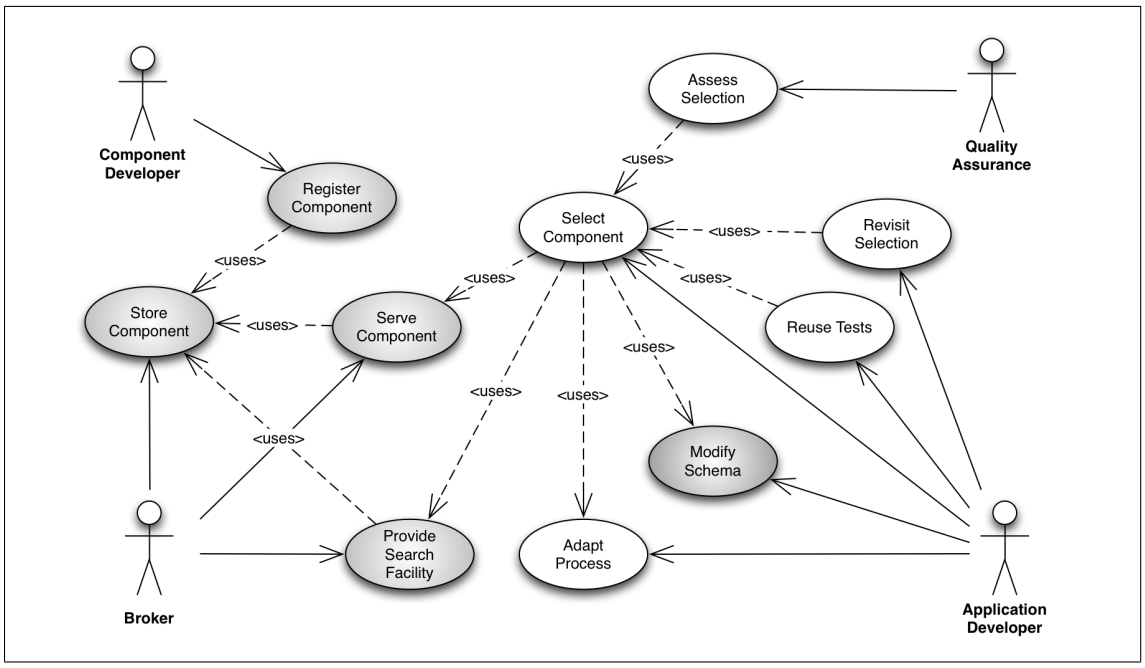


Figure 8.2: Use cases for the testing and evaluation steps of the CdCE Process (those not in the scope for this Spiral are greyed)

Considering the stakeholder view of this Spiral, the application developers require that the approach to testing and evaluation match their needs and be robust and easy to use. Academia require valid testing and evaluation approach, with a strong basis on existing testing theory. The stakeholder evaluation criteria are provided in the table of

Win conditions (Table 8.2).

Stakeholder	Win Conditions
Application Developers	Test generation matches specification. Automation is effective. Evaluation produces suitable candidates.
Component Developers	Know how their component is assessed and compared.
Academia	Test generation techniques are valid. Assessment criteria are relevant.

Table 8.2: Win conditions for stakeholders (Spiral 5)

8.1.1 Context

This Spiral extends the specification, Process, strategies and tools that have been developed in Spirals 1 to 4. Demonstration of the use of evaluation and reporting used a range of repositories, providing project metadata and access to associated executables for testing. The case study scenario is the selection of a calculator component which has been used throughout the investigation to trial ideas.

The following Sections group the concepts relevant to Steps 3 to 8 of the CdCE Process, then discuss the approach, implementation and examples of each. The first grouping is metrics, which are used to drive the evaluation and are a common thread throughout these Steps. The second focus is test generation, where the Z specification is used to create a suite of tests. Once the tests are available and a short-list of candidates has been determined, the tests can be adapted to each of the candidates and executed in the target context. This is explained in the adaptation and execution section. The final concept discussed is the approach to ranking and reporting of results.

8.2 Spiral 5 Metrics

For this project, the metrics for the functional evaluation of the candidates need to be broadly applicable to a general software problem - targeting areas affecting quality and fit of a third party component. Some of the metrics could be discovered through static means (functional fit) while others require execution of the program, such as performance against a given test suite. The requirement for this work is that the contextual suitability will also be assessed. When shortlisting candidates, context referred to the non-functional requirements relating to the target environment (e.g. operating system, framework).

For testing purposes in this investigation, context refers to the specific requirements on functionality, performance, reliability, stress or usage. Metrics are required to provide a score for each area to be included in the overall evaluation of the software to the current selection task.

8.2.1 Background

Metrics for this work are driven by the type of information of interest when assessing the 'fit' of a component. Given the ideal specification as a basis for comparison, the metrics are limited to what it enables. In preparation for test generation, the ideal specification has a high level Z specification to use as a test directive. This includes, at least, the operations and their parameters/types. The metrics can then be based on the required interfaces compared with those offered by the candidate component. The matching and excess functionality can be calculated based on these interfaces. Consideration of the type of mismatch, if any, can provide an indicator of the effort required to adapt the candidate component.

8.2.2 Approach

This project uses the SWEBOK to guide the choice of metrics (Abran et al, 2004). The formal specification for the software (ideal specification) is intended as a definition of the required functionality. With an executable component, the functional fit could be evaluated on the match of required and available interfaces. For an application, the interface measures can be abstracted to the functions provided by the software under test. It is also important to measure excess functionality, as it presents a risk in the reuse of code (Sorensen, 2004). Two metrics are used to represent functional fit and functional excess in the Process, `FFIT` and `FEXS`:

FFIT: The measure of the interfaces provided compared to the interfaces required in the ideal specification

FEXS: The measure of the interfaces provided that are not required, compared to the interfaces required in the ideal specification.

In the case of an imperfect match, the effort required to adapt the interfaces needs to be quantified and included in the overall evaluation. The measure may vary depending

on the selection task and must include an indication of the quantity and complexity of the adaptation. The adaptation process is discussed in Section 8.4. The metric indicating the effort for adaptation on each candidate is `AEFT`:

AEFT: The measure of the effort needed to adapt the interfaces offered to satisfy those required in the ideal specification.

To provide evaluation options for testing, the metrics consider the fit of the tests to the candidate - which may be different to `FFIT`. This allows the separation of whether tests have failed, or have not been able to be applied. Thus there are two metrics for the (base) testing results, `TFIT` and `TRES`:

TFIT: The measure of the tests that are able to be run compared to the number of tests in the test suite

TRES: The measure of the tests passed compared to the number of tests in the test suite.

Continuing with the dynamic evaluation, contextual tests also need to be executed and results quantified. To this end, the SWEBOK is used to consider the various types of testing that are relevant to this work: those relating to this part of the software development lifecycle are performance, reliability, stress and usage.

To measure the performance or response time for parts of the code, the `CX_P` metric is provided. In addition, there may be a need to test the reliability of the functions which are most critical in the target context, or to stress test certain functionality. The `CX_R` and `CX_S` metrics accommodate these two types of testing. In usage-based testing, a model of the expected usage guides the testing and thus more tests are executed against popular parts of the code/application (`CX_U`). The specification for all of these context-based tests is defined in Z notation, described in Section 8.3.3. As the tests are based on a subset of the original functional tests, the `TFIT` metric is considered sufficient to indicate the contextual testing fit for each candidate. The working definitions for the four context metrics are:

CX_P: The measure of the performance-based tests passed compared to the number of performance-based tests in the test suite

CX_R: The measure of the reliability-based tests passed compared to the number of reliability-based tests in the test suite

CX_S: The measure of the stress-based tests passed compared to the number of stress-based tests in the test suite

CX_U: The measure of the usage-based tests passed compared to the number of usage-based tests in the test suite.

Other relevant types of testing are regression and integration. As the aim is to have a test harness or environment modelling the target environment, all of the tests could be reused for integration testing. The reusability of the tests allows them to be re-run in future maintenance or re-selection tasks, which provides an added benefit in regression testing.

8.2.3 Implementation

The metrics identified in the previous section have been defined at an abstract level. To include them in the CdCE Process, they need to be more formally defined, and as they are quantifying the performance of the candidates, there needs to be a scale. In the implementation of the transformations for the selection classifier, an arbitrary scale of {0..10} was adopted in a 'normalising' of results (Solberg and Dahl, 2001). This scale is also used for the metrics to give consistency and make results easier to understand.

Table 8.3 shows the default definitions for metrics in the CdCE Process. These may be redefined to suit a particular selection task and are based on the general pattern for test metrics drawn from Myers (1979) where `metric = number tests passed / total number tests`.

The Process user should document any changes to these metrics. As an example, when evaluating applications instead of components, the interface adaptation is not appropriate. Instead, the effort required to access, install and get the application running, can be assessed out of 10 and used for the AEFT score. Alternatively, component brokers and developers may be interested in the impact that ease of installation has on the evaluation process, as this was an issue in the case studies in this investigation and reflected poorly on affected software.

Metric	Calculation
FFIT Functional fit	# interfaces matched / # interfaces required
FEXS Functional excess	# interfaces matched / # interfaces in component
AEFT Adaptation effort	# interfaces adapted / # interfaces required
TFIT Testing fit	# tests possible / # test cases
TRES Test result	# tests passed / # test cases
CX_P Performance testing	# performance tests passed / # performance test cases
CX_R Reliability testing	# reliability tests passed / # reliability test cases
CX_S Stress testing	# stress tests passed / # stress test cases
CX_U Usage testing	# usage tests passed / # usage test cases

Table 8.3: Evaluation metrics - default definitions

Each of the metrics is entered via the ideal specification, or omitted if not required in the current selection task. They are implemented as numeric attributes and thus need an optimal value and a min/max range to be used in Step 7. The nine metrics were added to the CdCE schema (swvML, see Table 4.9) and correspondingly into the input/parameter files for the classifier related applications.

8.3 Spiral 5 Test Generation

With the metrics for evaluation in place, the test generation supplies the tests to provide values for the metrics. Five of the metrics are derived from the test results: TRES, CX_P, CX_R, CX_S and CX_U. TRES considers the complete set of generated tests, where the other metrics are the result of focussing the tests on specific operations (methods/functions) and values.

Many aspects of the testing were decided prior to Spiral 5, or result indirectly from earlier decisions. The ideal specification utilises a Z specification to define interfaces, types and some of the logic for the software under test. A full formal specification of the required software is not required, instead a directive is used to indicate the required functionality. In addition, as third party components are being sought, there is no access to the component source code, restricting the choice of testing techniques. This limited view of the components under evaluation reinforces the applicability of black-box testing, requiring a less detailed specification. If test oracle functionality is desired, a more complete specification would be warranted. The specification also needs to be able to represent the context testing to feed into the associated metrics. Therefore it needs to allow for performance, reliability, stress and usage testing.

The test generation for this project is based on applying existing techniques, rather than creating a novel testing solution. The test generator needs to take the behavioural specification and create abstract test cases ready to apply to each of the candidates. These will be represented in XML.

8.3.1 Background

The approaches to automated testing using Z notation have three main strategies: to use Z directly for specification and test generation; to start with another model (e.g. finite state machine (FSM)) and convert to Z; or to specify with Z, then convert to a state machine for test case generation. Techniques are applied across Z and other model (state) based specification languages (e.g. VDM) and if the Z specification is converted to a state machine, FSM and graph theory-based techniques can be applied. This approach to test generation was first discussed by Hall and Hierons (1991), and applied to VDM by Dick and Faivre (1993) and to Z by Stocks and Carrington (1993). Their work extended the category-partition approach by developing a framework that defined the test and test suites structures using Z.

Helke et al (1997) generate tests from the Disjunctive Normal Form (DNF) representation of a Z specification, then encode it with Isabelle/HOL to show how existing test generation techniques can make use of a state-of-the-art theorem prover. Carrington et al (1998) continued their research to take Object-Z specifications and generate a state machine using the Dick and Faivre approach. They then choose test sequences and encode these as testgraphs (subsets of the original state machine) which can be fed into their Classbench testing tool. Later work in this area by Burton (2000) transforms statecharts and reactive components to Z, and then automates partitioning, test generation and the creation of test scripts for Ada applications.

Burton adopts fault-based testing approaches as input to test generation (along with partitioning). Horcher and Mikk (1996) provide guidelines for inserting partition information into a specification based on the input variable types. The enhanced specification is then converted to DNF for tests to be generated. The evaluation of test results is facilitated using a schema compiler to generate executable code from the Z, which provides test oracle functionality.

Another approach to test generation with Z is to combine the DNF partitioning and

the classification-tree method (Singh et al, 1997) in order to improve on the unstructured test cases generated by DNF partitioning. A classification-tree creates a hierarchy of the input domain, allowing the inputs to be ordered in terms of precedence. The combination of the classification tree and the DNF representation of the specification results in refined test cases which are the conjugation of the tree test values and the DNF operations.

In a boundary testing approach Legeard et al (2002) combine Z and B specifications to test every operation of a system at every boundary state. They calculate boundary goals from the pre and post predicates in the formal model, and then use their BZ-TT tool to simulate the execution of the specification to find a sequence to exercise each boundary goal. The output of the simulation is a test pattern file which is converted into executable test scripts.

Chang et al (2000) look at incorporating usage profiles into their test generation. They convert an Object-Z specification into an Enhanced State Transition Diagram (ESTD). The state model is used to derive the operation scenarios, as in the FSM approaches above, but in their approach, each transition can be given a weighting to simulate usage patterns.

The techniques that have been described for test generation are only an indication of the options available given a Z specification and inform the approach taken in this project, and potential future work.

8.3.2 Approach

Given a full Z specification, any of the test generation approaches in the previous section could be applied. However, as the specification in the CdCE Process is less detailed, this affects the choice of approach. For this investigation, the strategy is to use Z directly for test generation (i.e. not converting to another model). As type and partition information are available, it is possible to use the category-partition approach. If this exhaustive approach became impractical, information could be added to the specification to indicate restrictions on partitions and allow a DNF approach. However, the restriction of testing to expected and valid combinations of values may reduce the effectiveness of the testing as unexpected and/or invalid combinations may expose errors.

A common task in testing is to go through a sequence of operations where the output state of an operation becomes the input state for the next operation. This may be for

usage based tests, or to initialise the software into the required state for valid testing of certain functions. For example, a banking system would require transactions to occur prior to being able to test functionality relating to exceeding daily transaction limits. Another example is needing to enter an incorrect password three times to test password lock-out mechanisms.

As the rest of the specification has been represented in Z, the context based tests are also declared using Z notation. A schema matching each of the CX metrics is used to extend the base tests. Thus the `CX_P` metric is generated from the results of the tests based on the CX_performance schema. Sequences of operations are commonly used in testing and are represented by CX_sequence. These sequences can be referred to in the other four CX schemas and thus be included in the test suite.

An approach of interest is usage profiles (Chang et al, 2000). Although not applying that approach in this work, usage data has been collected for some of the case studies with consideration of how they might be included in the testing. The context schemas allow for sequences to be defined according to usage, and test data to reflect expected types of data (for example, in email software a sequence might be made up of: checkMail, openMessage, forwardMessage, sendMessage).

8.3.3 Implementation

The behavioural specification is modelled in Z Notation and has tools to allow automated processing of the specification for test generation. The specifications themselves are encoded as XML using the LaTeX standard, which can be viewed in LaTeX as graphical Z schemas. The specification is included in the ideal specification within `<techDescription><Zspec>` tags. Five Z schemas are used to represent the behavioural context: CX_performance, CX_reliability, CX_stress, CX_usage, CX_sequence and CX_environment[1].

The specification is distilled to draw out key information used in test generation. The parts of the specification that are filtered from the Z specification are: types, partitions, states, operations and variables (changed and unchanged). These are held in a simple

[1]Earlier documents used slightly different names and focii for these schemas (Maxville et al, 2003b). CX_values is now in the ideal specification under separate tags and CX_environment may be developed in the future to model the environment of the software under test.

text file which is parsed into a Java program. The Specification object collates the types, partitions and states along with the operation schemas for each interface. A combinatorial approach is then used to apply each partition to each type for each operation schema, which is output as an XML file. Pseudocode for base test generation is given in Figure 8.4. Although it is given as nested iteration, the final implementation utilises recursion.

To illustrate, given a Z schema, Example, and an operation on that schema, Op1, the following definitions can be distilled to give the information in Figure 8.3:

Z notation:

Example

$$var1 : Type1$$
$$var2 : Type2$$

Op1

$$\Delta Example$$
$$inOne? : Type1$$
$$inTwo? : Type1$$

$$var1' = op1(inOne?, inTwo?)$$
$$var2' = var2$$

```
STATE Example
STATE Example var1 Type1
STATE Example var2 Type2
%
% Operations
OPERATION Op1
OPERATION Op1 state Example
OPERATION Op1 IMPORT inOne Type1 inTwo Type1
OPERATION Op1 changedState var1
```

Figure 8.3: Distilled version of Example schema and operation

The distilled information is read into TestGen, which generates the tests according to the pseudocode in Figure 8.4. Values can then be assigned for each partition, either manually, or through a list or link in the ideal specification. This provides the full functional test suite which serves as the base test set.

Currently the test directive is a text file distilled from the Z specification. While it is for internal use, it could easily be converted into XML - which would make it more robust, but less readable by humans. The distiller and test generator, TestGen are implemented

```
Base test set generation:

     For each operation (O_i)
          For each parameter (P1_i)
               For each partition (P2)
                   Output test case (O, P1, P2)
```

Figure 8.4: High level pseudocode for generating base test set

in Java and operate as filters. The class diagram for TestGen is given in Figure 8.5. Objects in the program match to the key items: specification, made up of state schema and operation state schemata. The state schema has a variable list, with each variable of a particular type, and each type containing partitions. In addition, operation state schemata can have import and export variables.

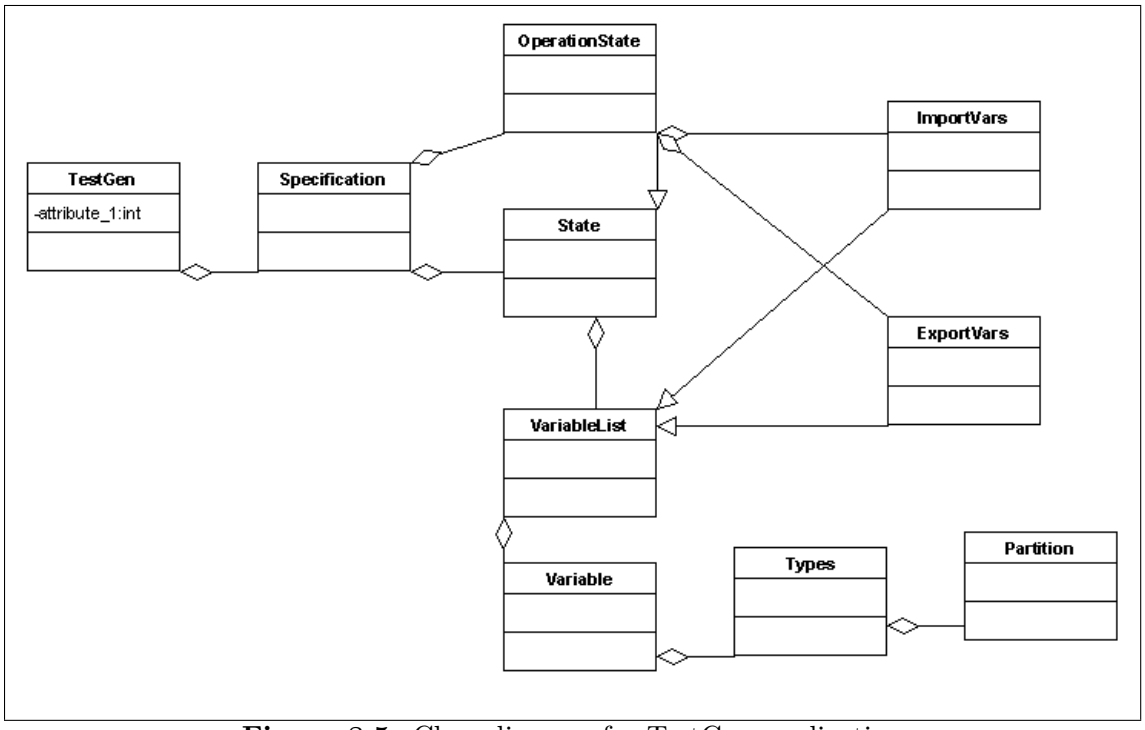


Figure 8.5: Class diagram for TestGen application

Context

For context testing, the CX schemas are used to target tests drawn from the base set for additional testing. The first of these is CX_performance, which results in a value for `CX_P`. In this schema, operations and sequences pass the performance tests if they

complete within the required time. The time unit is declared and the schema maps operations to their respective required times.

CX_performance

$$\begin{aligned}
&perfOp : operation \pfun R \\
&perfSeq : sequence \pfun N \\
\hline
&perfOp = \{(Op1, 1), (Op2, 2)\} \\
&perfSeq = \{(Seq1, 3)\}
\end{aligned}$$

To provide reliability testing, the CX_reliability schema lists operations which will be tested more rigourously. Inclusion in the list will imply a doubling of testing, which can be increased using an integer for multiplying the amount of additional testing (2 = double). Similarly, additional focus can be put on specific partitions within the CX tests. `CX_R` is the number of these tests that are passed, divided by the total number of reliability tests.

CX_reliability

$$\begin{aligned}
&relOp : operation \pfun N \\
&relPart : partition \pfun N \\
&relSeq : sequence \pfun N \\
\hline
&relOp = \{(Op1, 1), (Op2, 2)\} \\
&relPart = \{(Type1_Part1, 2), (Type1_Part2, 2)\} \\
&relSeq = \{(Seq1, 3)\}
\end{aligned}$$

Stress testing is based on repeatedly calling operations at a given rate (use definition). The CX_stress schema maps operations or sequences of operations to a rate that indicates how often it will be called in a unit of time (seconds).

CX_stress

$$\begin{aligned}
&stressOp : operation \pfun N \\
&stressPart : partition \pfun N \\
&stressSeq : sequence \pfun N \\
\hline
&stressOp = \{(Op1, 100), (Op2, 50)\} \\
&stressPart = \{(Type1_Part1, 2), (Type1_Part2, 2)\} \\
&stressSeq = \{(Seq1, 30)\}
\end{aligned}$$

While usage-based testing is similar to reliability, it is driven by the expected usage of the software. This may be the operations, the sequences of operations and/or the values (through the partitions).

Schema	Base Tests	CX_Sequence Tests	Partitions	Metric
Base Specification	all	-	all	TRES
CX_Performance	selected	selected	selected	CX_P
CX_Reliability	selected	selected	selected	CX_R
CX_Stress	selected	selected	selected	CX_S
CX_Usage	selected	selected	selected	CX_U

Table 8.4: Relationship between schemas, tests and metrics

CX_usage

$$useOp : operation \pfun \mathbb{N}$$
$$usePart : partition \pfun \mathbb{N}$$
$$useSeq : sequence \pfun \mathbb{N}$$

$$useOp = \{(Op1, 1), (Op2, 2)\}$$
$$usePart = \{(Type1\_Part1, 2), (Type1\_Part2, 2)\}$$
$$useSeq = \{(Seq1, 3)\}$$

Test sequences may be implemented as a series of operations. This is future work for the automated system. In Z, sequences of operations could be encoded as:

$$Seq1 = Op1(3, 4) \gg Op2(5) \gg Op3(6)$$

```
Context test set generation:

For each context schema (CX_i)
     For each operation (O_i)
           Apply operation weighting (O_i,w)
           For each parameter (P1_i)
                 Apply parameter weighting (P1_i,w)
                 For each partition (P2_i)
                      Apply partition weighting (P2_i,w)
                     Output test case (O_i, P1_i, P2_i)
```

Figure 8.6: High level pseudocode for generating context test sets

Context tests are in addition to the base tests, and will repeat some of the base tests to focus on operations, partitions and sequences. The context tests each use similar algorithms, as shown in Figure 8.6. The total number of tests will be the sum of the base tests, plus the number of tests selected for each of the context schemas (see Table 8.4).

8.3.4 Example

In this example, the scenario is the selection of a calculator component to include in an application. There is one type of data, the BigReal, with five partitions: positive, negative, zero, large positive and large negative. Additional partitions could be SMALLPOS and SMALLNEG, but they are not used in this example.

Types

[BigReal]

Partitions

BigReal ::= POS | NEG | ZERO | LARGEPOS | LARGENEG

The state schema for a full calculator specification is shown in FullCalculator below, and adapted from Barden et al (1994). The full calculator is modelled in detail, including memory, error, mode and an expression tree for complex calculations.

FullCalculator

current : BigReal
memory : BigReal
error : errorState
mode : modeType
expression : expressionTree

This example scenario uses a reduced specification of a calculator to focus on the operations and their impact on the state variable: current. Memory is also included as a state variable, but has not been affected in the operations given in this specification.

Schema

Calculator

current : BigReal
memory : BigReal

In this case the application developer is interested in the operations CalcAdd, CalcMultiply, CalcDivide, CalcSine and CalcPower. In each operation schema, inOne? and inTwo? refer to input variables. Current' and memory' indicate the values of the state variables after the operation. ΔCalculator indicates there will be a change in state (current, memory) during the operation.

CalcAdd

ΔCalculator
inOne? : BigReal
inTwo? : BigReal

$current' = add(inOne?, inTwo?)$
$memory' = memory$

CalcMultiply

$$\Delta Calculator$$
$$inOne? : BigReal$$
$$inTwo? : BigReal$$
$$current' = multiply(inOne?, inTwo?)$$
$$memory' = memory$$

CalcDivide

$$\Delta Calculator$$
$$inOne? : BigReal$$
$$inTwo? : BigReal$$
$$current' = divide(inOne?, inTwo?)$$
$$memory' = memory$$

CalcSine

$$\Delta Calculator$$
$$inOne? : BigReal$$
$$current' = sine(inOne?)$$
$$memory' = memory$$

CalcPower

$$\Delta Calculator$$
$$inOne? : BigReal$$
$$inTwo? : BigReal$$
$$current' = power(inOne?, inTwo?)$$
$$memory' = memory$$

This completes the specification used in the functional testing for this example. A second version of each operation can be defined to allow a sequence of calculations to update the current value. These have one less input variable, with InOne? replaced by current, and inTwo by inOne.

CalcAddCurrent

$$\Delta Calculator$$
$$inOne? : BigReal$$
$$current' = add(current, inOne?)$$
$$memory' = memory$$

CalcSineCurrent

$$\Delta Calculator$$
$$current' = sine(current)$$
$$memory' = memory$$

The specification is distilled and transformed into the test directive in Figure 8.7. The directive

retains the types, partitions, state schema and operations. It begins with the BigReal types and the partitions for that type. Then the state and associated variables (memory and current) are declared. Following the state are the calculator operations that are of interest: addition, multiplication, division, sine and power. The import parameters represent the interface for each operation, altering the state variable, current, with the result of the calculation. The test directive is parsed into the TestGen program according to the class diagram in Figure 8.5.

```
% Types
TYPE bigReal
%
% Partitions
PARTITION bigReal POS 30 1000 4.94065645841246544E-324
PARTITION bigReal NEG -30 -1000 -4.94065645841246544E-324
PARTITION bigReal ZERO 0
PARTITION bigReal LARGEPOS 4E38 1.76769313486231570E308
PARTITION bigReal LARGENEG -4E38 -1.76769313486231570E308
%
% States
STATE Calculator
STATE Calculator memory bigReal
STATE Calculator current bigReal
%
% Operations
OPERATION CalcAdd
OPERATION CalcAdd state Calculator
OPERATION CalcAdd IMPORT inOne bigReal inTwo bigReal
OPERATION CalcAdd changedState current

OPERATION CalcMultiply
OPERATION CalcMultiply state Calculator
OPERATION CalcMultiply IMPORT inOne bigReal inTwo bigReal
OPERATION CalcMultiply changedState current

OPERATION CalcDivide
OPERATION CalcDivide state Calculator
OPERATION CalcDivide IMPORT inOne bigReal inTwo bigReal
OPERATION CalcDivide changedState current

OPERATION CalcSine
OPERATION CalcSine state Calculator
OPERATION CalcSine IMPORT inOne bigReal
OPERATION CalcSine changedState current

OPERATION CalcPower
OPERATION CalcPower state Calculator
OPERATION CalcPower IMPORT inONe bigReal inTwo bigReal
OPERATION CalcPower changedState current
```

Figure 8.7: Test directive for calculator case study

An extract from the resulting base test set is shown in Figure 8.8. The tests are grouped by operation. The involved variables are listed in the `<Variable>` tags, including an identifier, name and type. Each test case has an ID, and a partition allocated for each variable, based on its type.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE TestSuite SYSTEM "testsuite.dtd">
<?xml-stylesheet type="text/xsl" href="testsuite.xsl"?>

<TestSuite name="myTest">
      <Op name="CalcAdd">
            <Variable varNo="1" name="inONe" type="bigReal"\>
            <Variable varNo="2" name="inTwo" type="bigReal"\>
            <TestCase id="1">
                  <value varNo="1">POS</value>
                  <value varNo="2">POS</value>
            </TestCase>
            <TestCase id="2">
                  <value varNo="1">POS</value>
                  <value varNo="2">NEG</value>
            </TestCase>
            <TestCase id="3">
                  <value varNo="1">POS</value>
                  <value varNo="2">ZERO</value>
            </TestCase>

      <--- snip --->

            <TestCase id="25">
                  <value varNo="1">LARGENEG</value>
                  <value varNo="2">LARGENEG</value>
            </TestCase>
      </Op>
      <Op name="CalcMultiply">

      <--- snip --->

            <TestCase id="25">
                  <value varNo="1">LARGENEG</value>
                  <value varNo="2">LARGENEG</value>
            </TestCase>
      </Op>
</TestSuite>
</xml>
```

Figure 8.8: Abstract test specification for calculator case study

Context-based Tests

For context testing, the CX schemas are used to target tests drawn from the base set for additional testing.

CX_performance

$$perfOp : operation \pfun \mathbb{R}$$
$$perfSeq : sequence \pfun \mathbb{N}$$

$$perfOp = \{(CalcAdd, 1), (CalcMultiply, 2)\}$$
$$perfSeq = \{(SeqAddMult, 3)\}$$

CX_reliability

$$relOp : operation \pfun \mathbb{N}$$
$$relPart : partition \pfun \mathbb{N}$$
$$relSeq : sequence \pfun \mathbb{N}$$

$$relOp = \{(CalcAdd, 1), (CalcMultiply, 2)\}$$
$$relSeq = \{(SeqAddMult, 3)\}$$

CX_stress

$$stressOp : operation \pfun \mathbb{N}$$
$$stressSeq : sequence \pfun \mathbb{N}$$

$$stressOp = \{(CalcAdd, 100), (CalcMultiply, 50)\}$$
$$stressSeq = \{(SeqAddMult, 30)\}$$

CX_usage

$$useOp : operation \pfun \mathbb{N}$$
$$usePart : partition \pfun \mathbb{N}$$
$$useSeq : sequence \pfun \mathbb{N}$$

$$useOp = \{(CalcAdd, 1), (CalcMultiply, 2)\}$$
$$useSeq = \{(SeqAddMult, 3)$$

In the calculator example, test sequences could be a series of additional operations, for example adding four numbers as 3 + 4 + 5 + 6. In Z, this could be encoded as:

$$CalcAddFourNumbers = CalcAdd(3, 4) \gg CalcAddCurrent(5) \gg CalcAddCurrent(6)$$

Each of the variables is a BigReal, thus for CalcAddFourNumbers there will be 5^4 base tests, whereas CalcAddPower will have 5^3:

$$CalcAddPower = CalcAdd(3, 4) \gg CalcPowerCurrent(5)$$

8.4 Spiral 5 Adaptation and Execution

This Section describes the approach to Steps 4-6 of the CdCE Process - where the evaluation takes place. There are two main outcomes of the adaptation Step (4): prepare the abstract tests to be executed against specific candidates; assess fit of candidates and quantify the cost/effort involved in adapting the candidate to the target environment. The first outcome feeds into test execution and evaluation while the second provides values for `FFIT`, `FEXS`, `AEFT` and `TFIT`.

The adaption takes the information provided for the candidate and matches it to

the required interfaces/functionality. For a component, adaptation will involve bridging interfaces, while for an application, the key aspects will be installation and matching functionality.

In Step 5 the adapted tests are executed against each of the candidates. This step applies the tests from Step 3 and records results ready to be used to generate the `TRES` and `CX` metrics, as appropriate. For this, the results will need to be entered into an XML file. The output of Step 5 is a series of XML files, each holding one candidate's results against the test suite at a pass/fail level. In Step 6, the results are collated and values for all the evaluation metrics populated. This Step is thus a mechanical compilation of results with some potential for interpretation or scaling of results across candidates.

8.4.1 Background

The adaptation step recognises that there is a likelihood of mismatch between the required and observed interfaces (functionality) of the candidate components (applications). These can be considered in three categories:

- not offered
- offered (may need adaptation)
- excess to requirements.

The offered group includes those interfaces that do not match and need adaptation. Xie and Zhang (2007) provides a framework for adaption of software components with two types of adaptation - component signature and component function. Signature mismatch refers to the names or the types of the parameters, which must be adapted for the calls to be successful.

8.4.2 Approach

Through the other steps of the CdCE Process, the goal has been to avoid aggregation of values. The reason for this has been that the values are often incompatible or do not make sense to aggregate, and underlying data may also be lost or obscured. For Step 4 to 6, aggregation of the fit, adaptation and test results into metrics was considered valid. To an extent it supports the understandability of the results, particularly if large numbers of test cases were involved. The alternative was to have one criterion per test

case, which was considered unmanageable. If the raw information is required for decision making, it is easily retrieved from the XML files output from Step 5.

In preparation for automated test adaptation, and for metrics calculation, those following adaptation model has been developed in Z notation. The model shows that these can be defined in Z, which will be compatible with the test generation specification and implementation.

The adaptation of interfaces is defined through a series of options depending on how they vary from the ideal component's interfaces: it could be as simple as a method name, or more complex - mismatched types, alternate decomposition into methods etc. The first two types of mismatch, method name and type mismatch, require signature adaption in the terms of Xie and Zhang (2007). Decomposition mismatch requires functional adaptation in Xie terms.

The following mismatches will now be considered, with a Z description of the model that could be used:

- Method name
- Type mismatch
- Decomposition
- Missing interface
- Synchronous/Asynchronous.

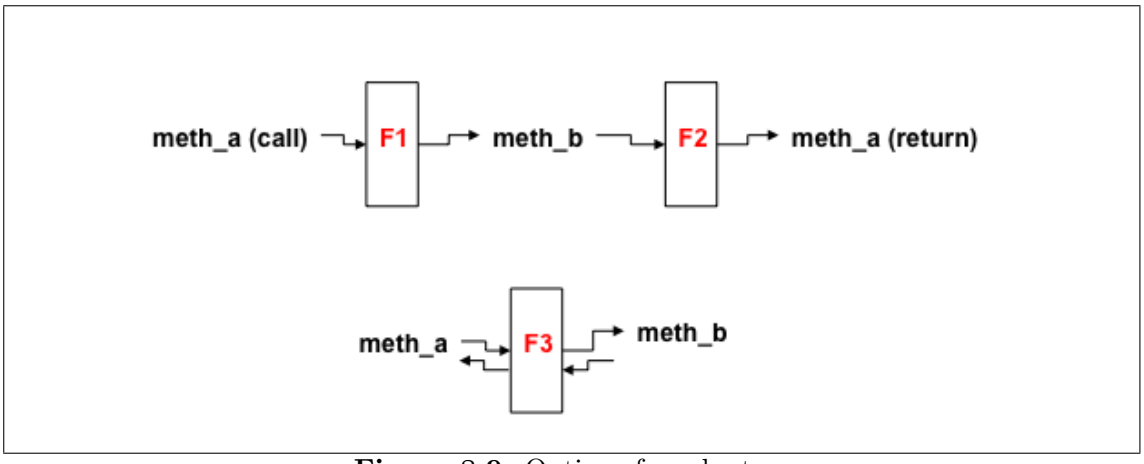


Figure 8.9: Options for adaptors

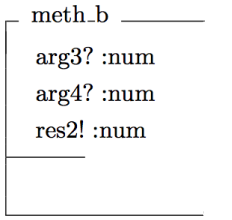
The adaptors can be written/implemented as **filters** on the call (F1) and on the return (F2) of the method call, or as a replacement to the method call (F3) (see Figure 8.9). Both options are looked at for the adaptors that follow.

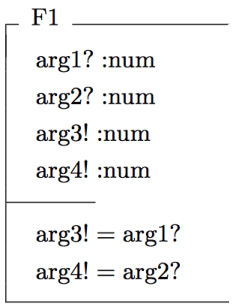
Method Name

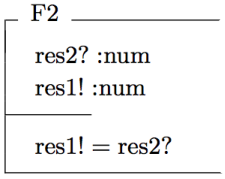
- **Want:** public int meth_a (arg1:int; arg2:int)
- **Have:** public int meth_b (arg3:int; arg4:int)
- meth_b has arguments of same number and type as meth_a, and has the same return type
- The adaptor needs to re-call meth_b with meth_a's arguments.

Filter Option

The input filter takes the incoming arguments for meth_a and equates them to outputs variables which will become the input to meth_b. The result from meth_b is passed through to meth_a. Schema calculus ties the operations together, piping the input/output variables between them.



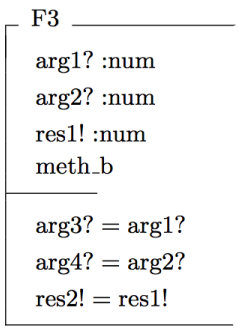




$$\text{meth\_a} == \text{F1} \gg \text{meth\_b} \gg \text{F2}$$

Replacement Option

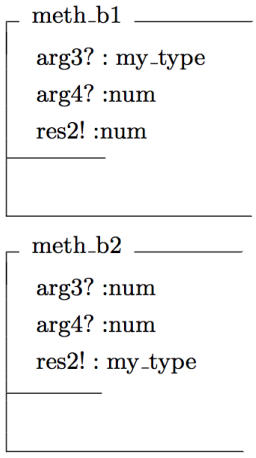
The adaptor absorbs the interface to meth_b and equates the meth_a variables to those of meth_b, then 'calls' meth_b.



meth_a == F3

Type Mismatch

- **Want:** public int meth_a (arg1:int; arg2:int)
- **Have:** public int meth_b1 (arg3:my_type;arg4:int) and public my_type meth_b2 (arg3:int; arg4:int)
- meth_b1 and meth_b2 have arguments of same number but different types to meth_a
- The adaptor needs to change the types of the variables between the meth_a and the meth_b calls.



Filter Option (b1: argument type change)

The input filter takes the incoming arguments for meth_a and equates them to outputs variables which will become the input to meth_b1. A type change operation is carried

out on arg1. The result from meth_b1 is passed through to meth_a. Schema calculus ties the operations together, piping the input/output variables between them.

F5

arg1? :num
arg2? :num
arg3! :num
arg4! :num

---

arg3! = type_change(arg1?)
arg4! = arg2?

F2

res2? :num
res1! :num

---

res1! = res2?

meth_a == F5 ≫ meth_b1 ≫ F2

Filter Option (b2: return type change)

The input filter takes the incoming arguments for meth_a and equates them to outputs variables which will become the input to meth_b2. The result from meth_b2 is passed through to meth_a. A type change operation is carried out on res2. Schema calculus ties the operations together, piping the input/output variables between them.

F1

arg1? :num
arg2? :num
arg3! :num
arg4! :num

---

arg3! = arg1?
arg4! = arg2?

F6

res2? :num
res1! :num

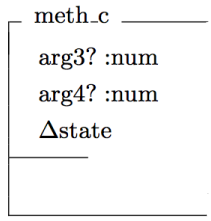
---

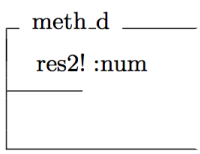
res1! = type_change(res2?)

meth_a == F1 ≫ meth_b2 ≫ F6

Decomposition

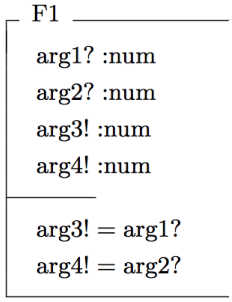
- **Want:** public int meth_a (arg1:int; arg2:int)
- **Have:** public void meth_c (arg3:int; arg4:int) and public int meth_d( )
- meth_c has arguments to set up the state of the component
- meth_d is an accessor method to get the result of the required operation.

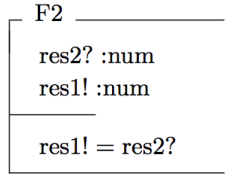




Filter Option

The input filter takes the incoming arguments for meth_a and equates them to outputs variables which will become the input to meth_c. The result from meth_d are passed through to meth_a. Schema calculus ties the operations together, piping the input/output variables between them.



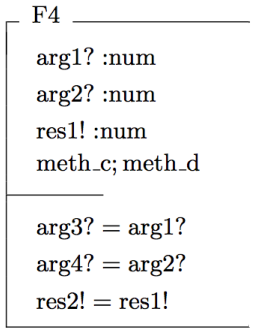


$$\text{meth\_a} == \text{F1} \gg \text{meth\_c}; \text{meth\_d} \gg \text{F2}$$

Replacement Option

The adaptor absorbs the interface to meth_c and meth_d and equates the meth_a

variables to those of meth_c. It 'calls' meth_c, then meth_d, using the results of meth_d.



meth_a == F4

Missing Interface

- **Want:** public int meth_a (arg1:int; arg2:int)
- **Have:** nothing

In this case, the test cases cannot be translated. The null will indicate that test cases, or steps within them, need to be skipped for a particular component.

meth_a == NULL

Synchronous/Asynchronous

Included for completeness in terms of types of adaptation. The mismatch in synchronous and asynchronous methods has not been addressed.

8.4.3 Implementation

Although the understanding and theory for adaptation has been developed, it was decided to use a manual approach to the task. Full implementation of the adaptation theory would not have been possible in the time available. However, the model does help with the understanding of the types of mismatch, and how they should impact on selection.

Using a manual approach, basic information required for determining the metrics for Step 4 are the interfaces required, the interfaces offered and the tests. These need to be scaled in the range of 0..10 for the metrics `FFIT`, `FEXS`, `AEFT` and `TFIT`, with consideration for spreading the field and differentiating the most suitable candidates from the least.

The same approach can be taken with applications, at a less formal level. For example, an email application would have functions for composing, sending, reply, forward, delete, check and viewing mail. The adaptation effort can be used to indicate the installation effort, or another measure appropriate to the selection task. Usability is a candidate - where the adaptation is in the user learning the software. The measure needs to be defined for future reference and then `FFIT`, `FEXS` and `TFIT` can be calculated based on the functionality offered by each candidate (based on operations in Z specification). At the end of Step 4, the adaptation for each candidate is documented.

At this point the definitions of each metric are applied as described in Section 8.2. Step 4 has contributed to `FFIT`, `FEXS`, `AEFT` and `TFIT` with raw values shown in Table 8.5. The results from Step 5 provide the raw values to calculate `TRES`, `CX_U`, `CX_P`, `CX_S` and `CX_R`. An example of the collated metrics at the end of Step 6 is given in Table 8.6.

Candidate	Required	Offered	Excess	FEXS	FFIT	AEFT
BasicCalc	5	4	0	1	3	0
RevPolishCalc	5	9	4	5	5	6
ScientificCalc	5	9	4	5	5	5
BasicRevPolCalc	5	4	0	1	3	3

Table 8.5: Raw adaptation data for shortlisted candidates

Candidate	FFIT	FEXS	AEFT	TFIT	TRES	CX_U
A	6	10	8	3	2	4
B	8	10	10	3	0	0
C	4	6	10	3	3	5
D	10	6	4	10	10	10
E	8	10	8	9	5	8
F	10	6	4	10	10	10
G	8	10	4	9	5	10
H	0	0	0	0	0	0

Table 8.6: Example of scores against metrics for shortlisted candidates

The XML test suite files provide the potential for automation of the test process. However, the testing and adaptation for this project has remained manual as it is not the overall focus of the project. To assist the testing, an XSLT transform has been written to provide a more readable representation of the test cases and results for each candidate. The resulting forms are used as guides for the testing process. The forms are illustrated in Figures 8.10 to 8.12. Once these are calculated, they are inserted into the XML file and into fields in the shortlist file.

The test results are then compiled into the metrics `TRES` and `CX*` in Step 6.

Test Suite: myTest

Software ID	Software Title
.	.
Tester Name	**Test Date**
.	.

Operation: ***LoadFile***

VarNo	Variable	Type
1	NEWfile	INfile

Test Case	Values Used	Result
1	VarNo: 1 fileName	PASS | FAIL | SKIP

	NumTests	NumSkipped	NumPassed
Subtotals	.	.	.

Operation: ***IsWellFormed***

VarNo	Variable	Type
1	NEWxmlFile	XMLfile

Test Case	Values Used	Result
1	VarNo: 1 notWellFormed	PASS | FAIL | SKIP
2	VarNo: 1 wellFormed	PASS | FAIL | SKIP

	NumTests	NumSkipped	NumPassed
Subtotals	.	.	.

Figure 8.10: Example of form for recording results of tests (Part 1/3)

Operation: ***IsValidDTD***

VarNo	Variable	Type
1	NEWxmlFile	DTDvalidation

Test Case	Values Used	Result
1	VarNo: 1 notValidDTD	PASS | FAIL | SKIP
2	VarNo: 1 notFoundDTD	PASS | FAIL | SKIP
3	VarNo: 1 validDTD	PASS | FAIL | SKIP

	NumTests	NumSkipped	NumPassed
Subtotals	.	.	.

Operation: ***IsValidSCHEMA***

VarNo	Variable	Type
1	NEWxmlFile	SCHEMAvalidation

Test Case	Values Used	Result
1	VarNo: 1 notValidSCHEMA	PASS | FAIL | SKIP
2	VarNo: 1 notFoundSCHEMA	PASS | FAIL | SKIP
3	VarNo: 1 validSCHEMA	PASS | FAIL | SKIP

	NumTests	NumSkipped	NumPassed
Subtotals	.	.	.

Figure 8.11: Example of form for recording results of tests (Part 2/3)

Operation: ***TransformFile***

VarNo	Variable	Type
1	NEWxmlFile	XSLTfile

Test Case	Values Used	Result
1	VarNo: 1 noTrans	PASS | FAIL | SKIP
2	VarNo: 1 transError	PASS | FAIL | SKIP
3	VarNo: 1 transOK	PASS | FAIL | SKIP

	NumTests	NumSkipped	NumPassed
Subtotals	.	.	.

Test Results:

Operation	NumTests	NumSkipped	NumPassed
LoadFile	.	.	.
IsWellFormed	.	.	.
IsValidDTD	.	.	.
IsValidSCHEMA	.	.	.
TransformFile	.	.	.
Totals	**.**	**.**	**.**
TFIT	**NumTests - NumSkipped / NumTests**		**___ / ___ = __ / 10**
TRES	**NumPassed / NumTests**		**___ / ___ = __ / 10**

Figure 8.12: Example of form for recording results of tests (Part 3/3)

8.5 Spiral 5 Ranking and Reporting

At this point the results of the testing (metrics) will be compared with those provided as required thresholds in the ideal specification. The ranking is equivalent to the shortlisting done in Step 2 and has similar scope for iteration. The shortlisting approach is described in Chapter 5. The output of this step is the ranked list of candidates and supporting XML files.

Given the ranked list from Step 7, the XML files from each step can be compiled to provide a report on the overall selection process.

8.5.1 Background

As in Step 2, the task in this step would often be approached by aggregating values using WSM or similar approaches. While the issue with incompatible datatypes is not present in this case, the semantics of what the metrics represent still make the WSM or averaging of results inappropriate (for example (`TRES` + `FFIT`)/2 has no real meaning). Following the logic used in Step 2, and the goal of reuse, the classifier is selected as the approach for Step 7. Completing the selection process (Step 8) involves presenting all information and artefacts used. These have been created and recorded throughout.

8.5.2 Approach

The metrics in use correspond with the tests that have been undertaken. The application developer decides on values between 0 and 10 for each of the metrics. As there is potential for the ideal values of the metrics to be too strong or weak, iteration and updating ideal metric values is allowed for. Unlike Step 2, the tuning would be to revise the ideal values/ranges, as opposed to removing criteria. For example, the initial values for the metrics may be {`FFIT`=8, `FEXS`=8, `AEFT`=6, `TFIT`=6, `TRES`=10, `CX_U`=8}. After running the classifier, it may be found that no candidate reaches the required values. The user can consider the metrics and related tests and measures to adjust. In this case, good results on the required functionality is prioritised and the other metrics relaxed. A second run using {`FFIT`=8, `FEXS`=6, `AEFT`=5, `TFIT`=6, `TRES`=8, `CX_U`=8} may provide better ranking results; if not, the values can be adjusted again.

The tools and approach for this step are identical to those of Step 2 with the addition

of the metric attributes into the XML files.

As all of the files are in XML, the reports are easily converted into browsable web pages and links to files using XSLT. Throughout the Process, filenames and directories have included identifying information, including project, date and transformation used, making all of the backend files easily discoverable. This is useful for finding the files for justification and reuse.

8.5.3 Implementation

In Step 7, the results have been collated and are ready to be used to rank/select the components. The required values for the metrics for the selection are included in the ideal specification. These are used to train a new classifier, using the same sequence of scripts and tools as in Step 2. The training data is generated using the processE script, which calls CdCETransformer and Intelligent. processE then runs Weka (J48) to train the C4.5 classifier and create a predictive model. The XML for each of the candidates is then processed using the script grab_predictE to filter the results and indicate the matching candidates. Both processE and grab_predictE are trimmed down versions of the original scripts process and grab_predict from Spiral 4 (see Appendix B).

At this point, the application developer will consider the results. If they are satisfied with the number and quality of selected candidates, they move to Step 8. Otherwise, the developer can reconsider the metrics to be more or less strict, then rerun the classification sequence. Once a decision is made, the report is available as a collation of files and associated XSLT for readability.

8.6 Spiral 5 Evaluation

The product of Spiral 5 is the instantiation of Steps 3 to 8 in a series of representations, scripts and tools. A representation was provided for metrics, behaviour, context, adaptation and evaluation documentation. Scripts and tools have been developed to support test generation and ranking.

The ideal specification used in the CdCE Process evaluates the candidates on both functional and non-functional criteria. Within the functional criteria, it also includes static and dynamic evaluation. The focus of this chapter is the static evaluation which is

included in the CdCE specification as four numeric attributes. These static attributes: `FFIT`, `FEXS`, `AEFT` and `TFIT`, are manually evaluated (at this time). The dynamic metrics are also numeric attributes, but require the execution of the candidate based on the behavioural specification.

The initial evaluation is with respect to the win conditions from Table 8.2. From the preceding discussion, the main automation is in the test generation. This has been able to apply equivalence class partitioning to a Z specification and provide sets of tests for a range of specifications. In each of these cases, thorough manual checking has confirmed that the tests do match the specification and that the resulting tests are valid. The case study in Chapter 9 applies the test generation and evaluation to the selection of an XML editor. The relevance of the metrics are supported by the very suitable candidates found in that case study. As in previous Spirals, the component developers are interested in how their component is assessed and compared, which is made clear for the CdCE Process. In consideration of the win conditions, the Spiral has been successful.

8.6.1 Spiral Goals

The following discussion refers to the goals listed in Tables 8.7 and 8.8 and provides a summary of the achievements of Spiral 5.

Quality

Each step is described in detail including the inputs, outputs, instrumentation and internal procedures (Q5A1). Most of the steps include a level of automation where the results can be easily repeated (Q5A2). For those that require manual effort, documentation is facilitated to help any retracing of steps that may be required. The only section which may give different results is in the test generation - where test data may be randomly allocated within an equivalence class on subsequent runs. The metrics were drawn from SWEBOK, selecting those that are relevant to this point of software development (Q5A3). More background on this is in Section 7.3.1.

Usability

The steps involved in testing and evaluation are clearly described (Q5B1). Support for the tasks involved is provided through automated tools and forms for recording results.

SPIRAL 5	**Purpose** **Issue** **Object** **Context**	**Evaluate** **effectiveness of** **strategies for testing and evaluation** **Spiral 5**	**Result**
Goal 5A	Focus	**Quality**: *Provide structured, repeatable approach to testing and evaluation, drawing on literature*	
	Viewpoint	Quality Assurance personnel	
	Q5A1	Are the updates well documented?	YES
	Q5A2	Is the process repeatable?	YES
	Q5A3	Are the metrics appropriate?	YES
Goal 5B	Focus	**Usability**: *Provide tools to suit user needs and automate testing and evaluation*	
	Viewpoint	Application developer	
	Q5B1	Is the testing and evaluation easy for the user to understand?	YES
	Q5B2	Has the work been tested on real world examples?	YES
	Q5B3	Has tool support and automation been provided?	YES
Goal 5C	Focus	**Intelligence**: *Use AI and knowledge management for testing and evaluation*	
	Viewpoint	Application developer	
	Q5C1	Have any intelligent approaches been used?	YES

Table 8.7: GQM Summary - Spiral 5 (Part 1/2)

A number of case studies have been used to inform the development of Steps 3-8 and the process as a whole is take through a complete case study in Chapter 9 (Q5B2). Automation is provided in the generation of tests and in the ranking of results (Q5B3). Throughout the steps, XML documents and transformations are provided to support documentation.

Intelligence

With respect to Q5C1, Step 7 reuses the classifier approach from Step 2. In this case, the attributes of interest are the metrics, so any tuning of results (iteration) involves changing the thresholds on the metrics. The test generation uses equivalence class partitioning, drawn from the behavioural specification in Z notation. This uses a basic approach of permutation of partitions across each interface.

Innovation

One novel aspect of the testing is that one set of tests is applied across all candidates (Q5D1). This represents the functional requirements for the selection task. The second novel aspect is the use of context schemas to allow for usage, performance and reliability

SPIRAL 5	**Purpose** **Issue** **Object** **Context**	**Evaluate** **effectiveness of** **strategies for testing and evaluation** **Spiral 5**	**Result**
Goal 5D	Focus	**Innovation**: *Consider novel approaches to testing and evaluation*	
	Viewpoint	Academia	
	Q5D1	Have innovations been developed for testing?	YES
	Q5D2	Have innovations been developed for evaluation?	YES
Goal 5E	Focus	**Dynamics**: *Provide flexibility for testing and evaluation*	
	Viewpoint	Application developer	
	Q5E1	Is it possible to modify the process for testing and evaluation?	YES
	Q5E2	Is it possible to update or modify the implementation and tools for testing and evaluation?	YES
	Q5E3	Is there support for iteration in the testing and evaluation?	YES
Goal 5F	Focus	**Reuse**: *Where possible make use of existing code and artefacts*	
	Viewpoint	Application developer	
	Q5F1	Has the work reused external resources?	YES

Table 8.8: GQM Summary - Spiral 5 (Part 2/2)

testing, with reference to the other schemas in the specification. Metrics have been developed based on the literature and the focus on context and testing in the CdCE Process (Q5D2). The classifier is utilised in the ranking of the candidates - avoiding aggregation-based approaches.

Dynamics

The CdCE Process provides independent steps which can be modified to suit organisational requirements (Q5E1). All aspects of the testing and evaluation have scope for modification (Q5E2). Localised changes would need to align to expectations for inputs and outputs between steps. Aspects considered for exchange include: specification language; test case generation; use of a test harness; and, alternative ranking techniques.

Iteration may be required in testing and evaluation to modify the behavioural specification or the requirements for the evaluation metrics (Q5E3). Strictly, the metrics should iterate from Step 7 back to Step 1, but in practice the iteration can occur within Step 7 and update the ideal specification later. If issues are found with the behavioural specification, the iteration would go back to Step 1, then through Step 2 to regenerate

test. In this case, it would be expected that much rework can be avoided by reusing the documentation from the initial run. For example, changing the evaluation metrics would not change the membership of shortlist of candidates, so the adaptation models should be reusable.

Reuse

This work reuses the LaTeX Z implementations and the C4.5 classifier implemented as J48 in the Weka data mining software (Q5F1).

8.7 Spiral 5 Review and Planning

Spiral 5 covered a range of tasks through Steps 3-8 of the CdCE Process. Manual use of the Process had informed the approaches taken. The Steps of the CdCE Process addressed in this Spiral included test generation, adaptation, execution, collation of results, ranking and reporting. The depth of coverage for each of these tasks had to be managed to fit time and project constraints. As drivers to the evaluation, the metrics were defined and matched the project focus on testing and context. Stakeholder win conditions were all satisfied. Responses to Spiral goal questions for the work were also all positive, particularly in the document and dynamics goal areas.

External review of this work came through publication of the complete Process in IET Software (Maxville et al, 2009). Contributions for the Spiral include the single source for abstract tests, allowing for a standardised comparison across all candidates.

When the Spiral began, it was envisaged this was to be the final 'strategy' (RE3) Spiral - the project would then implement a full case study. However, during the review, the researcher was confident that a decision support tool could be developed quickly and would add value. Commitment was given to evaluation and testing as described in this Chapter, and to a short Spiral to develop an emergent visualisation and exploration tool - the ClassifierSuite.

8.8 Summary

This chapter covered Spiral 5, the implementation of the evaluation steps of the CdCE Process. Key outcomes have been the development of a series of metrics for functional

fit, adaptation and testing (with context). The evaluation is driven by the results of the adaptation and testing. Tests are generated from a Z specification, providing a common set of tests for all candidates. These are adapted as needed, providing information on fit, excess functionality and adaptation effort. The results are collated into metrics which inform the ranking, which utilises a classifier and iteration to tune the evaluation if required. Although some parts of the Process remain manual, there is a structured, supported process which can aid repeatability and transparency of component selection. With the evaluation approach in place, Chapter 9 describes the development of a tool to support criteria selection in the CdCE Process.

Chapter 9

The ClassifierSuite

In this chapter a revised approach to shortlisting is presented, as explored in Spiral 6. In Chapter 6 the shortlisting approach was introduced using classifiers and iteration to refine the ideal specification to determine a satisfactory shortlist. That approach required a decision to be made on each iteration applying knowledge and/or intuition. In the new ClassifierSuite approach, the user-defined mandatory and non-mandatory criteria are used with the CdCE software to generate a **suite of classifiers** for all possible combinations of the criteria. These are used to create a corresponding set of shortlists. Using the graphical user interface of the ClassifierSuite tool, it is possible to interactively explore the impact of each of the criteria on the shortlists and make more informed choices. The user can then consider more combinations of criteria as the tool makes it easier to understand a larger range of options.

Table 9.1 lists the goals for the Spiral 6. There is potential for two levels of **quality** improvement, one in the understanding of the options available, and the other in the ability to consider a wider range of options - the aim being to have the 'best' shortlist in the context of the requirements. Spiral 6 is driven by the user perspective, and a

SPIRAL 6	GOALS
Quality	Provide tool support to allow user to have a better understanding of their choices and make more informed decisions
Usability	Assist user in making decisions by providing visualisation of information and tool support
Intelligence	Transform the outputs of the classifiers into a form more easily understood
Innovation	Provide an information rich interface to assist decision making
Dynamics	Allow user to explore the impact of changing parameters at a high level, but also drill-down to detail
Reuse	Where possible make use of existing code and artefacts

Table 9.1: Goals for Spiral 6

diagrammatic view of the sets of criteria helps in reasoning about choices. This improved **usability** can be further enhanced by the provision of supporting analysis tools for the user. The tool can help pull together the value provided by the application of machine learning by simplifying the understanding of the outputs of the classifiers, focussing more on the criteria and the shortlists - which is the user perspective (**intelligence**). The tool, and its interface, build on novel contributions, giving potential for **innovation** in the provision of support for the selection process. As an iterative and **dynamic** process, the tool support for the selection process needs to be flexible and responsive to the user needs. While the tool is a new piece of work, the code and approach should **reuse** existing resources where possible.

The chapter begins with a discussion of the objectives for the ClassifierSuite and its place in the CdCE Process, then looks at the conceptual side of the suite. Examples of the ClassifierSuite tool being used in four different selection scenarios follow, before a review of the work carried out in this spiral.

9.1 Spiral 6 Overview

Spirals 3-5 had developed procedures and tools to automate the CdCE Process as part of **RE3**. There was an emerging need to help the user manage and understand the increasing amount of data (candidates and sets of criteria) able to be considered as a result of automation. In this Spiral the aim is to plan and implement a visualisation and exploration tool for the shortlisting data, along with procedures and guidelines for its use. This makes an additional contribution on **RE3** to improve the portrayal and exploration of the selection information made available in the shortlisting phase of the CdCE Process (Step 2). Considering the actors in the selection process, the relevant use cases (Figure 9.1) are **Select Component**, **Revisit Selection** and **Assess Selection**. All can be made easier for the application developer, both in carrying out the task and complying with quality requirements.

The stakeholder evaluation criteria are provided in the table of Win conditions (Table 9.2). In this Spiral, the application developers are interested in a tool that helps them to select the criteria set to improve the shortlist. For quality assurance the tools and related procedures must include documentation and be repeatable. Academia require that the

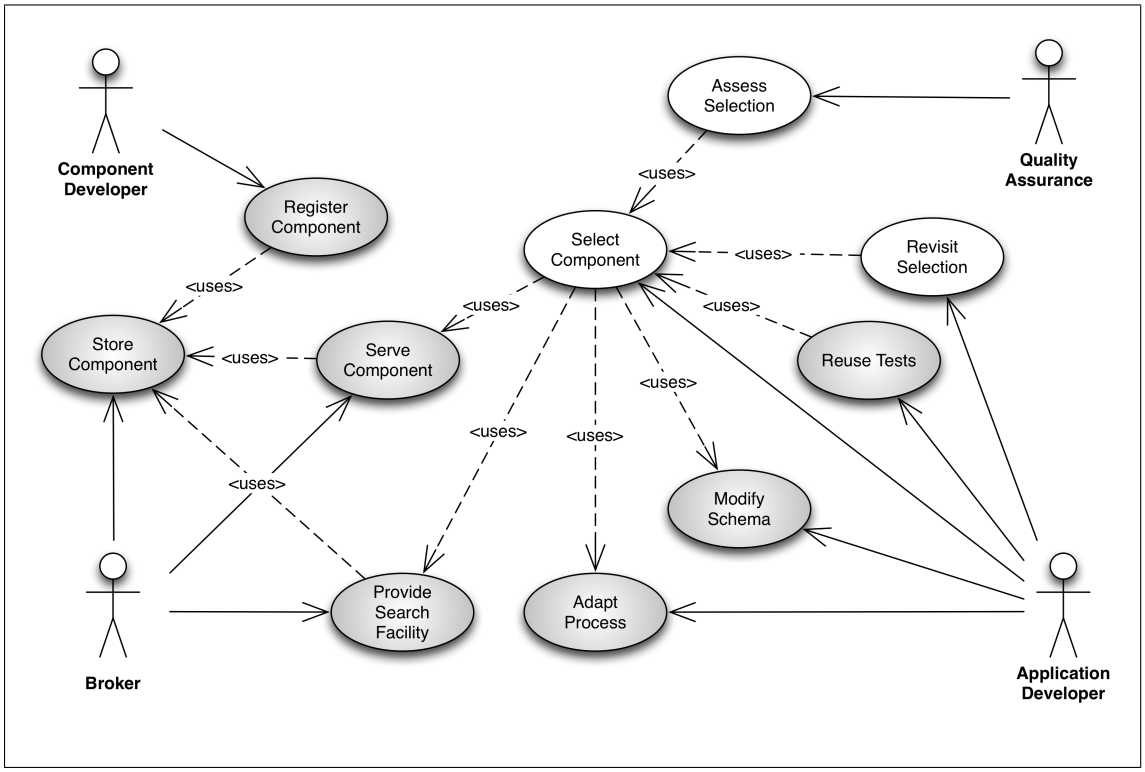


Figure 9.1: Use cases for the ClassifierSuite, the focus of Spiral 6 (those not in the scope for this Spiral are greyed)

approach consider and use, or add to, existing literature.

9.2 Spiral 6 Context

Spirals 3 and 4 implemented strategies to support the shortlisting of candidates. The tools developed in the previous Spirals automated the generation of shortlists based on the ideal specification (user requirements) and the C4.5 classifier. With use of the tools, the bottleneck became the trading off between criteria to get the 'best set' of candidates with respect to the ideal specification. The tools and scripts from previous Spirals make

Stakeholder	**Win Conditions**
Application Developers	Tool makes decisions easier Allows more selection scenarios
Academia	Representation techniques are valid Approach has novelty and draws on literature
Quality Assurance	Allows documentation and justification of decisions Recall and relevance are enhanced

Table 9.2: Win conditions for stakeholders (Spiral 6)

it trivial to run extra sets of criteria, so it is possible to generate all possible sets. A new problem needing solution was how to take in the amount of data that could be generated. The preferred approach was to provide a visualisation and exploration tool to help the application developer compare the various choices for selection criteria.

The ideas for this work grew organically from diagrams the researcher used to help understand and validate the data from previous shortlisting tasks. When considering formalising the approach, similar 'lattices' were found in the literature in other domains (Ganter and Wille, 1997, Chen and Yao, 2008). The combination of proven usefulness for manual work, and external validation in the literature indicated a low risk in adopting this approach and including it in the CdCE Process tools.

9.3 Spiral 6 Approach

The ideal specification is used to generate a classifier for shortlisting in Step 2 of the CdCE Process. The converted repository data is classified and if the resultant shortlist is unsatisfactory, the user iterates to Step 1 to refine (loosen, tighten, alter) the ideal specification. The first pass of the shortlisting uses the full specification to allow tuning of the selection criteria, particularly those that may lock out all candidates. The Weka system can be used to view the statistics across the input data set to pick out issues and create a meaningful base ideal specification[1], which is referred to as S10 (Set 0, level 1). From this ideal specification the mandatory and non-mandatory criteria are identified by the user, with the non-mandatory criteria providing flexibility. As the refining process has been automated, compiling additional shortlists is quick and easy. A graph can be built representing all of the possible combinations of criteria by dropping one non-mandatory criterion at a time (Figure 9.2) and providing a count of the candidates that result from each set. The naming convention for the sets is '`S+level+set#`', with '`set#`' starting at 1. The criteria used in each set are indicated by the letters in the corresponding box. For example Set S22 in the Figure includes criteria {A,B,D,E}.

The graph can be traversed to help find an optimal criteria set to identify a shortlist of components. This approach condenses the iteration process, and removes some of the heuristics and subjectivity: providing a better overall view of the impact of including or

[1] If the base specification is too restrictive, many criteria will need to be dropped before any matching candidates are returned. Criteria with high levels of missing data can be identified through Weka.

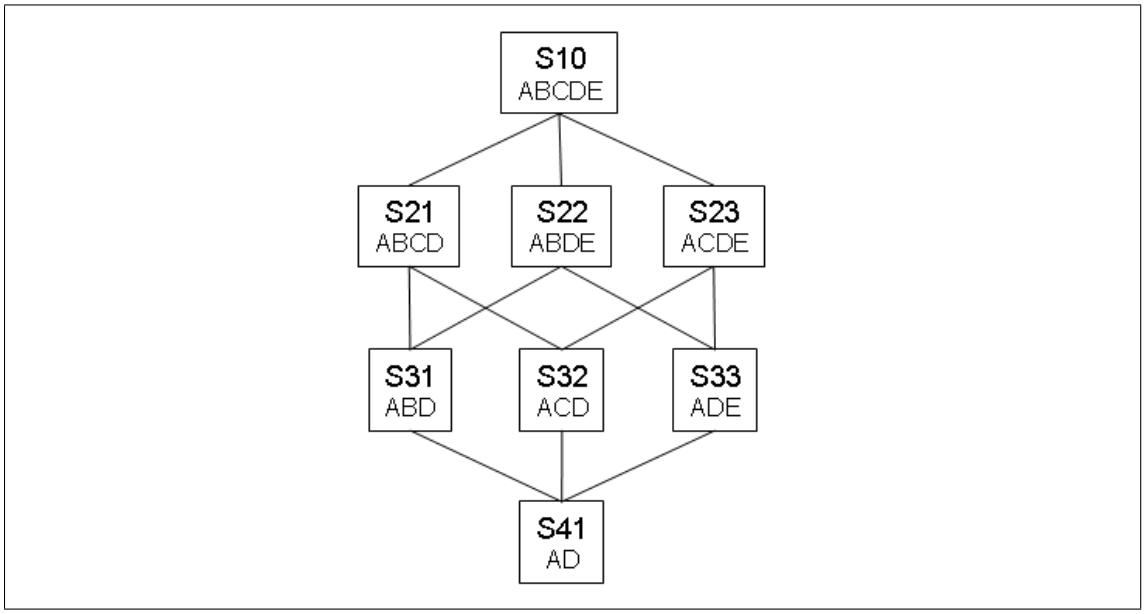


Figure 9.2: Graph representing a series of criteria sets, classifiers and subsequent shortlists. $\{A, D\}$ are mandatory and $\{B, C, E\}$ are non-mandatory.

excluding specific criteria.

9.4 Spiral 6 Implementation

At this point the project has a strong set of software and scripts which will be utilised in this Spiral. The applications written to date have been command line and script driven. This work was aimed at presenting an improved interface, which brings in a different element of programming. ClassifierSuite is a Java program with an interactive interface developed using AWT. Additional scripts were developed to take the existing outputs of the shortlisting and create an XML file as input for ClassifierSuite. New scripts were created to post-process the existing output files to provide summary data. In addition, some change in naming files and folders was necessary to make it possible to drill-down into the shortlists.

The software was designed and implemented to include a graphical display, saving and printing of images and text and GUI interaction. The first pass of implementation drew the graph of the sets and the connections between them. This was validated against graphs containing 8, 16 and 32 sets (3 non-mandatory criteria = $2^3 = 8$ sets), although larger graphs can be accommodated. With the criteria sets drawn, the next iteration within this Spiral was to allow the user to select, trace and compare criteria choices.

More discussion of the implementation and a discussion of some key points follows.

9.4.1 The ClassifierSuite

When considering the shortlisting step, the CdCE Process has allowed for iteration between the specification of the ideal component and the shortlisting of candidates to tune the specification by loosening or tightening the criteria. In practice, the first iteration has used the most strict set of criteria with loosening occurring iteratively to find a suitable shortlist. This led to an effective process but was subjective in the choice of the next criterion to loosen. As the tools for generating the classifiers and extracting the shortlists had been automated in Spirals 3 and 4, a more exhaustive approach was considered. This would allow all possible sets of criteria to be determined and the shortlists made available to the user. By creating a series of ideal specifications, a series - or suite - of classifiers can be generated using scripts. This can, in turn, be used to classify the items in the repository dataset and filter them into the respective shortlists.

During the specification step, the user can flag certain criteria as mandatory (e.g. the description or the development language). Holding these criteria fixed, they can progressively loosen the criteria by dropping one criterion at a time out of the selection set. For example, with the criteria $\{A, B, C, D, E\}$ a mandatory set may be $\{A, D\}$ and the non-mandatory set $\{B, C, E\}$. The resulting possible sets of criteria are:

$\{ABCDE\}$,
$\{ABCD\}, \{ABDE\}, \{ACDE\}$,
$\{ABD\}, \{ACD\}, \{ADE\}$,
$\{AD\}$

This can be represented as a graph, as shown in Figure 9.2. Each node represents a selection set and each edge the removal of one of the criteria. After the respective classifiers have processed the repository data, the user can then view the graph marked up with the number of items in each shortlist and the respective criteria to help to select one or more sets for their shortlist. The metadata from the software repository is available for each candidate on each shortlist which can be viewed to help the user's decision.

In terms of the processing of the repository data, in the past it has taken many iterations to create a satisfactory shortlist, with many sets of criteria untried. In the new

approach, the user can get an overall picture of the variations, drill down as required and still have that option of returning to the specification to make changes before a second iteration. The scenarios in Section 9.5 show the application of the new approach to variety of selection tasks using a real world repository.

9.4.2 Scalability

The height of the graph is dependent on the number of non-mandatory criteria. Each level of the tree progressively removes one criterion from the set, starting with the full set at the top of the graph and finishing with the mandatory set at the bottom. The number of sets and classifiers in each level of the graph is:

$$
\begin{aligned}
&\mathrm{M} \;=\; \text{set of mandatory elements}\\
&\mathrm{N} \;=\; \text{set of non} - \text{mandatory elements}\\
&\text{Number of mandatory elements} \;=\; \mathrm{m}\\
&\text{Number of non} - \text{mandatory elements} \;=\; \mathrm{n}
\end{aligned}
$$

$$
\text{Graph height} \;=\; \mathrm{n}+1
$$

$$
\begin{aligned}
&\tbinom{\mathrm{n}}{0} \times \{\mathrm{m}_1,\ldots,\mathrm{m}_\mathrm{m}\} \cup \{\mathrm{n}_1,\ldots,\mathrm{n}_\mathrm{n}\}\\
&\tbinom{\mathrm{n}}{1} \times \{\mathrm{m}_1,\ldots,\mathrm{m}_\mathrm{m}\} \cup \{\mathrm{N} \setminus \{\mathrm{n}_\mathrm{j},\mathrm{n}_\mathrm{k}\} \bullet \mathrm{n}_\mathrm{j},\mathrm{n}_\mathrm{k} \in \mathrm{N}\colon\\
&\ldots\ldots\mathrm{j} = 1\ldots\mathrm{n}, \mathrm{k} = 1\ldots\mathrm{n}, \mathrm{j} \neq \mathrm{k}\}\\
&\ldots\\
&\tbinom{\mathrm{n}}{\mathrm{n}-1} \times \{\mathrm{m}_1,\ldots,\mathrm{m}_\mathrm{m}\} \cup \{\mathrm{n}_\mathrm{k} \in \mathrm{N}\colon \mathrm{k} = 1\ldots\mathrm{n}\}\\
&\tbinom{\mathrm{n}}{\mathrm{n}} \times \{\mathrm{m}_1,\ldots,\mathrm{m}_\mathrm{m}\}
\end{aligned}
$$

$$
\text{Number of classifiers/sets required} = 2^\mathrm{n}
$$

Note that the number of mandatory elements (criteria), m, does not affect the size of the graph. Considering the criteria in Figure 9.2, the number of mandatory elements (m) is 2: {A,D}, with 3 (n) non-mandatory {B,C,E}. The height of the graph is n+1 = 4. The number of items at each level increases by a power of two, then reduces symmetrically. The total number of sets required in the example is $2^3 = 8$ sets. Inclusion of a large number of non-mandatory criteria could lead to combinatorial explosion - doubling with

each additional criterion.

The time taken to generate classifiers and shortlists is affected by the attribute types and the selected transformation (described in Section 6.3.5). This is related to the number of values possible for each attribute, which ranges from 2-3 to over 100. The CdCE tools are currently written in Java and are not optimised, and the complete processing took an average of two minutes per set on a 2 GHz Pentium with 767 Mb RAM. The generation of the classifiers is the most time-consuming part and can run in the background while the user carries on with other tasks. The classification of the repository data takes less than a minute. In practise, limitations on increasing the size of the graph may be more in relation to the user's ability to interpret it than in the processing time required.

9.4.3 Interpreting the Data

There are many possible ways to represent the data effectively for the user to be able to have a more intuitive view of the possible shortlists. The approach for the ClassifierSuite is to list the criteria included in the set, represented by letters (e.g. A-E), along with the number of items in the shortlist for that set, then view these sets in a 2-D graph. This works well for a small graph such as the one in Figure 9.2. Where the number of non-mandatory items goes above four or five, the graph may become difficult to interpret. One way to reduce graph size is to split it on a well-understood criterion (e.g. date updated - criterion F in Figure 9.3). There are now two graphs, one with the criterion and one without. As the order of all other criteria are the same across each graph, the user can compare within and across the graphs and criteria. The user can then consider which criteria are important with, and in the absence of, the split criterion.

Another approach is to highlight nodes with criteria of interest. In the case study, criterion A was of particular interest as it caused dramatic changes in shortlist sizes. Figure 9.4 highlights the sets/nodes that include criterion A, and the edge which links to the equivalent node without A.

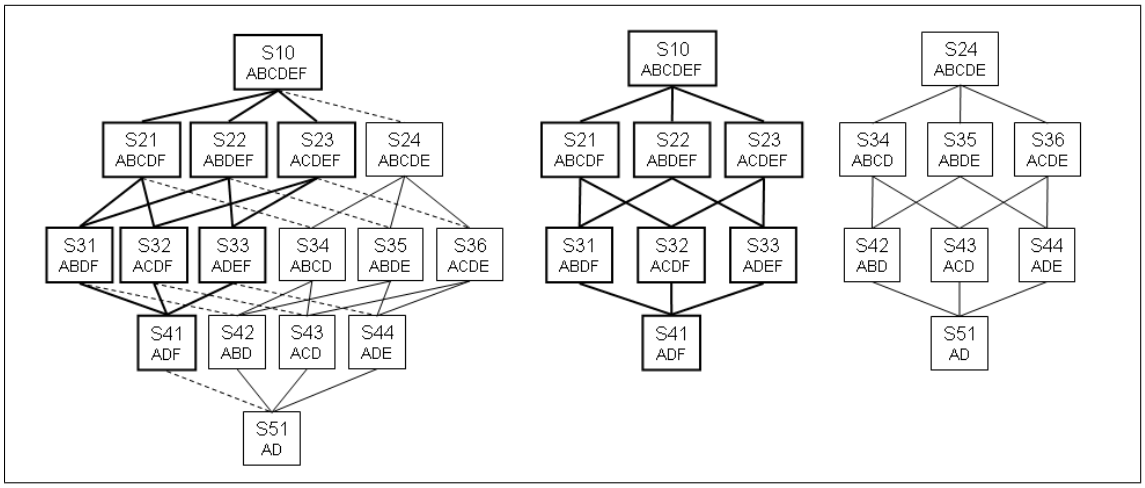


Figure 9.3: Splitting the graph on criterion F. The complete graph is on the left, with the dashed lines showing the 'split'. By restricting to those sets that include criterion F, the middle graph can be extracted. The remaining sets (without F) are shown in the graph on the right.

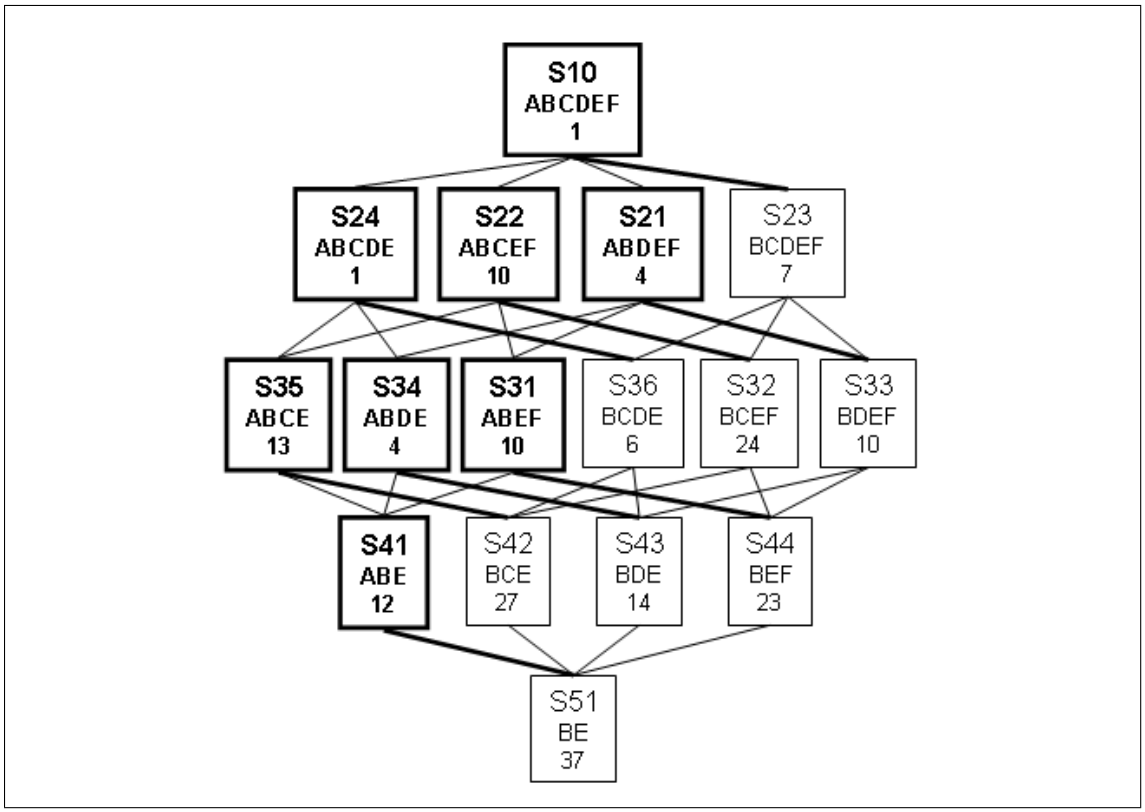


Figure 9.4: Graph representation of criteria sets, highlighting criterion A

The difference between the numbers returned in the shortlists with and without A in Figure 9.4 are listed below:

$\{A, B, C, D, E, F\} = 4 \ldots \{B, C, D, E, F\} = 7 \quad (\text{diff} = 7 - 4 = 3)$

$\{A, B, C, D, E\} = 1 \ldots \{B, C, D, E\} = 6 \quad (\text{diff} = 6 - 1 = 5)$

$\{A, B, C, E, F\} = 10 \ldots \{B, C, E, F\} = 24 \quad (\text{diff} = 14)$

$\{A, B, D, E, F\} = 4 \ldots \{B, D, E, F\} = 10 \quad (\text{diff} = 6)$

$\{A, B, C, E\} = 13 \ldots \{B, C, E\} = 27 \quad (\text{diff} = 14)$

$\{A, B, D, E\} = 4 \ldots \{B, D, E\} = 14 \quad (\text{diff} = 10)$

$\{A, B, E, F\} = 10 \ldots \{B, E, F\} = 23 \quad (\text{diff} = 13)$

$\{A, B, E\} = 12 \ldots \{B, E\} = 37 \quad (\text{diff} = 25)$

These patterns of differences are significant in the choice of criteria. To determine if a criterion is blocking (bad) or filtering (helpful), we need to drill down into the shortlist. An example from the email client case study (Section 9.5.2) showed that, while the description (B) was significant, more investigation was needed to identify if the impact was positive or negative. As the criteria also included detail (detailed description - C) it could be that: the attributes were redundant; one of them was rarely matched; both were required; or one was needed and not the other. Drilling into the shortlists revealed that those with 'description' in the criteria were almost entirely relevant, while those that did not include 'description' were almost completely irrelevant. This knowledge gave the additional benefit of adding another criterion to the mandatory set, halving the graph size. Awareness of such patterns are always of value, with different impacts depending on the data.

9.4.4 Properties of the Graph

Intuitively, and through working with the ClassifierSuite and its graphical representation, there are properties which can help the user understand the selection data. It should be noted that the classifiers are not always 100% accurate[2], so there are times that the properties will not hold, or that there may be an underlying issue which becomes apparent when viewing the suite as a whole:

Property 1 - Increasing shortlist size

[2]e.g. 96% accuracy, on a large dataset can result in mis-classifications. This is more common on Transformation 5 (T5.

The number of items in a shortlist for a set at level L will always be greater than those of the connected sets at level L-1.

Property 2 - Descendants of shortlists are subsets

The items in the shortlist for a set at level L will include the items from the connected sets at level L-1.

Property 3 - Impact of criteria

Given a particular criterion, A, an indication of its impact on the selection task can be calculated by comparing the number of items in each set that includes A, with its complementary set which excludes A.

These properties can aid the user in having an intuitive feel on decision making. For example, moving up through the levels results in smaller (or equal) shortlists, moving down will give larger (or equal) shortlists (Property 1). Following connected branches will give sets that have a high level of overlap (Property 2). An approach to getting coverage of possibilities is to choose shortlists on less connected branches of the graph. Finally, the impact information (Property 3) can assist in understanding of the impact of each criterion in that scenario and repository.

9.5 Spiral 6 Results: The ClassifierSuite in Action

This section considers the following scenarios that have been used as case studies during the development of the CdCE Process:

1. Scientific calculator
2. Email client
3. XML editor
4. XML editor with date.

These examples revisit Step 2 of the process using the ClassifierSuite instead of relying on iteration, which was used in scenarios 1 and 2 previously. In each case there has been an initial iteration which removed problematic criteria[3] and mandatory/non-mandatory criteria have been identified[4].

[3]Problematic criteria are those that have a large proportion of missing data or are never matched.

[4]The case studies use T5 with the level of abstraction set to 2 - where the ontology tree is pruned to 2 levels deep.

9.5.1 Scientific Calculator

This case study explores the selection of a component to provide scientific calculation functionality to a target system. The system provides the interface for the user to enter the information to set up the calculation, with the calculator component carrying out back-end calculations. Context information recorded in the ideal specification includes the platform, programming language (desirable), memory usage (disk and RAM), required functionality and context information.

Criteria	Description	Value	Mandatory
A	Description	Calculator	Y
B	Detail	Scientific Calculator	N
C	Development Language	Java	Y
D	Operating System	Linux	N
E	Development Status	5 - Production	N

Table 9.3: Selection criteria for Scientific Calculator

This scenario was used as proof of concept for the manual application of the CdCE Process as described in Chapter 5. As a small criteria set (Table 9.3), it can help to introduce the ClassifierSuite. The ClassifierSuite takes an XML input file composed from the output of running scripts to generate the suite of classifiers. As shown in Figure 9.5, the XML file includes information describing the selection criteria, along with details for each set of criteria.

When ClassifierSuite is run, the XML input file is given as a command line argument. The interface for the ClassifierSuite is shown in Figure 9.6. Users can see the criteria, output 2-D and text versions of the graph, highlight and compare criteria, drill-down into the respective shortlists, draw on the canvas, print and save the image. In Figure 9.6 criterion B is highlighted (boxes shown in red). Another way of investigating the non-mandatory criteria is to select them for comparison. The output of comparing the three non-mandatory criteria {B,D,E} respectively is shown in Figure 9.7. This has been approached by considering the relative impact of the criteria.

The differences between the numbers returned by each of the criteria sets are listed below. Differences are calculated by taking connected sets and comparing the number of items in the set with and without a specific criterion, e.g. B and B' (without B). Following through the graph on each criterion, the differences in order of their level in

```
<xml>
<classifier_suite>
<!-- Criteria -->
<criteria length="5" nonMandatory="3">
<crit name="A" desc="description" value="calculator"/>
<crit name="B" desc="detail" value="scientific calculator"/>
<crit name="C" desc="devLanguage" value="Java"/>
<crit name="D" desc="operatingSystem" value="Linux"/>
<crit name="E" desc="devStatus" value="5 - Production/Stable"/>
</criteria>
<rootNode name="s10"></rootNode>
<set name="s10">
<directory>C:\_valerie\dev\FMfilter\calc_2008\calc_2Jan08t5s10_2008-01-03_01_22</directory>
<filename>shortlist.xml</filename>
<setCrit>ABCDE</setCrit>
<count>1</count>
<connects>s21</connects>
<connects>s22</connects>
<connects>s23</connects>
<level>1</level>
</set>
         ....
<set name="s41">
<directory>C:\_valerie\dev\FMfilter\calc_2008\calc_2Jan08t5s41_2008-01-03_01_45</directory>
<filename>shortlist.xml</filename>
<setCrit>AC</setCrit>
<count>14</count>
<level>4</level>
</set>
</classifier_suite>
</xml>
```

Figure 9.5: XML input file for ClassifierSuite (scientific calculator)

the graph (L1-L3) are:

$$\text{B} : 5, 10, 6, 12 \text{ ... Derived from : } \text{L1} = 5; \text{L2} = 10, 6; \text{L3} = 12$$

$$\text{D} : 0, 5, 1, 7 \text{...L1} = 0; \text{L2} = 5, 1; \text{L3} = 7$$

$$\text{E} : 0, 1, 13 \text{...L1} = 0; \text{L2} = 1, 1; \text{L3} = 3$$

It is clear from these differences that the highest impact is criterion B, then D then E. A high-impact criterion can either be eliminating useful items or helping to create a more relevant shortlist. Using the drill-down facility, it is possible to look at the shortlists for each of the sets to determine if the criteria are having a positive or a negative effect on the relevance of the shortlist. Although it would normally be impractical to drill down on all of the shortlists (only a sample would be looked at) in this case all shortlists can be investigated for relevance.

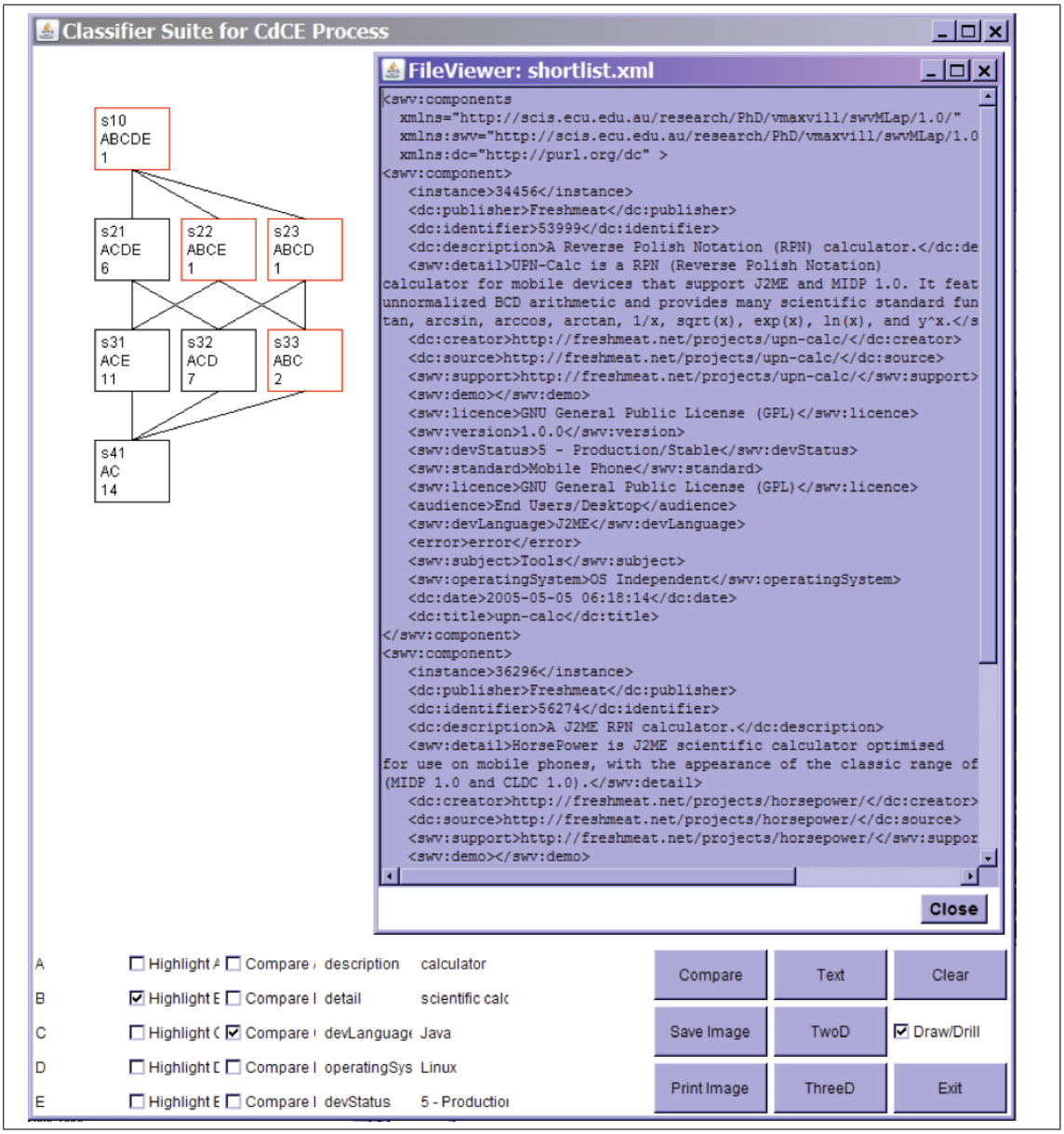


Figure 9.6: ClassifierSuite output for the scientific calculator scenario

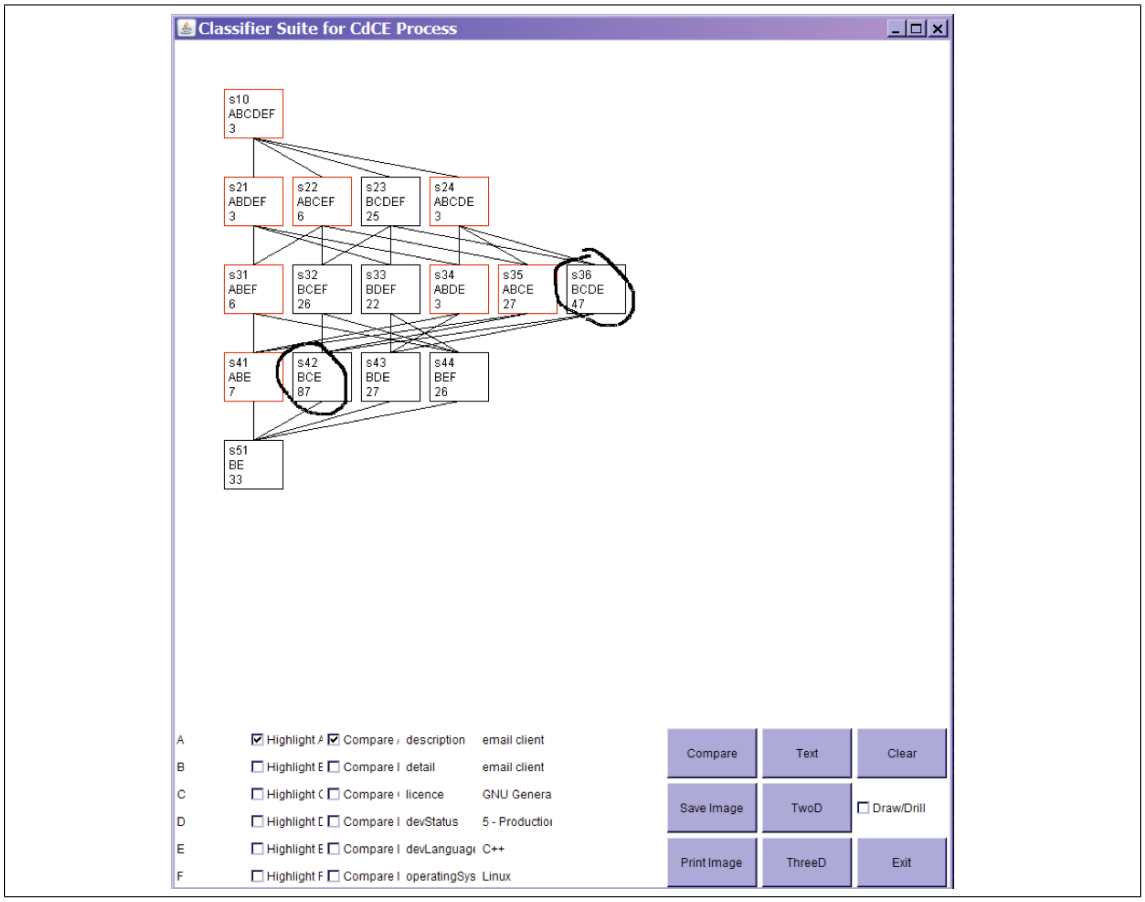


Figure 9.7: Screenshot of comparison of impact of non-mandatory criteria.

For each set, the full shortlist was manually evaluated as to whether each item was suitable enough to work with:

s10 : ABCDE – 1 relevant item / 1 item in the set

s21 : ACDE – 3/6

s22 : ABCE – 1/1 (same shortlist as s10 and s23)

s23 : ABCD – 1/1 (same shortlist as s10 and s22)

s31 : ACE – 3/11

s32 : ACD – 4/7

s33 : ABC – 2/2

s41 : AC – 5/14

Drilling down on set s33[5] gives two very relevant items on the shortlist, whereas s41 gives five good options from fourteen (36% relevance). This implies that the shortlist

[5]s33 is equivalent to S33 - there is no significance to any change in case on the set names.

with the highest relevance is s33[6]. Recall is also important as some relevant items may not be in the returned set[7]. Considering all of the sets, five items have been identified, although there may be more in the repository. In this discussion, the total number of relevant items is assumed to be five. For recall, set s41 has five items that are relevant, out of a shortlist of 14, giving 100% recall and 35.7% relevance. User knowledge and the scenario at hand can be used to help guide the choice of criteria, supported by the ClassifierSuite.

For this scenario, s33 is considered to be the preferred criteria set/shortlist. If needing to be more selective, shortlist s22 or s23 could be used, following the edge to the sets with one criterion difference. Using the suite of classifiers and the ClassifierSuite tool resulted in a different set of criteria being selected s33 = {Description = 'calculator', Detail = 'scientific calculator' and devLanguage = 'Java'} compared to the manual approach, which ended with only the mandatory criteria s41 = {Description = 'calculator' and devLanguage = 'Java'}. Using the ClassifierSuite has resulted in a tighter set of selection criteria and a greater understanding of the impact of each criterion.

9.5.2 Email Client

Much of the development and refinement of the CdCE Process and tools were carried out using the email client scenario. This scenario is the search for an email client written in C++ under a GPL licence. Development status should be at production level with Linux as the target platform. The mandatory criteria are the detailed description and the operating system. The selection criteria are summarised in Table 9.4

Criteria	Description	Value	Mandatory
A	Description	email client	N
B	Detail	email client	Y
C	Licence	GNU General Purpose License (GPL)	N
D	Development Status	5 - Production	N
E	Development Language	C++	Y
F	Operating System	Linux	N

Table 9.4: Selection criteria for Email Client

[6]relevance = number of relevant items / number items returned.
[7]recall = number of relevant items returned/ total number relevant items in dataset.

Assigning letters to the criteria, {B,E} are mandatory, with {A,C,D,F} non-mandatory. This results in a suite of 16 criteria sets and corresponding classifiers.

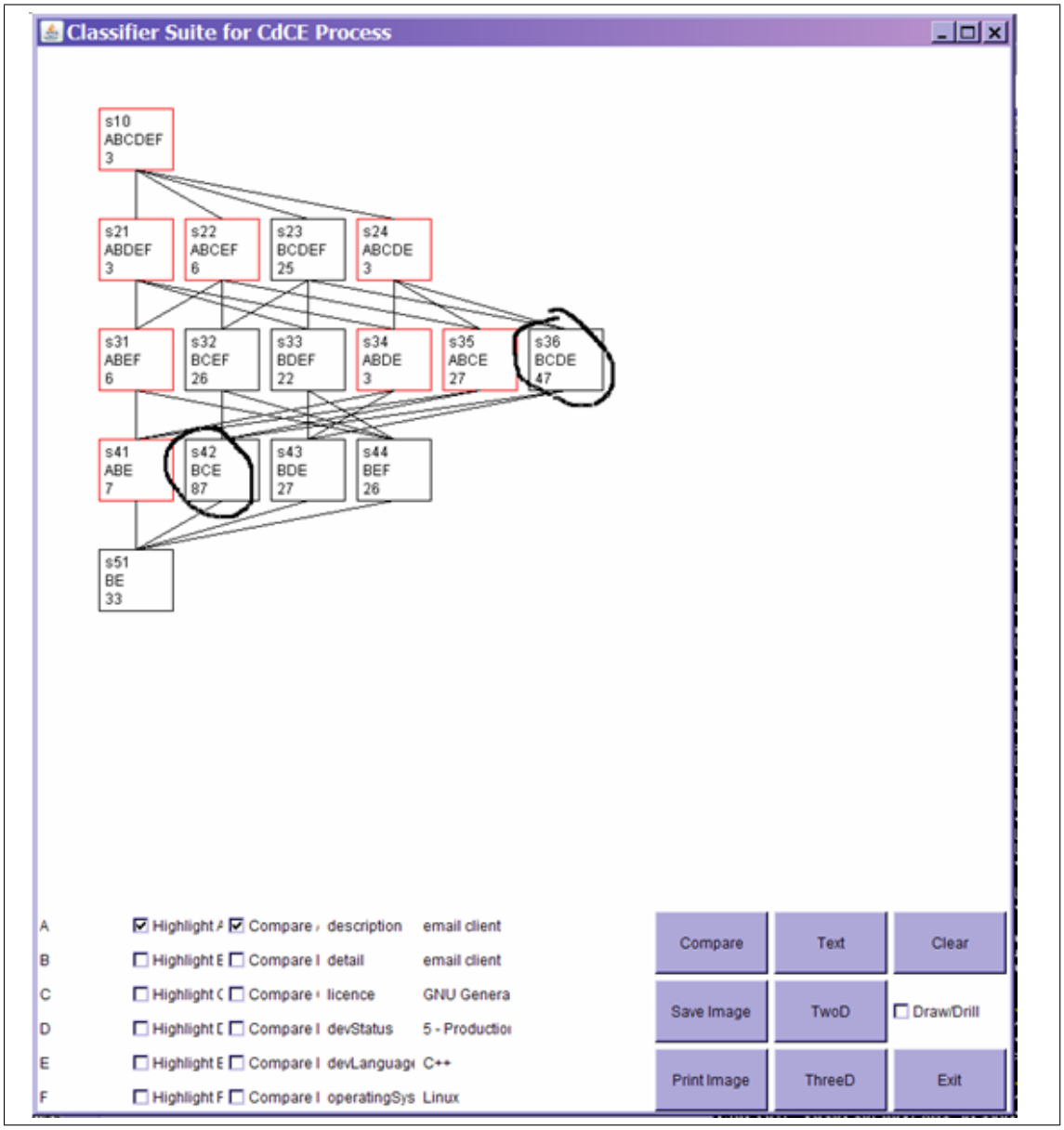


Figure 9.8: ClassifierSuite output for the email client scenario

Using the ClassifierSuite tool, the graph in Figure 9.8 is generated. Running a comparison (Figure 9.9), {A} shows up as high-impact, making a difference to shortlist sizes of approximately 4-fold, averaging around 25 items. Drilling down using the tool on sets s21 (3/3) and s33 (4/22), it appears that criterion A is necessary for relevant results - possibly because the terms 'email client' are often used in the description of software that is not an email client itself.

```
---Start of comparison---

Comparison of values : {A}

Count for  ABCDEF is : 3 and BCDEF is :25
Count for  ABDEF is : 3 and BDEF is :22
Count for  ABCEF is : 6 and BCEF is :26
Count for  ABCDE is : 3 and BCDE is :47
Count for  ABEF is : 6 and BEF is :26
Count for  ABDE is : 3 and BDE is :27
Count for  ABCE is : 27 and BCE is :87
Count for  ABE is : 7 and BE is :33

---End of comparison---
```

Figure 9.9: Screenshot of impact of non-mandatory criteria in email client scenario

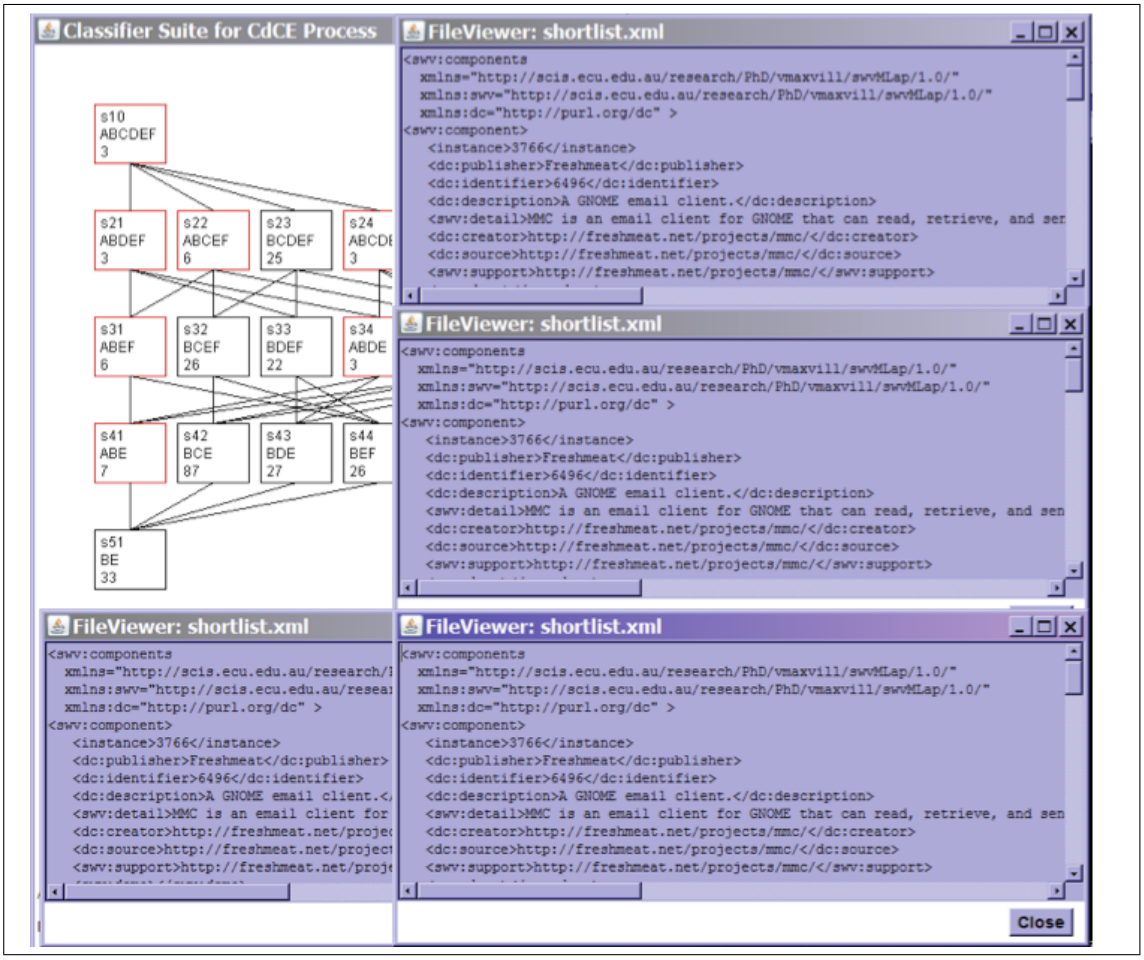


Figure 9.10: ClassifierSuite output for the email client with drilldown on s10, s21, s24 and s34

The next interesting aspect of this dataset is that sets s10, s21, s24 and s34 give identical counts. Drilling down on the four shortlists as shown in Figure 9.10 confirms that the sets are the same. There are also some results in the data that go against the properties listed in Section 9.4.4. Sets s36 and s42 (circled in Figure 9.8) have higher shortlist sizes than their descendants. Investigation into log files shows that the classifiers for each of these sets has a lower percentage correctly classified than the others, although it is rated at 98%. The issue occurs when shortlists include multiple ontology criteria with large numbers of possible values.

Criteria	**L1**	**L2**	**L2**	**L2**	**L3**	**L3**	**L3**	**L4**
A	3	3	6	3	6	3	27	7
A'	25	22	26	47	26	27	87	33
C	*3*	*6*	25	*3*	*26*	27	47	87
C'	*3*	*6*	22	*3*	*26*	7	27	33
D	3	3	25	3	22	3	47	27
D'	6	6	26	27	26	7	87	33
F	*3*	*3*	6	25	6	26	22	26
F'	*3*	*3*	27	47	7	87	27	33

Table 9.5: Comparison of non-mandatory criteria for email client scenario

Analysis of the shortlist data can provide an exhaustive view of impacts across all criteria (Table 9.5). Each column represents a set at one of the levels of the graph (L1-L4) and the two values for each criterion (e.g. A = included and A' = not included) are the number of candidates in each resulting shortlist. The table confirms criterion A as having the most impact. In four cases, C has no impact (indicated in italics), and F has no impact in two cases.

This analysis could be done on a more intuitive level; however looking at the criteria through their impact calculations gives a guide that mandatory criteria {B,E} plus one or both of {A,D} should define the shortlist.

This scenario was intended to be the subject of the final case study. When the shortlisted software was downloaded, it was found almost all of them were neglected projects and not able to be installed. This indicates that date or maturity need to be focussed on with the freshmeat dataset. A contributing factor may be that email software is not often written in C++.

9.5.3 XML Editor

The scenario for this case study is the acquisition of software for editing and validating XML documents[8]. In this selection task there are six criteria, with two mandatory and four non-mandatory, resulting in sixteen sets in the suite. The selection criteria are summarised in Table 9.6. Both description (A) and Detail (B) are 'XML editor'. Other requirements are GNU GPL licence (C); Java for the development language (E); and, to be written for a Linux platform (F). Note that the maturity requirement (D) has been increased to 'Mature' for this scenario.

Criteria	Description	Value	Mandatory
A	Description	XML editor	N
B	Detail	XML editor	Y
C	Licence	GNU General Purpose License (GPL)	N
D	Development Status	6 - Mature	N
E	Development Language	Java	Y
F	Operating System	Linux	N

Table 9.6: Selection criteria for the XML editor

Viewing the graph in Figure 9.11, this set conforms to the properties in Section 9.4.4. The impacts for each of the non-mandatory criteria are shown in Table 9.7. Criterion F exhibits a low impact - with no impact in 3 cases. The highest impact is with criteria A and D, with C having slightly less impact. Although the impact statistics are helpful, the meaning of the criteria must also be considered. For example, this can explain the low impact of 'Linux' in this case as Java is a cross-platform language.

s21, s22 or s23 are options for selection. Drilling down into the respective shortlists would be the advised approach to finding the most appropriate criteria. Using this approach gives the confidence that all candidates have been assessed via the same criteria and are thus equivalent. Searching manually may not provide such objectivity.

The criteria are in the same order and priorities (mandatory) are the same for the XML editor and the email client. This gives an opportunity to compare the impact of the various criteria. The overall numbers of candidates are similar in both scenarios. Criterion C has more impact with the email client scenario than with the XML editor

[8]The XML case study is the basis of the full case study in Chapter 10.

(note smaller differences in C and C' in Table 9.7). Using a higher level of development status has had some impact and adjusting this value is a possible way to tune the results in either of the scenarios. Criterion A has high impact across all levels in the email client scenario; its impact is 2-fold with the XML editor. For the email client, using both description and detail is required for the candidates to be email software, rather than just relying on the description.

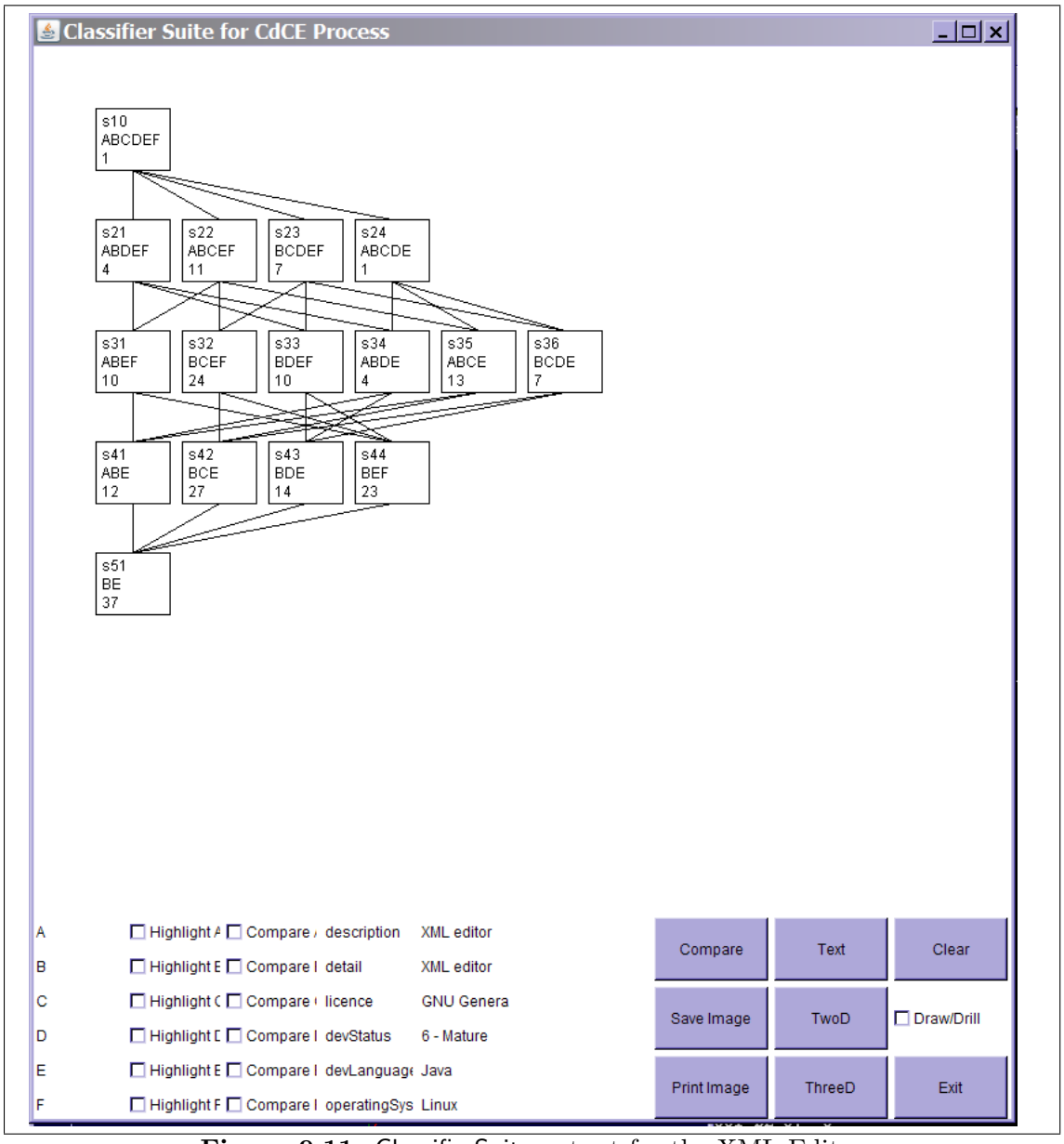


Figure 9.11: ClassifierSuite output for the XML Editor

Criteria	L1	L2	L2	L2	L3	L3	L3	L4
A	1	4	11	1	10	4	13	12
A'	7	10	24	7	23	14	27	37
C	1	11	7	1	24	13	7	27
C'	4	10	10	4	23	12	14	37
D	1	4	7	1	10	4	7	14
D'	11	10	24	13	23	12	27	37
F	1	4	11	7	10	24	10	23
F'	1	4	13	7	12	27	14	37

Table 9.7: Comparison of non-mandatory criteria for XML editor scenario

9.5.4 XML Editor with Date

This scenario involves seven criteria, two mandatory and five non-mandatory. It adds the date to the list of criteria as the email client scenario has shown that project age and development status are good indicators of whether an item can be downloaded and run. The criteria for this case are in Table 9.8.

Criteria	Description	Value	Mandatory
A	Description	XML editor	N
B	Detail	XML editor	Y
C	Licence	GNU General Purpose License	N
D	Development Status	6 - Mature	N
E	Development Language	Java	Y
F	Operating System	Linux	N
G	Date	1/1/05	N

Table 9.8: Selection criteria for the XML editor with date

This increases the number of sets and classifiers to 32 and generates a more complex graph. The set can be broken into two on a selected criterion to reduce complexity (as in Figure 9.3). In this exercise the graph will be kept whole (Figure 9.12) as the tool makes it easier to manage a 32 set graph. Highlighting criterion G can reveal the difference between the XML editor with and without date. The highlight and comparison features have increased usefulness at this scale to make the graph easier to read. In this case, the comparisons result in the values in Table 9.9. Criteria C and F do not have high impact in this scenario. Drill-down is required to see the underlying shortlists and understand the impact of criterion A.

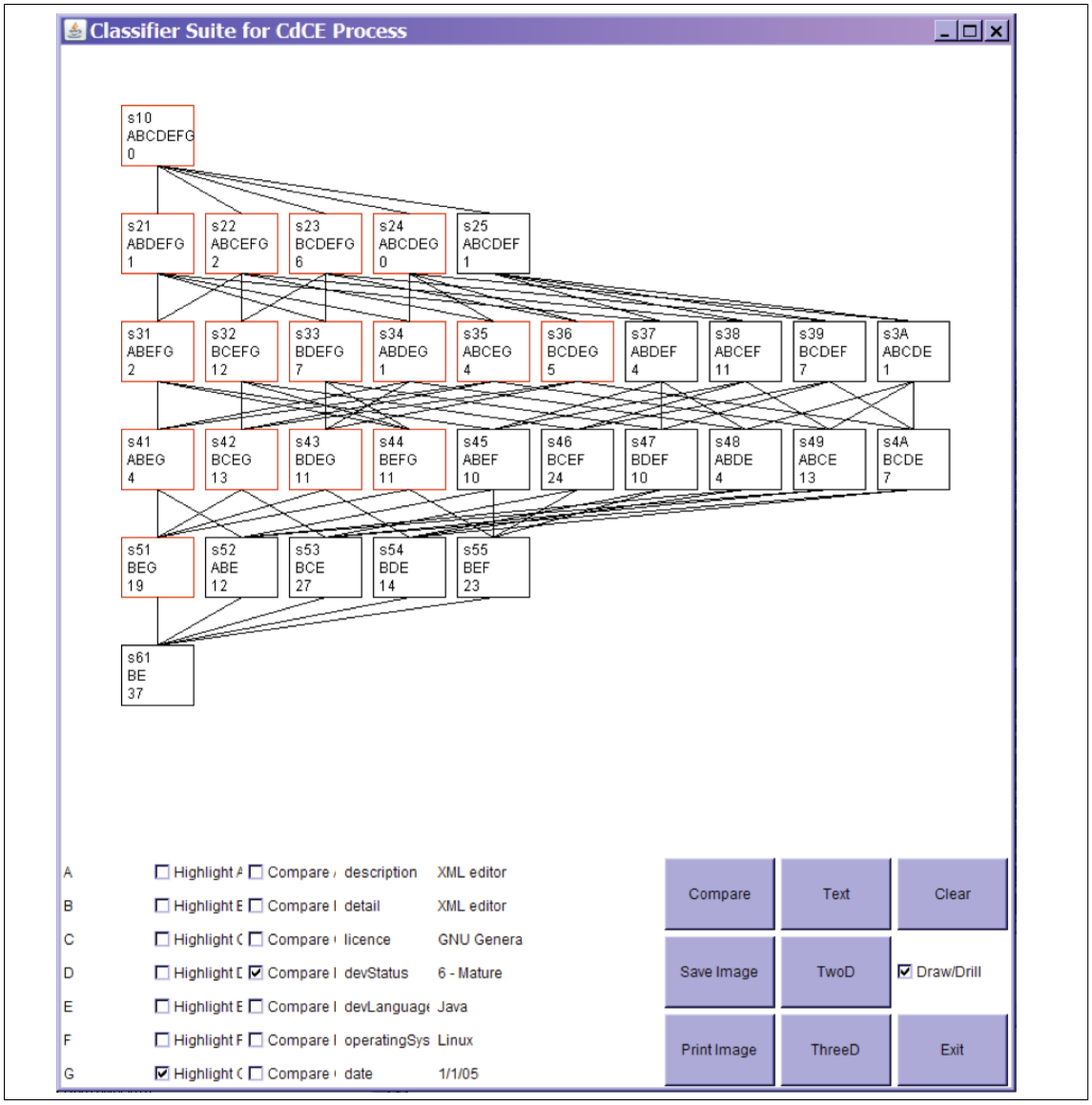


Figure 9.12: ClassifierSuite output for the XML editor with date scenario

Crit.	L1	L2	L2	L2	L2	L3	L3	L3	L3	L3	L3	L4	L4	L4	L4	L5
A	0	1	2	0	1	2	1	4	4	11	1	4	10	4	13	12
A'	6	7	12	5	7	11	11	13	10	24	7	19	23	14	27	37
C	0	2	6	0	1	12	4	5	11	7	1	13	24	13	7	27
C'	1	2	7	1	4	11	4	11	10	10	4	19	23	12	14	37
D	0	1	6	0	1	7	1	5	4	7	1	11	10	4	7	14
D'	2	2	12	4	11	11	4	13	10	24	13	19	23	12	27	37
F	0	1	2	6	1	2	12	7	4	11	7	11	10	24	10	23
F'	0	1	4	5	1	4	13	11	4	13	7	19	12	27	14	37
G	0	1	2	6	0	2	12	7	1	4	5	4	13	11	11	19
G'	1	4	11	7	1	10	24	10	4	13	7	12	27	14	23	37

Table 9.9: Comparison of non-mandatory criteria for XML editor scenario

From the data, criteria D and G seem to have similar impact. Drilling down shows that they are outputting quite different shortlists. As the email client scenario met difficulty with long dormant projects, having either of these variables is some insurance that the respective projects/software are usable and maintained. This scenario highlights the needs for and value of tools to support the selection of criteria for shortlists where there are larger numbers of criteria. The XML editor case study is discussed further in Chapter 10.

9.6 Spiral 6 Evaluation

The evaluation for Spiral 6 draws on the Win conditions in Table 9.2 and the Spiral goals in Table 9.1. The stakeholder Win conditions relate to the usefulness of the tool and the documentation it provides. From the application developer perspective, the tool allows a larger number of sets to be considered, and helps the user to reduce this number to make it more manageable. The case studies support the idea that the ClassifierSuite improves recall and relevance. In most cases using the ClassifierSuite results in different sets being selected then those in the manual process, even though the same sets could have been chosen. The ClassifierSuite makes options visible that may not have been considered. Although similar representations can be found for other applications (Ganter and Wille, 1997, Chen and Yao, 2008), supporting the validity of the ClassifierSuite, the technique is novel when applied to component selection. The tool not only assists the understanding of the data, it also provides a number of functions to help document and justify the decision-making process.

9.6.1 Spiral 6 Goals

The following discussion refers to the goals listed in Tables 9.10 and 9.11.

Quality

Guidelines for the use of the ClassifierSuite are provided including properties and interpretation tips. The ClassifierSuite works from an XML file, indexing existing files, and is run as a standalone Java application.

The rendering of the graph of the sets is fixed to the underlying classifier data.

SPIRAL 6	**Purpose** **Issue** **Object** **Context**	**Evaluate** **effectiveness of** **strategies to support decision-making in the shortlisting task** **Spiral 6**	**Results**
Goal 6A	Focus	**Quality**: *Provide tool support to allow user to have a better understanding of their choices and make more informed decisions*	
	Viewpoint	Quality Assurance personnel	
	Q6A1	Is the ClassifierSuite well documented?	YES
	Q6A2	Is the process repeatable?	YES
	Q6A3	Does the Suite aid understanding of data?	YES
Goal 6B	Focus	**Usability**: *Assist user in making decisions by providing visualisation of information and tool support*	
	Viewpoint	Application developer	
	Q6B1	Is the ClassifierSuite and procedures easy for the user to understand?	YES
	Q6B2	Has the work been tested on real world examples?	YES
	Q6B3	Has visual tool support and automation been provided?	YES
Goal 6C	Focus	**Intelligence**: *Give the user the combined value of all AI applied in previous spirals, with minimal overhead/complexity*	
	Viewpoint	Application developer	
	Q6C1	Does the ClassifierSuite reduce effort for the user?	YES
	Q6C2	Does the ClassifierSuite enhance the use of AI from the rest of the CdCE Process?	YES

Table 9.10: GQM Summary - Spiral 6 (Part 1/2)

The user has control of the selection criteria and analysis of the set, which can also be duplicated.

The researcher found diagrams similar to that produced via the ClassifierSuite were necessary for understanding the interaction between criteria. The ClassifierSuite extends this with facility for hiding criteria and for assessing the impact of criteria. Without the tool, this level of understanding would take hours, not minutes.

Usability

The ClassifierSuite is a standalone tool which provides a simple GUI to assist understanding of the data for shortlisting. The interface loads the data and has a combination of checkboxes and buttons for invoking functions. It also includes the attribute and value for each criterion; double-clicking on a set in the graph brings up the XML shortlist for that set.

After initial testing, the ClassifierSuite was applied to a series of existing case study

SPIRAL 6	**Purpose** **Issue** **Object** **Context**	**Evaluate** **effectiveness of** **strategies to support decision-making in the shortlisting task** **Spiral 6**	**Results**
Goal 6D	Focus	**Innovation**: *Provide an information rich interface to assist decision making*	
	Viewpoint	Academia	
	Q6D1	Have innovations been developed for visualising the selection choices?	YES
	Q6D2	Is the interface informationally rich?	YES
Goal 6E	Focus	**Dynamics**: *Allow user to explore the impact of changing parameters at a high level, but also drill-down to detail*	
	Viewpoint	Application developer	
	Q6E1	Does the ClassifierSuite allow for exploration of choices and their impact?	YES
	Q6E2	Can the user drill-down to complete information?	YES
Goal 6F	Focus	**Reuse**: *Where possible make use of existing code and artefacts*	
	Viewpoint	Application developer	
	Q6F1	Has the work reused external resources?	YES

Table 9.11: GQM Summary - Spiral 6 (Part 2/2)

data. These case studies originated as manual and tool-supported shortlisting tasks for the project.

The ClassifierSuite provides an effective visual tool to represent the sets generated during shortlisting. The tool automates comparisons and scenario checking for combinations of criteria.

Intelligence

Prior to the ClassifierSuite, exploration of the effects of changing selection criteria was manual. If the selection criteria were satisfactory from the start, then the ClassifierSuite would not reduce effort. However, if there is need for considering different options, particularly with large numbers of non-mandatory criteria, the ClassifierSuite will reduce effort.

Without the ClassifierSuite, the user is less likely to go through labour intensive sweeps of different sets of criteria. With the tool, the user is more likely to work with more sets of data, as it does not add much effort and can potentially show up more suitable results.

Innovation

Although the lattice visualisation is intuitive and seen in diagrams in other application areas, the additional functions are novel. Options such as highlighting criteria, running comparisons and the drill-down facility are task specific and only generated as ideas after the tools was developed.

The interface provides an abstract version of the shortlisting data which can be explored and manipulated. The user can also move from the abstraction to detail using the drill-down mechanism.

Dynamics

A key aim for the tool was to aid selection and impact assessment as this was found to be difficult and time consuming. The tool not only provides the visual support, it can give text output and the user can annotate the diagram and save it as part of the selection documentation.

The user can click on any set to bring up a text window containing the corresponding shortlist in XML. Multiple shortlists can be viewed simultaneously for easier comparison.

Reuse

The source data for the trials came from the previous case studies. XML manipulation code was reused from previous tool development. The Java code for print/save was adapted from the Internet.

9.7 Spiral 6 Review and Plan

Evaluation of the Spiral provided positive responses for both Win conditions and GQM evaluation. Advantages of using the ClassifierSuite include increased understanding of the options for varying the selection criteria, the low overhead of generating the criteria sets, classifiers and shortlists and the ability to make selection more visual and easier for the user to explore. While the previous approach of iterative refinement in Step 2 was of benefit, it was subjective in the choice of criteria to loosen/tighten. It also risked missing criteria sets of interest unless an exhaustive iteration of all permutations was undertaken.

Two main issues exist, the number of classifiers (and processing) required and the complexity of the data being interpreted. The size of the suite is dependent on the number of non-mandatory criteria, with the graph size equal to 2^n. There could be many improvements to the code for processing the data, which has been written in strict object-oriented Java. Where Transformation 5 (T5) is used, the run times are significantly longer, averaging 2 minutes per criteria set - 30 minutes for four non-mandatory criteria and one hour for five non-mandatory criteria. Scope exists for optimisation of the code for shorter run times, so the processing time is not considered a barrier.

An approach to dealing with large graphs is to divide the graph for interpretation and to isolate criteria which may be able to be removed or shifted to 'mandatory'. Each reduction in the size of the non-mandatory set halves the size of the graph (and processing time and complexity). The ClassifierSuite can assist in reducing subjective choices and improving awareness of a solution space which was already of the given size, but previously had no tools to assist the user to make decisions.

The ClassifierSuite has automated and added functionality when compared to manual graph generation. The application provides text and graphical views, as well as highlighting, comparing and drawing on the graph. This enhances the value of the approach and is extensible for adding new features and visualisation choices.

By viewing the criteria, shortlist size and drilling down in to the metadata for candidates on the shortlists, the user can see the impact of their choice of selection criteria and be more confident in the resulting shortlist. This empowers the user and gives more dimension to the data they base their decisions on, as well as making the process less subjective.

This Spiral provided an important gain in usability for the CdCE Process and may have applications in other selection situations. The ClassifierSuite was conceived as a result of the possibilities that present themselves once a process is automated and the scale of the data is removed from exploring scenarios. The outcomes of this Spiral were accepted in a paper for the IEEE Congress on Evolutionary Computation (CEC) in June 2008 (Maxville et al, 2008).

This completed the strategy development for the Process. Future work can be flagged to include other repositories[9]. The investigation of strategies is thus complete and the

[9]An attempt was made to use the SourceForge repository, however, technical issues with recreating the PostGres database and then extracting a flat file from all of the tables.

commitment is to applying the Process to a final case study.

9.8 Summary

This chapter described the ClassifierSuite approach to selecting software. The automation of classifier generation and the use of the ClassifierSuite tool make it possible for users to visualise and explore the criteria and shortlists. Four examples were given of the ClassifierSuite in use, highlighting different functionality and analysis features that the tool provides for the user. The ClassifierSuite and tool support allows for a more informed selection process and better understanding of the interactions and impact between criteria in a given repository.

The contribution of this Spiral is the ClassifierSuite tool for decision support (C7), and the process information guiding its use. The tool makes it possible to explore and analyse potential shortlists, which can improve efficiency and makes it possible to consider larger criteria sets.

With the completion of the development Spirals, the next stage of the investigation is a case study. The next Chapter presents the CdCE Process case study, the focus of Spiral 7.

Chapter 10

CdCE Process Results

The CdCE Process provides a framework for the evaluation of third party components and software. This chapter examines the results of using the process. During development, a number of case studies were utilised: these have been presented in previous chapters and published for peer review. As a final case study, another real world selection problem is explored - sourcing an XML editor - in order to apply the complete CdCE Process and tools.

The goals of this final Spiral are listed in Table 10.1. To meet the goals, the case study needs to exercise all parts of the Process (**quality**). Taking a real world scenario for evaluation allows the researcher to view the Process from the user perspective. **Usability** and low effort are the desired outcomes.

10.1 Spiral 7 Overview

In this Spiral, the software, process, specification and dataset are applied to a full case study. The scenario needs to be selected to ensure that all aspects of the process can be exercised, so some pre-work was required to select a strong case study - the XML Editor.

SPIRAL 7	GOALS
Quality	Exercise the whole process in the case study, evaluate according to industry standards
Usability	Evaluate the process and tools in terms of real world requirements
Intelligence	N/A
Innovation	N/A
Dynamics	N/A
Reuse	N/A

Table 10.1: Goals for Spiral 7

The use cases in this Spiral are **Select Component** (application developer) and **Assess Selection** (quality assurance) (Figure 10.1).

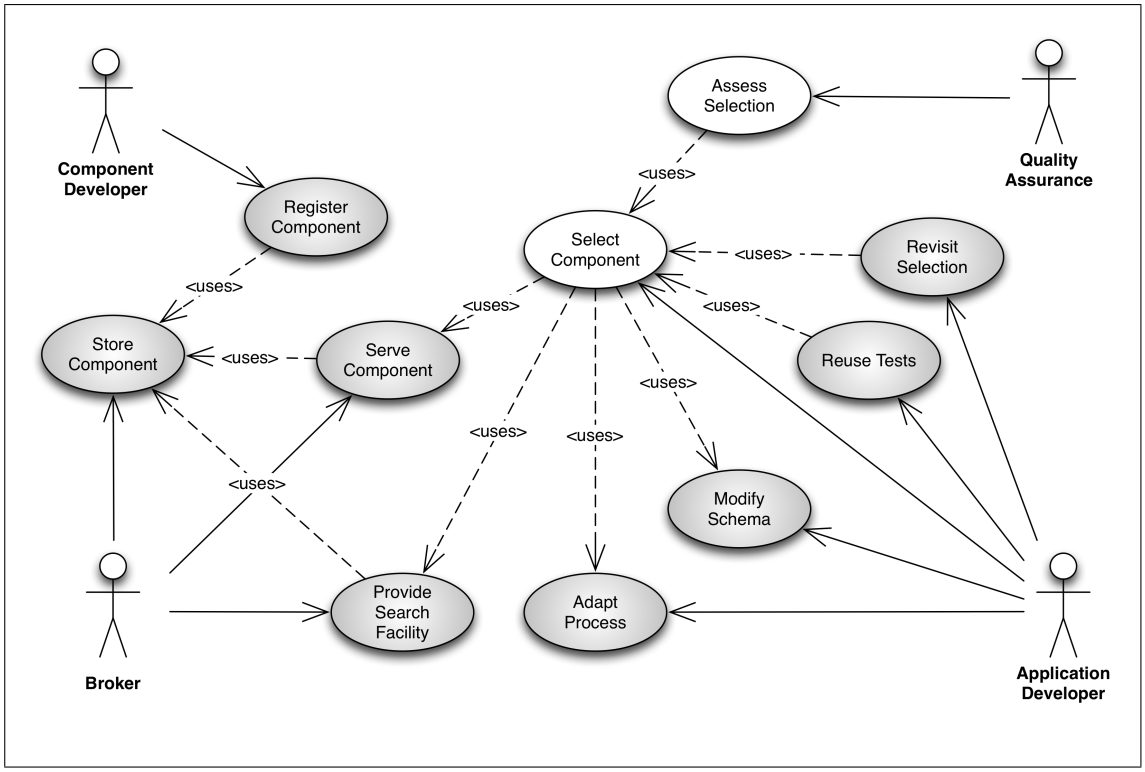


Figure 10.1: Use cases for component selection, the focus of Spiral 7 (those not in the scope for this Spiral are greyed)

As stakeholders in this Spiral, the application developers would want to see that the case study problem is relevant to their assessment problems and that the benefit of the whole approach is clear. Academia also requires a representative problem, along with useful criteria and the identification of issues and further work. For component developers, this Spiral provides a full picture of how their software may be shortlisted

Stakeholder	**Win Conditions**
Application Developers	Strategies are beneficial Representative problem
Component Developers	Know how their component is assessed and compared
Academia	Representative problem Assessment criteria Peer reviewed Issues and further work

Table 10.2: Win conditions for stakeholders (Spiral 7)

and then evaluated. This information may be used to target improvements in metadata and documentation for potentially increased uptake of their product. The stakeholder evaluation criteria are provided in the table of Win conditions (Table 10.2).

10.2 Spiral 7 Context

The CdCE Process (Figure 5.2) has been developed over six Spirals, culminating in this case study Spiral for evaluation. The case study will draw on the freshmeat repository as the source of candidates for selection.

No new risks or instruments are introduced in this Spiral. The instruments used are listed in Table 10.3, representing the final tool suite developed to support the CdCE Process. In the table, 'Bash shell scripts' includes all automation scripts developed for training, shortlisting and working with the ClassifierSuite. The software has been run under Cygwin, on CentOS running under VMware on a MacbookPro and through the Mac OS X terminal. The work that follows is primarily carried out under CentOS.

Item	Description
XML schema	Schema to describe ideal and candidate components
XSLT scripts	Scripts to reformat the XML files and make them more readable for the user
Intelligent	Java program developed to read in XML ideal specification and output training data in Weka's ARFF format
CdCETransformer	Java program developed to read in XML ideal specification and real world data and output the data in Weka's ARFF format
FM2CdCE	Java program developed to read in XML real world data (freshmeat) and output the data in CdCE XML format
TestGen	Java program developed to take in the technical specification (Z notation) and generate a test suite based on equivalence classes and boundary value analysis
ClassifierSuite	Java program to visualise and explore results of running multiple classifiers to see the impact of criteria choices
Bash shell scripts	Scripts written in Bash Shell to automate the processing of data and the collation of results

Table 10.3: Instrumentation used (Spiral 7) - developed as part of this project

In parallel to the execution path, there is a trail of documents, most of them XML. Some of the documents are parameter files, some are the input data (ideal, repository and ontology files) and the output data. All output files have an associated XSLT template to allow formatted viewing through a compatible browser. The final file format used is

ARFF input for Weka, which gives a text file as output (captured stdout).

All aspects of the Process have been evaluated prior to this case study, thus the risk in this Spiral is mainly in the choice of case study. In the first instance, there was a problem with that choice. When taking the Emailer further through the Process, difficulties arose in the download and installation of the software. In most cases it was not possible to successfully run the downloaded software, which may have always been faulty or had aged to a point of not being usable. This situation forced the development of a new scenario - the XML Editor, which is described in this chapter.

10.3 Spiral 7 Case Study

The scenario for this case study is the sourcing of software for XML editing and validation. The target environment is a Linux platform and the preferred language is Java. Previous case studies with the freshmeat repository have shown that maturity and date of last update are critical in finding runnable applications. This case study looks for production level projects and a 2005 cutoff on last update. It also requires a GPL licence and will start with the search terms, 'XML editor', included in both description and detail. The mandatory criteria are detail and development language. Table 10.4 summarises the criteria for this case study.

Criteria	Description	Value
A	Description	XML editor
B	Detail	XML editor
C	Licence	GNU General Purpose License (GPL)
D	Development Status	6 - Mature
E	Development Language	Java
F	Operating System	Linux
G	Date	1/1/05

Table 10.4: Selection criteria for the XML editor with date

10.3.1 Step 1 - Specification

The scenario description defines much of the ideal specification. Also added are non-functional requirements for price, disk space and memory. The XML in Figure 10.2 shows

```
<?xml version="1.0"?>
<Description xmlns="http://www.scis.ecu.edu.au/swvML/1.0/"
          xmlns:dc="http://purl.org/dc/elements/1.0/"
          xmlns:swv="http://www.scis.ecu.edu.au/swvML/1.0/" >
     <dc:description  type="mandatory">XML editor</dc:description>
     <dc:detail type="mandatory">XML editor</dc:detail>
     <swv:licence type="mandatory">GNU General Public License (GPL)</swv:licence>
     <swv:devStatus type="mandatory">6 - Mature</swv:devStatus>
     <dc:date type="mandatory" min="01-01-2005" max="31-12-2007">31-12-2007</dc:date>
     <swv:technical>
             <swv:devLanguage type="mandatory">Java</swv:devLanguage>
             <swv:operatingSystem type="mandatory">Linux</swv:operatingSystem>
             <swv:systemRequirements>
                     <swv:memory type="mandatory" min="15" max="50">20</swv:memory>
                     <swv:diskSpace type="mandatory" min="30" max="50">40</swv:diskSpace>
             </swv:systemRequirements>
     </swv:technical>
 </Description>
 </xml>
```

Figure 10.2: Initial ideal specification for case study

this non-functional side of the ideal specification which will be used for shortlisting.

Next to be defined is the behaviour and functionality required. This could be very detailed if fully specifying an XML editor, but as the selection task only needs specific external behaviour, the formal specification will include operation schemas to:

- Load a document
- Check well-formedness
- Validate against a DTD
- Validate against a schema
- Transform XML document via XSLT.

The formal specification in Z notation is in Section 10.3.3. Each item of functionality is modelled as a Z operation schema. For each of the operations, the input variables are defined, with types that include the partitions we are interested in testing. For example, the validate against DTD schema operation may have three partitions on the input file type: no errors; error in file; error with DTD reference. Each of these can have one or more input files for test data. The product of the number of partitions on each of the types determines the number of combinations being tested, and thus the number of test cases. As each of the operations in this case study only has one input variable, there are a total of twelve test cases. Each test case can be rerun with different data, so in reality more than twelve tests are likely to be run.

With the required behaviour defined, the ideal specification is completed by stating the functional fit and test performance. As with the non-functional attributes, these criteria can be loosened, depending on the actual results. There are nine metrics for evaluation: `FFIT`, `FEXS`, `AEFT`, `TFIT`, `TRES`, `CX_P`, `CX_R`, `CX_S` and `CX_U` which are scored out of 10 (see Section 8.2 for more detail on metrics). These user-defined values are shown in Table 10.5.

Metric	**FFIT**	**FEXS**	**AEFT**	**TFIT**	**TRES**	**CX_U**
Value	8	8	6	6	10	8

Table 10.5: Initial metrics for XML editor case study

These thresholds on the metrics require high performance in most areas. On functionality, near full compliance (`FFIT` = 8) is required with little excess functionality (`FEXS` = 8). There can be some amount of work to install the software and adapt the tests, so `AEFT` is 6/10. The testing fit (`TFIT` = 6) is somewhat dependent on functional fit (`FFIT`), but where the tests do run, all must be passed (`TRES` = 10). The functionality that will be most used (`CX_U` = 8) will require will need to pass 80% of tests.

10.3.2 Step 2 - Shortlisting

To begin the shortlisting, the full, most restrictive ideal specification is used with all attributes set to mandatory. Data transformation **T5** is selected from the available transformations as previous investigations have indicated it provides improved shortlists through the use of distance measures and ontologies for abstraction of some text attributes.

The initial ideal specification is used to generate training data for the classifier. This provides input to Weka and a C4.5 classifier is saved as a predictive model. The repository data is transformed based on the ideal specification and the selected transformation (T5). The transformed input file is run through the classifier, resulting in an empty shortlist. Analysis via the Weka statistical tools shows that there is a high level of missing data on the `memory` and `diskSpace` attributes. These are removed from the ideal specification

and two criteria are flagged as mandatory - `detail` and `devLanguage` (Java was specified in this scenario).

At this point the automated tools developed for the CdCE Process are used to generate a suite of classifiers for the combinations of sets of criteria. In the graph (Figure 10.4) the `date` criterion is overlaid as a second value in each node. The level 1 node (S10) has only one item in its shortlist. As the aim is to gain the highest relevance of results while not having too many candidates on the shortlist, the next level is examined, which involves loosening criteria. In levels 2 and 3, there are a few set counts below ten, which is manageable. By level 4 the relevance of candidates is reduced, so the focus is on the higher levels (1-3).

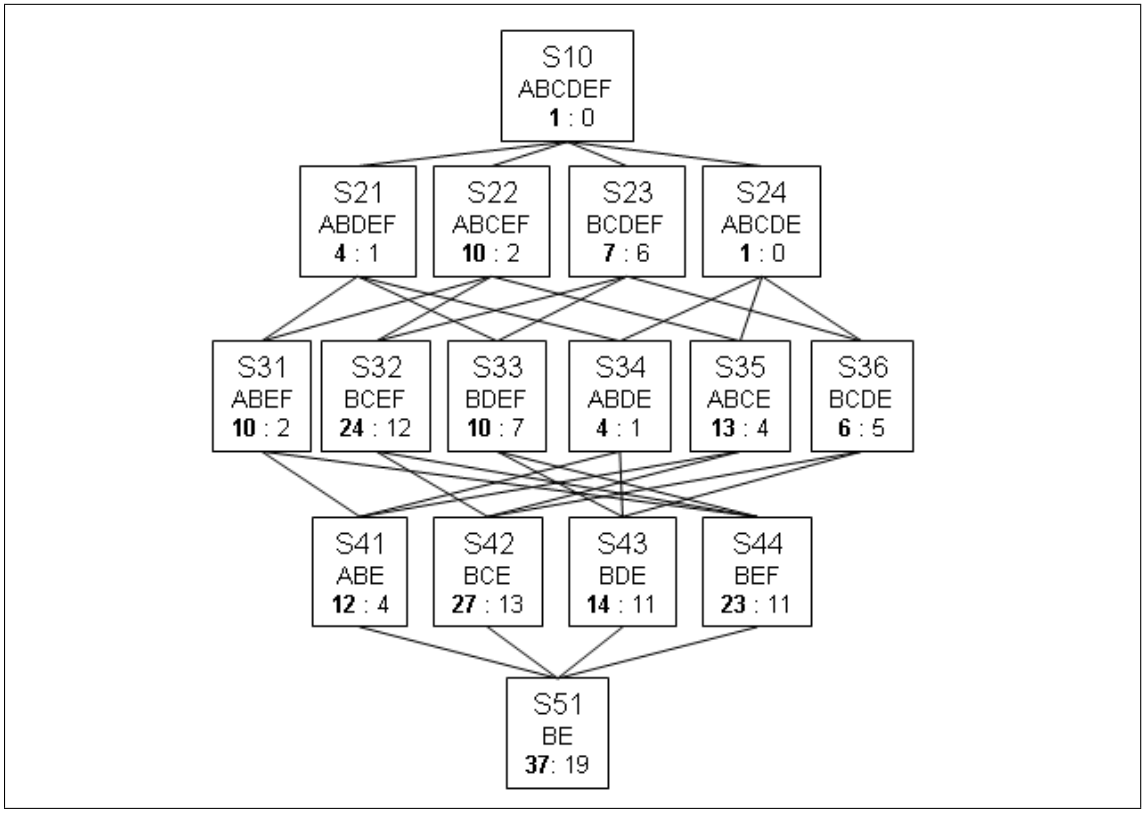


Figure 10.4: Graph representation of case study shortlists. Date (G) is overlaid on the graph. Left count = without date, right count = with date.

By drilling down to the metadata it was found that the `description` criterion is critical to relevance of results and should be mandatory. This allows the removal of eight nodes: S23, S32, S33, S36, S42, S43, S44 and S51 from the graph to be considered. Two nodes: S21 and S34 are identical, which also helps to reduce options. Choosing S21 without `date` puts four items on the list. Previous case studies with the freshmeat

dataset have shown that `date` and `devStatus` are important for good projects. As `date` is not in the set S21, a second set of criteria with `date` will be selected. Nodes with this criterion are targeted along with as many other criteria as possible (the right number of the graph). S22-date[1] and S24-date have 2 and 0 items respectively, which is less than would be preferable[2]. The choice is then between S31-date and S35-date. As S31-date is closely related to S21+date, S35-date is selected to give more variety, adding four items. Two four-item shortlists are accepted - S21-date, S35+date, giving a total of eight candidates with these changes.

10.3.3 Step 3 - Generate Tests

In Step 1 the functional requirements were defined in Z notation. This is not a complete specification as its role is to drive the test case generation. This test directive is given in Z below:

INfile ::= fileName

XMLfile ::= notWellFormed | wellFormed

DTDvalidation ::= notValidDTD | notFoundDTD | validDTD

SCHEMAvalidation ::= notValidSCHEMA | notFoundSCHEMA | validSCHEMA

XSLTfile ::= noTrans | transError | transOK

BOOLEAN ::= true | false

XMLEditor

xmlFile : XMLfile
cssFile : CSSfile
xsltFile : XSLTfile
isWellFormed : BOOLEAN
isValidDTD : BOOLEAN
isValidSCHEMA : BOOLEAN
transOK : BOOLEAN

[1] S22-date refers to set S22 without the date criterion included {A,B,C,E,F}, S21+date is S21 with the date criterion included {A,B,D,E,F,G}.

[2] As the researcher has experienced low quality software from this repository, the aim was for five to ten candidates on the shortlist.

LoadFile

$\Delta xmlFile$
$\Delta transOK$
$NEWfile? : INfile$

$xmlFile = NEWfile?$
$transOK' = true$

IsWellFormed

$\Delta isWellFormed$
$\Xi XMLfile$
$NEWxmlFile? : XMLfile$

$isWellFormed' = validateForm(NEWxmlFile?)$

IsValidDTD

$\Delta isValidDTD$
$\Xi XMLfile$
$NEWxmlFile? : DTDvalidation$

$isValidDTD' = validateDTD(NEWxmlFile?)$

IsValidSCHEMA

$\Delta isValidSCHEMA$
$\Xi XMLfile$
$NEWxmlFile? : SCHEMAvalidation$

$isValidSCHEMA' = validateSchema(NEWxmlFile?)$

TransformFile

$\Delta transOK$
$\Xi XMLfile$
$NEWxmlFile? : XSLTfile$

$isWellFormed' = validateForm(NEWxmlFile?)$
$transOK' = true$

To prepare the Z specification for the test generator, the specification is distilled to extract the required information.

The distilled specification is loaded into the test case generator software, TestGen. The

resulting tests are based on partition information in the Z specification and matching test data can also be pulled in from the specification.

The test cases are listed in XML in Figure 10.5. To make the recording of results consistent, there is an XSLT transformation which converts the XML test cases into HTML forms to record results for each candidate. Page one of the form records the software ID, title, tester name and test date. From there the tests for each interface are listed, with the variables to be used for each parameter, as generated by TestGen. The final page summarises the tests and is where `TRES` and `TFIT` are calculated. That completes the base functional tests.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE TestSuite SYSTEM "testsuite.dtd">
<?xml-stylesheet type="text/xsl" href="testsuite.xsl"?>

<TestSuite name="myTest">
      <Op name="LoadFile">
            <Variable varNo="1" name="NEWfile" type="INfile"/>
            <TestCase id="1">
                        <Value varNo="1">fileName</Value>
            </TestCase>
      </Op>
      <Op name="IsWellFormed">
            <Variable varNo="1" name="NEWxmlFile" type="XMLfile"/>
            <TestCase id="1">
                  <Value varNo="1">notWellFormed</Value>
            </TestCase>
            <TestCase id="2">
                  <Value varNo="1">wellFormed</Value>
            </TestCase>
      </Op>
      <Op name="IsValidDTD">
            <Variable varNo="1" name="NEWxmlFile" type="DTDvalidation"/>
            <TestCase id="1">
                  <Value varNo="1">notValidDTD</Value>
            </TestCase>
            <TestCase id="2">
                  <Value varNo="1">notFoundDTD</Value>
            </TestCase>
            <TestCase id="3">
                  <Value varNo="1">validDTD</Value>
            </TestCase>
      </Op>
      <--- snip --->
</TestSuite>
</xml>
```

Figure 10.5: Test specification for XML editor case study

There are also context based tests. In this study they are usage tests in the `CX_U` schema. These tests allow greater emphasis to be put on the functionality that will be used most (e.g. validating a file against a DTD) in this case study.

10.3.4 Step 4 - Adapt Tests

The abstract tests generated are then ready to be adapted to each of the shortlisted candidates. The adaptation stage includes downloading and installing each candidate. In a component selection scenario, this would require a test harness. For this project the candidates are manually installed and tested. The adaptation effort metric, `AEFT`, is based on the difficulty encountered when installing and working with the software to a point that the tests can be run. The eight shortlisted candidates had various issues which impacted their rating.

Table 10.6 lists the candidates, their scores out of ten and a comment on the difficulties/ease of use. The target environment was CentOS running in VMware on a MacBook Pro. Where the install failed, the installation was attempted in Cygwin running on a Windows XP Thinkpad. Third option was Windows XP, then Mac OS X. Each failed attempt impacted negatively on the `AEFT` score by 2 points. Where quite an effort was required, the comment states that it 'eventually worked' in the target environment.

Candidate	AEFT	Comment
A	8	Fixable problem when installing in target environment
B	10	Installed easily according to instructions
C	10	Installed easily according to instructions
D	4	Installer problems in CentOS and Cygwin, eventually worked on Windows XP
E	8	Did not run under CentOS, worked with Cygwin on PC
F	4	Installer problems in CentOS and Cygwin, eventually worked on Windows XP
G	4	Installer problems in CentOS and Cygwin, eventually worked on Windows XP
H	0	Commercial software - originally open source, no trial version available

Table 10.6: Adaptation results for XML editor candidates

10.3.5 Step 5 - Execute

The test cases from Step 3 included sample data for each partition. These were variations on the test case XML file with errors inserted to test correct functioning of well-formedness, validation and transformation. The results for Candidate G are shown in Table 10.7 which passed six of the twelve tests and failed three (IsValidDTD). An abbreviated version of the test results form for Candidate G is given in Figures 10.6 to 10.7. As

Candidate G had no functionality for the XSLT transformation, the three related tests were skipped (Transform).

Table 10.8 includes the results summaries for all of the candidates. The NumSkipped column provides valuable information for assessing actual functionality. The candidates fall into three categories: full or near-full functionality (D,E,F,G), minimal functionality (A,B,C) and not assessed (H). Candidate H could not be installed, thus there are only results for seven of the candidates. In the number of tests passed column the same candidates perform well. The test results are used to generate the metrics `TFIT`, `TRES` and `CX_U` in Table 10.8.

Test Suite: myTest

Software ID	Software Title
Candidate G	withheld
Tester Name	**Test Date**
VM	9th November

Operation: ***IsWellFormed***

VarNo	Variable	Type
1	NEWxmlFile	XMLfile

Test Case	Values Used		Result
1	VarNo: 1	notWellFormed	PASS
2	VarNo: 1	wellFormed	PASS

	NumTests	NumSkipped	NumPassed
Subtotals	2	0	2

Operation: ***IsValidSCHEMA***

VarNo	Variable	Type
1	NEWxmlFile	SCHEMAvalidation

Test Case	Values Used		Result
1	VarNo: 1	notValidSCHEMA	PASS
2	VarNo: 1	notFoundSCHEMA	PASS
3	VarNo: 1	validSCHEMA	PASS

	NumTests	NumSkipped	NumPassed
Subtotals	3	0	3

Figure 10.6: Form for recording results of tests (Page 1/2)

Test Results:

Operation	NumTests	NumSkipped	NumPassed
LoadFile	1	0	1
IsWellFormed	2	0	2
IsValidDTD	3	0	0
IsValidSCHEMA	3	0	3
TransformFile	3	3	0
Totals	**12**	**3**	**6**
TFIT	**NumTests - NumSkipped / NumTests**		**_12-3_ / _12_ = _9_ / 12**
TRES	**NumPassed / NumTests**		**_6_ / _12_ => _5_ / 10**

Figure 10.7: Form for recording results of tests (Page 2/2)

Operation	NumTests	NumSkipped	NumPassed
LoadFile	1	0	1
IsWellFormed	2	0	2
IsValidDTD	3	0	0
IsValidSchema	3	0	3
Transform	3	3	0

Table 10.7: Raw test results for Candidate G

Cand.	NumTests	NumSkipped	NumPassed	TFIT	TRES	CX_U
A	12	9	2	3	2	4
B	12	9	0	3	0	0
C	12	9	3	3	3	5
D	12	0	12	10	10	10
E	12	3	5	9	5	8
F	12	0	12	10	10	10
G	12	3	6	9	5	10
H	0	12	0	0	0	0

Table 10.8: Test results for shortlisted candidates

10.3.6 Step 6 - Evaluate

At this point the metrics for each of the candidates can be collated. Table 10.9 lists the eight candidates and their results against each of the metrics. These results are added into the XML file for all the shortlisted candidates in preparation for Step 7. The columns for `FFIT`, `TFIT` and `TRES` would theoretically match if the documented functionality was usable and performed correctly. It may be the nature of the repository, but there are marked variations in these three metrics, except for Candidates D and F.

Candidate	FFIT	FEXS	AEFT	TFIT	TRES	CX_U
A	6	10	8	3	2	4
B	8	10	10	3	0	0
C	4	6	10	3	3	5
D	10	6	4	10	10	10
E	8	10	8	9	5	8
F	10	6	4	10	10	10
G	8	10	4	9	5	10
H	0	0	0	0	0	0

Table 10.9: Scores against metrics for shortlisted candidates

10.3.7 Step 7 - Rank

The metrics from the evaluation are numeric from 0-10, but there is no common unit to allow them to be meaningfully aggregated. To decide on the ranking, a classifier is used, as described in Section 8.5. For Step 7, the classifier focusses on the metrics attributes and using the ideal ranges for each as specified by the user in Step 1 as initial values for the six metrics (first row of Table 10.10). This new classifier is trained and the candidates are classified. On the first pass, none of the candidates meet the ideal specification. Unlike Step 2 of the process, where criteria were included/excluded, this time the loosening is done by changing the acceptable ranges. This approach could also be used to loosen criteria in Step 2, where appropriate. The final values used for this case study are given in the bottom row of Table 10.10.

Metric	FFIT	FEXS	AEFT	TFIT	TRES	CX_U
Initial Value	8	8	6	6	10	8
Final Value	8	6	5	6	8	8

Table 10.10: Final values for evaluation metrics

After the second pass, there are two clear recommendations - Candidates D and F. Beyond these two there are two more that would be next best ranked - Candidates E and G. The other four performed poorly and would not be recommended. An issue with the top two is their licensing. Both programs are from the same company, one released under an academic licence and the other as commercial. Where licensing restrictions exclude the top two, Candidates E and G should be considered.

10.3.8 Step 8 - Report Results

To provide a record of the selection process, all artefacts and input files are archived, along with versions of the ideal specification, test cases, shortlists, decision trees, test results, evaluation and ranking discussion.

The total time taken for this case study was fourteen hours, including computation time. Specification for the XML editor took four hours, with another three hours required for the shortlisting step (computation time and analysis using the ClassifierSuite. Developing the Z specification, XML files for test data and generating the tests took two hours. Once this was complete, the software was installed and the adaptation effort recorded. The forms created by the test generation were used to record the performance of the software, with Steps 4-8 taking around five hours.

10.4 Spiral 7 Evaluation

This Spiral provides input to **RE4**, where the Process and strategies are to be evaluated. The stakeholder Win conditions were listed in Table 10.2. For both the application developer and academia, the choice of problem needed to be representative. There are two ways to look at this - did it represent a realistic problem, and did it exercise the Process to show its value? The answer in both cases is yes. Application developers are interested in whether the strategies developed throughout the investigation would be beneficial to them. The researcher believes that the range of strategies, tools and solutions has value as a suite, but also as individual ideas to transfer into a different environment. The case study showcases the final version of the evaluation and this can inform component developers who have an interest in how their software is assessed.

Returning to the academic perspective, the focus turns to assessment criteria, peer

SPIRAL 7	**Purpose** **Issue** **Object** **Context**	**Evaluate** **effectiveness of** **case study and evaluation** **Spiral 7**	**Results**
Goal 7A	Focus	**Quality**: *Exercise the whole process in the case study, evaluate according to industry standards*	
	Viewpoint	Quality Assurance personnel	
	Q7A1	Does the case study exercise the entire process?	YES
	Q7A2	Does the evaluation use an industry standard?	YES
Goal 7B	Focus	**Usability**: *Evaluate the process and tools in terms of real world requirements*	
	Viewpoint	Application developer	
	Q7B1	Has the evaluation worked with real world data?	YES
	Q7B2	Has the work been tested on real world examples?	YES

Table 10.11: GQM Summary - Spiral 7

review, issues and future work. In terms of assessment criteria, all Spirals have been assessed through real world application, then matching against Win conditions and GQM assessment. This is a thorough and appropriate evaluation. All Spirals have also been exposed to peer review, including the outcomes of this keystone case study (Maxville et al, 2009). Issues and future work will be discussed in Section 10.5 and again in the Conclusion. It is considered that the Win conditions for Spiral 7 have all been satisfied.

10.4.1 Spiral 7 Goals

For this Spiral, there are two goals that are relevant: **quality** and **usability**.

Quality

The XML Editor case study provided eight strong candidates to exercise the shortlisting and ClassifierSuite. A behavioural specification was developed along with XML files for test data to represent each equivalence class. From this the testing and evaluation provided a thorough comparison and showed the complete process in action. As with all Spirals in this investigation, the GQM approach has been used to evaluate the outcomes.

Usability

Throughout all Spirals the investigation has been based on real world data. Along with the data real world scenarios have been used for the case studies: in this case the scenario was the selection of an XML editor. In going through the selection process, the researcher

found that many of the tasks within selection had been made much easier through the selection strategies and tools that had been developed. The self-documentation of the Process provided all the resources required to discuss the selection scenario in this Chapter.

10.5 Spiral 7 Review and Plan

The case study has satisfied all of the requirements for stakeholder Win conditions and the GQM. The results were externally reviewed through publication in *IET Software* (Maxville et al, 2009). The main contribution of this Spiral is the synthesis of the strategies and procedures across the investigation. It switches the focus from strategy development and implementation to the user experience of the Process. This keystone case study completes the project and feeds into the conclusions in Chapter 10.

10.6 Summary

This Chapter explored the XML Editor case study to exercise all strategies, tools and procedures in the CdCE Process. As the seventh and final Spiral, it brings together the cumulative contributions across all six preceding Spirals, showing that they can work together to address the initial research problem.

The initial scenario for the case study met with difficulties due to poor quality of the shortlisted candidates. This provided lessons in the software quality and vitality of open source projects in the selected repository. It also became clear that installation and execution were required to assess a candidate, as top candidates based on metadata did not always translate to top results overall.

The revised scenario of the XML Editor allowed a fresh dataset to be used. The Emailer scenario had been explored in previous Spirals. The final case study provided an unknown selection task and the timings involved were thus representative of a real-world selection scenario.

The full case study allowed the researcher the opportunity to experience the Process from the user perspective. This included the global context of defining the requirements and relevant criteria for a new selection task. The value of strategies and tools became apparent, while remaining manual tasks could be considered in terms of keeping them

manual or automating them as future work. It also helped identify implications and recommendations which are discussed in the following chapter: Contributions of the Study.

Chapter 11

Contributions of the Study

This thesis has presented the investigation of strategies for the intelligent selection of components. The work has explored four Research Elements across seven Spirals of activity. Each Spiral chapter includes an evaluation in terms of the six goals set for the investigation. In this Chapter, the conclusions, implications and recommendations resulting from the entire study will be discussed.

11.1 Conclusions

The problem addressed in this thesis is:

> *What strategies and techniques can be developed to support the selection of third party software components?*

The focus has been the development of fundamental strategies to enable the intelligent selection of suitable components from all available components. This was approached through the Spirals, with each focussing on one or more Research Element. The findings from each bundle of work were reviewed within each Spiral, and the conclusions based on the findings follow. There may be alternative explanations for these findings, which are discussed in Section 11.1.2. In some aspects of the investigation, the ideal approach was not possible. These limitations on the study are outlined in Section 11.1.3. Based on the conclusions, explanations and limitations, Section 11.1.4 addresses the impact of the study.

11.1.1 Conclusions Based on the Findings

The study considered four Research Elements as outlined in Chapter 1. The investigation and resulting conclusions are now viewed in turn.

Research Element 1

> ***Development or extension of a template for the specification of components (RE1)***

The template for characterising components was the focus of Spiral 1 of the investigation. The approach taken allows the template to be the driver of the selection process, as well as the transformation format for repositories. The template conforms with the Dublin Core standard for describing electronic resources. The specification template has been successfully applied and enhanced throughout the investigation, enabling evaluation of candidates on functional and non-functional criteria. The use of XML and XSLT for the template and other documents throughout the CdCE Process has provided consistency, flexibility and transparency by encoding the data model in the de-facto standard for data interchange.

Conclusion 1: *The template was able to successfully characterise components for use in the CdCE Process.*
Documented in Chapter 4.

Research Element 2

> ***Development of a process for the selection of software components (RE2)***

Spiral 2 of the study centred on the development of a process for the selection of software components. This was an initial, manual version of the CdCE Process which was implemented, automated and applied through Spirals 3-7. The three-phase approach of shortlisting, evaluating and ranking proved appropriate with the addition of feedback loops/iteration for tuning the ideal specification in response to the results presented. After applying it on a variety of case studies, it has shown itself to be generally applicable.

Conclusion 2: *The CdCE Process provided a structured, repeatable process for selection and evaluation of software and was able to be applied across a range of case studies.*
Documented in Chapter 5 and applied in Chapters 5 to 10.

The approach taken also aimed to provide a pattern for selection in a more general sense. Aspects of the Process can be replaced or exchanged, a task made easier through the clarity of the XML files for input and output.

Conclusion 3: *The investigation has developed a reusable framework for selection, and shown its instantiation for software selection in the CdCE Process.*
Documented in Chapter 5, Section 8.

Research Element 3

Investigation of and implementation of strategies for the shortlisting and evaluation of suitable software components (RE3)

With the specification and Process in place, the study changed focus to the development of strategies to apply intelligence and automation to improve quality and scaling. Although other AI approaches were considered, the C4.5 machine learning classifier was selected as most appropriate based on its transparent representation of decisions via the resulting decision tree, the high accuracy of results and training ability.

Conclusion 4: *The C4.5 classifier was effective in classifying the list of components and creating a shortlist.*
Documented in Chapter 6.

Enhancements to the data representation were investigated and implemented in Spiral 4. These were selected in the review of Spiral 3 as the next strategies to explore. The new work included re-defining attribute types, developing a range of transformations to better utilise the data, implementing ontologies and abstraction and formalising the handling of missing data. Comparative studies were undertaken and showed that these enhancements improved recall and relevance.

Conclusion 5: *Enhancements to the representation of data improved the recall and relevance of shortlists.*
Documented in Chapter 7.

With the procedure and tool support for shortlisting well-advanced, Spiral 5 considered the evaluation and ranking phases of the Process (Steps 3 to 8). Metrics were developed to represent the results of the static and dynamic evaluation. A formal specification in Z was used to encode the desired behaviour and provide a test directive to the abstract test generator. A single test suite was prepared for all candidates to allow a better comparison than developing tests for each candidate separately. An adaptation model was developed to enable the customisation of the abstract tests for each candidate. The information from adaptation and testing were collated into metrics. A second classifier was trained and used to rank the candidates.

Conclusion 6: *Abstract test case generation from behavioural specifications provide a meaningful comparison of candidates.*

Documented in Chapter 8, Section 3.

Conclusion 7: *Metrics for evaluation were an effective representation of performance against requirements.*

Documented in Chapter 8, Section 2.

Conclusion 8: *The C4.5 classifier was effective in ranking of candidates.*

Documented in Chapter 8, Section 5.

A key driver of the study was to make the Process usable and scalable. In Spiral 6 the ClassifierSuite added an improved approach to selecting criteria and understanding the impact of decisions. In the original (manual) approach, the collation of values against criteria limited search functionality and restricted the number of items that were considered. Through the use of the classifier, the full repository could be assessed and the enhanced data representation took the recall and relevance above that of database queries. The query or criteria set is tuned to create a shortlist through iteration, which had typically been 4-6 sets in case studies. The ClassifierSuite allows the consideration of all possible sets in a graphical format with tool support. This may identify sets that would not have been considered along a heuristic path of exploration. The suite also provides a mechanism to justify the choices of sets of criteria.

Conclusion 9: *The ClassifierSuite is an effective decision support tool.*

Documented in Chapter 9.

Research Element 4

Evaluation of the effectiveness of the template, process and strategies via case studies (RE4)

The final Spiral addressed **RE4** to evaluate the Process through application to a case study which comes with the evaluation of each Spiral to allow a complete evaluation in terms of overall project goals. The outcomes of the Spirals performed well against all six goals and in terms of peer review.

Conclusion 10: *The outcomes of the tasks addressed in the Spirals achieved the goals set for the project.*

Documented in Chapter 10, and in the evaluations in Chapters 4 to 9.

11.1.2 Alternative Explanations

This investigation was carried out by the researcher, including all experiments and case studies. In some cases a scenario was revisited to assess the value of a new or different strategy. These strategies were expected to improve outcomes (e.g. T5 improved on T3) and the results supported this for automated over manual. It is also possible that the increased familiarity with the scenarios resulted in improved results. In the case of the transformations, the scripts used to generate the shortlists were identical and thus any biases were restricted to the calculation of relevance and recall. The researcher identified the relevant projects on the repository as a whole, which allowed an objective view of which shortlists had best recall and relevance.

Familiarity may have affected the comparison between manual and automated approaches to shortlisting, including the use of the ClassifierSuite. Although not specifically or intentionally dealt with, it is less likely to have caused problems as the experiments and case studies were often separated by a long time interval. This would have diminished the effect of researcher familiarity with the scenarios. In addition, the 'email client' scenario used for most of the investigation had to be replaced. This was due to low quality in the candidates on the shortlist, where few were able to be installed and executed. A new case study, the 'XML editor', was developed, resulting in completely new shortlists and removing any familiarity bias.

In the testing and evaluation, a range of metrics were calculated. In the case studies these metrics effectively differentiated between functionally suitable and unsuitable software. An alternative explanation is that a subset of the metrics would have been enough to make a decision. The complete set of metrics are `FFIT`, `FEXS`, `AEFT`, `TFIT`, `TRES`, `CX_P`, `CX_R`, `CX_S` and `CX_U` (see Table 8.3).

The metrics were designed to be comprehensive - to give all information the developer may need. It is possible that one or a subset of the metrics are sufficient for the evaluation. A likely candidate would be metric `TRES`, which to some extent combines functional fit, testing fit and implies that the adaptation effort is not insurmountable. It does not incorporate `FEXS`, but would give some clue to the context (CX) metrics. A larger number of case studies could be analysed to see if there is a key metric(s). If so, the metric set could be reduced. At this point, more metrics are preferred as they allow for greater visibility of the performance of the evaluation.

11.1.3 Limitations of the Study

The limitations of the study relate to: social context; the dataset used; and, internal-only usage and evaluation of the Process, products and procedures. It was decided to assume a generic global context as a delimitation of the study. This allowed the focus to be on the selection task from the point of having the requirements for the desired component available. The social and project context is a point of variation, however, the provision of a structured, repeatable process is a step towards insulating selection from these contextual aspects.

As mentioned in Chapter 6 the freshmeat repository is populated by projects, the majority of them standalone applications. This research has aimed to work with real world data whenever possible. The manual case study (Spiral 2) had accessed Tucows, Component Source, Flashline and SourceForge and through this developed an awareness of the richness of metadata they held. Unfortunately, the leading component repository, Component Source, could not approve access direct to full metadata for their listings. Other component repositories were considered, but none had significant numbers of components registered. This led to a trade-off between the value of real world application data over a synthesised dataset with specifically developed components to evaluate. The decision was made to work with a larger dataset which was openly available at freshmeat

(33,262 projects at the time) and to focus on shortlisting. Much of the work in Spirals 3 and 4 required metadata describing potential candidates for shortlisting. As a result the evaluation and testing was made more general to deal with applications rather than components in the case studies. The Process and evaluation has still been targeted at components. Subsequent studies would approach additional repositories and to access metadata for components.

Another limitation of the study is that the automation is applied to a single repository (Spiral 4 onward). Although one repository has been used, the strategies and tools are not bound to the specific repository format. The data is converted to CdCE format by the FM2CdCE filter. This is an exemplar for other repositories and would simplify bringing in future repositories. Unfortunately, after much effort the conversion of the SourceForge repository was found to be too complex due to the internal database structure. A flat file export simplifies the transformation process and should be sought out for future repositories.

The remaining limitations of the project are in the range of comparisons made. It would have been informative to compare the responses of independent developers in using the Process and reflect on their experiences. This would have tested documentation and ease of use, as well as moved beyond the researchers familiarity with the Process. It would also have been of value to have external evaluation of the Process, products and case studies. These were peer-reviewed, however a more specific and detailed review may have found areas to improve. Time did not allow this and it is a clear priority for future work.

11.1.4 Impact of Study

This investigation opens up a new direction in strategies for component selection. On a low level, the specific tools and techniques are available for component selection and other selection tasks. These include:

- Component specification template
- Support for context
- Test generation
- XML/XSLT support for documentation

- Use of the classifier for selection
- Training data generation
- Decision-support through ClassifierSuite.

At the process and specification level, they can be utilised in the application of the Process as described in this thesis, or they may be used to explore alternative strategies as a framework. This is where the research conducted could have benefited from equivalent specification, processes and case studies, had they existed. Future studies may find this work useful as a benchmark.

Finally, the approach to the investigation which used SDM as the central research methodology has been instructive as a method for carrying out objective setting, development of strategies, implementation and evaluation. More than a software development exercise, this work has shown a systematic approach to the entire study through the use of the SDM. This is particularly suited to a PhD as the Spirals include checkpoints for evaluation which provide opportunities to write up cohesive sections of work for peer/review and publication.

11.2 Implications

The investigation is considered to have a range of contributions at a variety of levels. Each has potential to influence professional practice, as was the intent of the project. In all cases aspects have been identified to be taken further through future studies. This would aim to improve understanding and provide a wider comparison of techniques. Future work could extend the component specification, the selection process, the use of the classifier, data repository, testing and evaluation, ClassifierSuite, pattern and SDM. Three of the identified contributions are considered to have strong potential in extending scholarly understanding in the field: data representation; testing and ClassifierSuite. In addition, data representation, pattern and use of SDM may influence theory in the area.

The impetus for this work was to support application developers in their discovery and evaluation of components when developing COTS-intensive systems. Although many processes have been published, there is evidence that the uptake is low and that most developers use informal or ad-hoc approaches when selecting software (Li et al, 2005). With the challenge of adoption in mind, the goals for GQM throughout the project

were **quality**, **usability**, **intelligence**, **innovation**, **dynamics** and **reuse**. These align closely to the Win conditions and are all concerned with the professional practice of the stakeholders.

Key stakeholders for quality were the application developers and quality assurance personnel. Win conditions relating to quality included validity of results and documentation of the decision process. Kotonya and Hutchinson (2005) points out the importance of documenting the selection process.

Each of these avenues of influence is now discussed in more detail in terms of the contributions (C1-C9) which were described in Section 1.8. These are linked to the goals of the study in the following discussion:

C1: Component specification template

There is support for the **usability** of the template as the required fields are synthesised from existing templates in the literature and those used in software repositories. The specification has potential to influence specification standards, as a whole, in part or in the handling of data representation. The component specification is implemented as an object in the developed code to isolate changes to attributes, allowing for **dynamic** update and substitution of the template within the Process. The specification itself was not highly **innovative**, however the content includes both functional and *non-functional* attributes and *context* information, suited to both discovery and evaluation. It was a key factor enabling the use of artificial **intelligence** throughout the Process.

C2: Repeatable, semi-automated process for component selection

A key to **quality** in selection is a repeatable, well documented process. Developers and quality assurance can work with the CdCE and the automation within it to remove much of the subjective work, giving supporting evidence for any decision that is made. Inputs and outputs of the Process are held in XML files which can be audited if required. The Process, and its tools, support the quality of selection through this documentation of processes and decisions.

An intuitive, flexible process is more likely to be adopted than one that is difficult to understand or adapt to local procedures (**usability**). The CdCE Process formalises the common evaluation steps and may contribute to a quality process that developers will

actually use. In a manual selection process, it can help transition from an ad-hoc to a structured approach. Using the automation tools takes on some of that manual effort and gives the ability to assess larger repositories and more complex criteria.

The support for flexibility in the implementation of the Process is reflected in the low coupling between steps in the process, and in the well-defined 'interfaces' between Steps (**dynamics**).This was demonstrated as the Process evolved and the implementation changed across the course of the investigation. The Process itself was **innovative** in its generality and in its implications for the separation of functional (testing) and non-functional (shortlisting) parts of the evaluation.

C3: Support for context

Context is clearly a key issue in **quality** for component-based systems, particularly in testing (Weyuker, 1998). Support for context was targeted throughout the investigation. For example, the non-functional criteria provide context information for the shortlisting and a mechanism was developed for context-based tests. The context support is flexible in that it can encode the context information for different selection tasks (**dynamics**). The context information and related results and artefacts are **reused** throughout the Process. The context-based tests and their specifications have potential for reuse beyond the selection process.

C4: Use of classifiers for selection

An issue identified in the literature was the limitations of aggregated assessment approaches (Ncube and Dean, 2002). The application of computational **intelligence** for assessment is an **innovation** with a purpose. The requirement was that the new strategy needed to avoid: the loss of information of aggregation; calculations based on incompatible criteria types; and the widely used AHP's assumption of independent criteria. The C4.5 classifier takes each criterion on its own merits, generating a decision tree that filters through the criteria. It is capable of identifying interplay (dependencies and incompatibilities) which are obscured by aggregation approaches. AHP and WSM are still widely applied and the classifier is a contribution to the practice in that it provides analysis and justification without aggregation and assumptions of independence. The selection calculation impacts on the **quality** of the results and this study provides another option

for application developers.

In terms of **dynamics**, the choice of classifier is an option on the call to Weka in the scripts, which allows easy substitution of a different machine learning tool within the Weka Suite. If another machine learning tool is used, the scripts can be updated easily, with the main additional change the matching of file formats for input and output.

C5: Data representation enhancements

Using the shortlisting approach allows the user to use more advanced knowledge representation than a standard database query. This is encapsulated within the choice of transformation (T1-T5) and has been shown to provide greater recall and relevance in the given case studies. For professional practice, adding value without increasing the complexity of the user interface helps maintain **usability**.

When looking at the data representation it is possible to alter attribute types and add new transformations (**dynamics**). Modifying attribute types and handling requires changes to the Java classes. These have been implemented with object-orientation and polymorphism which provides good structural guidance for change. Adding transformations requires insertion of newly valid options (e.g. Transformation 6 = T6) and handling code in the class for each attribute type. To best see the impact of changes, it may be advisable to modify the handling of one attribute type in each new transformation and compare results.

The aim of the data representation Spiral (Spiral 4) was to improve the results delivered via computational **intelligence** in Spiral 3. Although ontologies have become popular, the integration of the knowledge-based approach into the Process and tools is uncommon (**innovation**).

C6: Testing and evaluation approach

To support **quality** and compatibility, testing needs to be done in context and the tests be comparable. The CdCE Process enforces the use of the same tests on all candidates and advocates that the test environment be as similar to the target environment as possible. The Process maintains a direct connection between behavioural requirements, generated abstract tests and adapted tests, giving confidence that the right things have been tested and that the testing and evaluation has been consistent across all candidates.

The Process provides a straight forward approach to test generation. Although choosing Z notation might be considered a potential barrier to **usability**, the level of complexity required in the behavioural specification is low - at the level of interface specification. With this information, the specification is parsed to find required specific information for the tests to be generated. So, using a simply defined behavioural specification, the developer can generate a set of abstract tests.

A modestly **intelligent** approach to test generation has been used, **reusing** methods from the testing literature.

C7: Classifier suite for decision support

The ClassifierSuite is aimed squarely at improved **usability** of the CdCE Process - specifically in providing a greater understanding of the results of running sets of criteria through the classifier. Where the use of classifiers could potentially put forward a difficult to understand list of numbers, the Suite gives representation for the underlying relationships between the numbers. It also connects to the shortlists, providing the user with a drill-down facility from criteria to shortlist. The ClassifierSuite provides decision support via an innovative tool which improves usability of the shortlisting strategies and tools.

While the graph structure (lattice) does exist in the literature, this use of the structure is **innovative**. The approach allows the user to gain greater benefit from the **intelligence** and knowledge techniques used for shortlisting.

C8: Pattern for software selection

In support of **reuse**, the process can be generalised into a template for selection which has an instantiation in the CdCE Process. This involves the implementation of each of the steps and relevant tools. However, many aspects of the Process can be implemented differently to suit local requirements. This may be at an organisation, team or project level. For example, if Object Constraint Language[1] (OCL) is already being used on a particular project, an adjustment can be to use OCL in place of Z notation throughout. This would require a new test generator and parser, but most of the Process would be unaffected. The flexibility of a pattern for selection has potential in professional practice and makes the outcomes of this research more widely applicable.

[1] http://www.omg.org/spec/OCL/2.2/

Other data repositories are also easily substituted. The ontology can be extended or substituted, the distance matrix can be altered or replaced and/or the repository itself. Similarly, the testing and evaluation implementation can be substituted with different specification language; test generation, test metrics, a test harness and evaluation method. Additional tools, such as the ClassifierSuite, can be included or excluded. The ClassifierSuite is an example of modifying the Process, as it was inserted into Step 2 and attached to the existing output files. This exemplifies the **dynamic** nature of the Process as a template for selection. Indeed, the selection task may be changed to assessing items other than software and still be applicable.

C9: SDM as a research methodology

An **innovation** the researcher considers highly effective is the use of the SDM for the entire investigation. This has been discussed under Section 11.1.4, Impact of the Study, and is reinforced here as a novel aspect of the work which may affect professional practice in how research is undertaken. In terms of validity of results, the investigation has utilised the stakeholder Win conditions, the Spiral evaluations and peer review to continually validate results. The SDM has facilitated a **quality** process for doctoral research.

11.2.1 Implications for Professional Practice

The CdCE Process as a whole indicates a way forward for organisation seeking greater quality, transparency and traceability in the selection of components. As a pattern, the Process can be tailored to suit the existing procedures and tools used on-site.

Although not the focus of this investigation, brokers may find value in the specification template, and in the support for the shortlisting activity: use of classifiers, leverage of knowledge representation and visualisation of selection criteria.

The ClassifierSuite has potential for extension and broader application as a mechanism for understanding the impact of attributes on decisions. The value for supporting selection has been shown. It is possible to consider multidimensional datasets and use the tool to help understand the interactions of complex sets of variables (parameters, attributes).

Another contribution of the investigation is the use of the SDM for managing the entire project. The application of SDM in doctoral studies is not novel, however, it would

typically be focussed on the software development, separate to the wider investigation. SDM has proven suitable across this study, supporting the reflection and self-evaluation required for such an investigation. Thus, along with industry applications for the Process, template and ClassifierSuite, the use of the SDM has potential implications for academic professional practice.

11.2.2 Implications for Scholarly Understanding

The work of Spirals 3 to 5 are considered to have the most potential for influencing scholarly understanding. In these parts of the investigation, case studies and quasi-experiments were used to explore potential solutions and compare alternatives.

Spiral 3 explored alternatives for applying artificial intelligence to the shortlisting task. A broad survey of traditional and new techniques highlighted those most suited to this problem. The exploration of these techniques and any unforeseen issues are discussed in Chapter 6. This information and analysis may be of value to other researchers as a guide to additional options which may be suitable to investigate.

The exploration of data representation in Spiral 4 provided valuable information on how to extract additional information from a dataset. Many small experiments were carried out to compare the impact of each of the modifications. This provided an understanding of the individual and cumulative effects of the transformations and missing data handling.

The third area having particular relevance to scholarly understanding is Spiral 6. The ClassifierSuite formalised a manual approach to understanding criteria sets and shortlists. In exploring the usefulness of the ClassifierSuite, properties and guidelines were developed. As a novel approach to this analysis, the ClassifierSuite and the reflections on it are contributions to scholarly understanding.

11.2.3 Implications for Future Research Studies

There are two main implications for future research studies: topics to extend and material to build on. The CdCE Process and associated tools have been developed for reuse, modification and extension. As such, they are available to be utilised in future research studies. Beyond tool reuse, the case studies and scenarios may also be used as benchmarks for comparison in future research. The researcher had looked (unsuccessfully) for existing

benchmarks to allow comparison of the work with other approaches.

Another implication is guidance for approaches to pursue in future research. From this investigation and others (Ncube and Dean, 2002), there are many alternatives to WSM and AHP (aggregation) based approaches. New work applying aggregation should be able to justify the choice, given arguments against it and alternatives that are in the literature.

Techniques including the use of XML/XSLT for process documentation, the C4.5 classifier and other machine learning techniques, distance calculations from ontologies, abstract test generation and working with the freshmeat repository show promise and can be recommended for further investigation.

11.3 Recommendations

There are many directions this work can be continued, and recommendations that can be made for industry and academia.

11.3.1 Recommendations for Further Research

This investigation has highlighted many potential areas for future research. In most cases, these relate to the identified limitations of the project. These areas include:

- A comparative study including CdCE and other processes
- Trial the Process in a software development environment
- External evaluation of the Process and tools
- Additional repositories
- Access to a component repository
- Other applications for ClassifierSuite
- Improve decision support through visualisation
- Test generation and the adaptation of tests.

In addition, further information could be elicited from the stakeholders about their requirements. This could be through a variety of means, including surveys, focus groups or case studies to find out more about current processes, requirements and allow for feedback on the CdCE Process. The experiments and cases in this study are laboratory-based,

and it would be beneficial to trial the Process in a commercial/organisational setting. This would highlight any issues with Process suitability when socio-organisational factors play a significant role, and any adjustments that may be required.

Of particular interest are two areas, test generation and to work with a dataset specific to components. Unfortunately, it wasn't possible to access real data from a component repository at the time the work was carried out. The freshmeat dataset had strengths in the type of metadata available and the number of entries - making it possible to work through the shortlisting process. However, the intent of this work was to select components. If access was available, a new case study would require the transformation of the repository data to CdCE format. The selection process would be as described in Chapters 5 to 9, with possible differences around the formal specification and the test generation, adaptation and execution. Once the metrics were calculated, the existing approach would continue.

A related extension of the work would be to further explore the generation and adaptation of the tests. One direction would be to go further with the adaptation models, with the aim of automating that process. In addition, the tests could be executed via a harness. In that case, some effort would be required to transform the abstract tests to suit both the harness and the component adaptations. Once the transformations are defined, an automated, repeatable process could take place to convert the test suite.

11.3.2 Recommendations for Professional Practice

The stakeholders identified for this investigation are the roles relating to these tasks in professional practice. As a result, the outcomes are now considered in terms of the stakeholders.

The key stakeholders throughout this investigation have been the application developers. The recommendation for them is to assess their current practices in selection of third party software. If selection of externally developed components occurs, they may look at adapting the CdCE Process to their environment. This could involve adjusting for documentation standards, specification languages, relevant repositories and integrating with the development methodology used for the overall system. During adoption, the new techniques should be staged in, as recommended by Rifkin (2003). Reviews of performance and applicability after each selection task would aid in tuning the new

processes to match the organisational context.

Quality assurance personnel may not be aware of the practices used by developers when selecting third party components. The Win conditions from the Spirals give an indication of what parts of the CdCE Process the researcher believes these stakeholders would be interested in. Overall quality can be considered as the minima of quality in each part of the process - the weakest link affects overall quality. To avoid component-intensive development from bringing unacceptable risk into the development, quality assurance needs to encourage better practices. The Process provides documentation and traceabillity, as well as formalising a task that is often informal.

For brokers, some of the issues encountered in the use of repositories may need to be addressed. Provision of access to metadata for the repository affected the choice of dataset for the research, and reduced options for comparative studies. For industry, the effects are more serious and open access is encouraged. At the data level, the CdCE template attributes may indicate additional information that could be added to a broker's data model. This research may also impact repository search facilities as the classifier approach, and/or other aspects used in comparisons, may be able to enhance their search.

Component developers were not the direct targets for this work, but would have an interest in how their components are evaluated. Key recommendations from this study are: completeness of metadata; documentation of interfaces and functionality; regularity of updates and ease of installation. If certification of components becomes more common, this would be included in metadata and may become a discriminator for the selection of software.

The final stakeholder is academia, in respect of how the study was carried out. As has been mentioned, the SDM has been used to structure the entire study - beyond the basic development of software. In this way, a software engineering or computer science study can draw on alternative methodologies to the more commonly used R&D. As computing is typically used to 'solve problems', this approach can put the problem at the centre of the study, rather than the solution. Within the Spirals of the SDM, a mixed methods approach has been used for evaluating strategies and their implementation. This approach is novel, appropriate and has a solid theoretical basis. It is recommended to be considered for future studies.

11.4 Conclusion

This thesis has documented the systematic exploration of strategies to support software component selection. The study was based around four research elements which were designed to provide the foundation for the development and implementation of strategies and their evaluation. Section 11.1.1 lists the ten key conclusions drawn from the Research Element findings.

The implications of the study span the professional practice of the stakeholders across the goals of quality, usability, intelligence, innovation, dynamics and reuse. Spirals 3-6 focussed on strategies and hence provide the greatest potential impact on scholarly understanding. The outcomes and lessons from each of the Spirals has been published as the investigation progressed (or after review and reflection).

The work raises new questions and avenues for future research as recommended in Section 11.3.1. Recommendations are also given for the stakeholders: application developers; quality assurance officers; brokers; component developers and academia.

At this point the project has resulted in a series of contributions, across the seven Spirals. These address issues and gaps identified in the critique in Chapter 2. Spiral 1 provided the component specification template. The lack of a standard specification causes problems for storage, discovery, selection and for automation. The CdCE template, swvML, is developed from existing standards, repository templates and the literature, along with the requirements for this study. It has been enhanced as the project progressed to provide the information required in later Spirals.

The contributions from Spiral 2 are on two levels. Directly, a repeatable, semi-automated process for component selection has been defined. The CdCE Process addresses the issue raised by Ruhe (2002) regarding the lack of scope for automation in existing processes. At a reuse level, Spiral 2 also contributed a pattern for a selection process, including filtering to a shortlist, functional evaluation and ranking. This provides flexibility for the Process to be implemented in different ways and for different selection tasks.

Early (and some recent) literature on evaluation applied aggregation techniques (e.g. WSM and AHP) for moving from results against selection criteria to a single result. This has been criticised in the literature (Ncube and Dean, 2002) and was targeted in the exploration of strategies in Spiral 3. The approach taken is the use of the ideal

specification to generate training data for a predictive model using C4.5. This approach centres the shortlisting on requirements as represented in the ideal specification. It is a novel approach with the potential for application beyond C4.5 and component selection.

The classifier approach solved some issues and opened opportunities to make better use of knowledge in the component attributes. Enhancements to the representation of data and the methods of comparison were shown to enhance relevance and recall in the resultant shortlists. The handling of missing data was also improved and made consistent. The contribution of Spiral 4 is in the strategies for data representation and in the evidence shown to support the premise that recall, relevance and consistency have been improved through these strategies.

There were two strategic directions that were in the early intent of the project: application of artificial intelligence and testing as part of the selection process. The testing and evaluation in Spiral 5 was built on a behavioural specification in Z notation. The contribution is not in the automated test generation itself - it is in the use of abstract test cases and the integration of the testing and evaluation. Generating abstract test cases ensures consistent tests can be used across all of the candidates, as the selection process is matching to a single specification. The evaluation uses automatic test generation from the Z specification. The resulting abstract tests are then adapted to each of the candidates.

Basic tests exercise the functionality as specified, then subsets of the tests are run to focus on aspects of the functionality as required by the user context. These context-based tests are in four areas: performance, reliability, stress and usage. A set of corresponding metrics has been defined to capture these results. These are collated along with five metrics for functional fit, functional excess, adaptation effort, testing fit and test result. The classifier approach is reused for the ranking of candidates. Information across all of the steps in the Process is provided to the application developer to assist in the final decision and its documentation.

One of the benefits of automated and structured approaches is in the repeatability of the process. The CdCE Process creates reusable artefacts which can be used when revisiting the selection process. System evolution may force the replacement of components, or components may become unavailable. Making it easy to ensure that the selection matches the initial task reduces rework and provides consistency. Quality assurance activities can

make use of these artefacts in any audits of the selection.

The use of automation in shortlisting led to an increase in the number of shortlists considered in the case studies. Extending this to an exhaustive exploration of all shortlists created a new issue - the comprehension of all the shortlist data. The ClassifierSuite was developed in response to this emergent need. The contribution of a novel and effective decision support tool came in Spiral 6. Case studies from previous Spirals were revisited and the ClassifierSuite tool uncovered new and unexpected shortlists. The ClassifierSuite provided understanding and confidence in the choice of selection criteria and enhanced the CdCE Process as a whole.

Spiral 7 focussed on Research Element 4 - evaluation of the products of the investigation. The final key contribution is in the structure of the evaluation and how the research was conducted. The investigation has been structured around the use of the SDM. This also structures the review and evaluation for each Spiral. Goals were defined for the overall project, then interpreted in the context of each Spiral. These contextual goals were used for the GQM evaluation at the end of each Spiral, along with the Win conditions for each of the stakeholders. Thus the final contribution is in the manner in which the research was conducted, showing that the SDM is appropriate not only to the development of software, but also to entire research projects.

In conclusion, the investigation was concerned with the development of strategies to support software component selection. The project has addressed the gaps identified in the literature and practice, and made corresponding contributions through the strategic approaches taken towards these gaps. There are implications for professional practice, scholarly understanding and to future research in the area. It is hoped that this work helps to progress the uptake of CBSE and the ability for application developers to employ quality in their selection of third party software components.

Bibliography

Abbass H A and Sarker R (2001) "Simultaneous Evolution of Architectures and Connection Weights in ANNs", *in* "The Artificial Neural Networks and Expert Systems Conference (ANNES2001)", Dunedin, New Zealand, pp 16–21

Abran A, Bourque P, Dupuis R, Moore J W and Tripp L L (eds) (2004) *Guide to the Software Engineering Body of Knowledge - SWEBOK*, Piscataway, NJ, USA: IEEE Press, 2004 version ed

Albert C and Brownsword L (2002) *Evolutionary Process for Integrating COTS-Based Systems (EPIC) Building, Fielding, and Supporting Commercial-off-the-Shelf (COTS) Based Solutions*, Technical Report CMU/SEI-2002, Carnegie-Mellon University

Alexander C (1977) *A Pattern Language: Towns, Buildings, Construction*, Oxford University Press

Alvaro A, de Almeida E S and de Lemos Meira S (2005a) "Software Component Certification: A Survey", *in* "31st EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA'05)",

Alvaro A, Almeida E S D and de Lemos Meira S R (2005b) "Quality Attributes for a Component Quality Model", *in* "In the 10th International Workshop on Component-Oriented Programming (WCOP) in Conjunction with the 19th European Conference on Object Oriented Programming (ECOOP)",

Alvaro A, de Almeida E S and de Lemos Meira S R (2010) "A Software Component Quality Framework", *ACM SIGSOFT Software Engineering Notes*, 35(1)

Alves C and Castro J (2001) "CRE: A systematic method for COTS components selection", *in* "Proc. of the XV Brazilian Symposium of Software Engineering", Brazil

Alves C and Finkelstein A (2002) "Challenges in COTS Decision-Making: A Goal-driven Requirements Engineering Perspective", *in* "Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE 2002)",

Anderson W (1989) "A heuristic for software evaluation and selection", *Software Practice and Experience*, 19(8), pp 707–717

Andreea Vescan A and Grosan C (2008) "A Hybrid Evolutionary Multiobjective Approach for the Component Selection Problem", *in* "Proceedings of the International Conference on Hybrid Artificial Intelligence Systems (HAIS)", pp 164–171

Andreou A and Tziakouris M (2007) "A quality framework for developing and evaluating original software components.", *Information and Software Technology*, 49(02), pp 122–141

Andreou A S, Vogiatzis D G and Papadopoulos G A (2006) "Intelligent Classification and Retrieval of Software Components", *in* "IEEE signature conference on Computer Software and Applications (COMPSAC)",

Aoyama M (1998) "New Age of Software Development: How Component-Based Software Engineering Changes the Way of Software Development", *in* "International Workshop on Component-Based Software Engineering", Kyoto, Japan

Atkinson S (1997) *A Formal Model for Integrated Retrieval from Software Libraries*, Technical Report 97-01, University of Queensland

Bachmann F, Bass L, Buhman C, Comella-Dorda S, Long F, Robert J E, Seacord R C and Wallnau K C (2000) *Volume II: Technical Concepts of Component-Based Software Engineering*, Technical Report CMU/SEI-2000-TR-008, Carnegie Mellon University: Software Engineering Institute

Bader A, Mingins C, Bennett D and Ramakrishan S (2003) "Establishing Trust in COTS Components", *in* "Proceedings of the Second International Conference on COTS-Based Software Systems", ICCBSS '03, London, UK: Springer-Verlag, pp 15–24, `http://dl.acm.org/citation.cfm?id=646853.707759`

Bakås O, Romsdal A and Alfnes E (2007) "Holistic ERP selection methodology", *in* "14th International EurOMA Conference",

Barden R, Stepney S and Cooper D (1994) *Z in Practice*, Prentice-Hall

Basili V (1992) *Software Modelling and Measurement: The Goal/Question/Metric Paradigm*, Technical Report CS-TR-2956, University of Maryland

Basili V and Boehm B (2001) "COTS-Based Systems Top 10 List", *IEEE Computer*, 34(5), pp 91–93

Basili V, Caldeira G and Rombach H D (1994) *The Goal Question Metric Approach*, New York, USA: John Wiley and Sons, `http://wwwagse-old.informatik.uni-kl.de/pubs/repository/basili94b/encyclo.gqm.pdf`

Bass L, Buhman C, Comella-Dorda S, Long F, Robert J E, Seacord R C and Wallnau K C (2000) *Volume I: Market Assessment of Component-Based Software Engineering*, Technical Report, CMU/SEI-2001-TN-007

Bass L, Clements P and Kazman R (1998) *Software Architecture in Practice*, Reading, Massachusetts: Addison Wesley Longman

Batista G and Monard M (2003) "An Analysis of Four Missing Data Treatment Methods for Supervised Learning", *Applied Artificial Intelligence*, (17), pp 519–533

BCS (2001) *Standard for Software Component Testing*, Standard Working Draft 3.4, British Computer Society: Specialist Interest Group in Software Testing (BCS SIGIST)

Benoît G (2002) *Data Mining*, Silver Spring, MD: American Society for Information Science and Technology, pp 265–310

Bergner K, Rausch M, Sihling M and Vilbig A (1999) "Componentware - Methodology and Process", *in* "International Workshop on Component-Based Software Engineering", Los Angeles

Bertolino A (2007) "Software Testing Research: Achievements, Challenges, Dreams", *in* "Future of Software Engineering (FOSE'07)", IEEE Computer Society

Beus-Dukic L (2000) "Non-Functional Requirements for COTS Software Components", *in* "Position Paper. ICSE '2000 Workshop on Continuing Collaborations for Successful COTS Development", ACM Press, pp 4–5

Boehm B (1988) "A Spiral Model of Software Development and Enhancement", *IEEE Computer*, pp 61–72

Boehm B (2006) "A view of 20th and 21st century software engineering", *in* Osterweil L J, Rombach H D and Soffa M L (eds) "ICSE '06: Proceedings of the 28th International Conference on Software Engineering", ACM, pp 12–29

Boehm B, Port D, Yang Y, Bhuta J and Abts C (2003) "Composable Process Elements for Developing COTS-Based Applications", *in* "Proceedings of the 2003 International Symposium on Empirical Software Engineering (ISESE03)", pp 8–17

Bonoma T (1985) "Case Research in Marketing: Opportunities, Problems, and a Process", *Journal of Marketing Research*, XXII, pp 199–208

Bote-Lorenzo M, Dimitriadis Y and Gãmez-Sãnchez E (2004) "Grid Characteristics and Uses: A Grid Definition", *in* Fernãndez Rivera F, Bubak M, Gãmez Tato A and Doallo R (eds) "Grid Computing", Springer Berlin / Heidelberg, volume 2970 of *Lecture Notes in Computer Science*, pp 291–298, `http://dx.doi.org/10.1007/978-3-540-24689-3`$_3 6, 10.1007/978 - 3 - 540 - 24689 - 3 - 36$

Bracciali A, Brogi A and Canal C (2002) "A formal approach to component adaptation", *The Journal of Systems and Software*, 74(2005), pp 45–54

Brereton P and Budgen D (2000) "Component-Based Systems: A Classification of Issues", *IEEE Computer*, pp 54–62

Briand L C (2002) "On the many ways software engineering can benefit from knowledge engineering", *in* "In Proceedings of the 14th international conference on Software engineering and knowledge engineering", ACM, pp 3–6

Briand L C, Labiche Y and Sowka M M (2006) "Automated, Contract-based User Testing of Commercial-Off-The-Shelf Components", *in* "ICSE '06: Proceedings of the 28th International conference on Software Engineering", New York, NY, USA: ACM, pp 92–101

Brogi A, Canal C and Pimentel E (2006) "On the semantics of software adaptation", *Science of Computer Programming*, 61, pp 136–151

Brooks F P (1987) "No Silver Bullet - Essence and Accidents of Software Engineering", *IEEE Computer*, 20(4), pp 10–19

Brou K (2005) "Querying of Open Source Programs Libraries: an Approach Based on a XML Metadata Repository", *in* "IEEE SITIS", pp 100–106

Buglione L and Abran A (2000) "Balanced Scorecards and GQM: What are the differences?", *in* "FESMA-AEMES Software Measrurement Conference",

Burgues X, Estay C, Franch X, Pastor J A and Quer C (2002) "Combined Selection of COTS Components", *in* "International Conference on COTS-Based Software Systems (ICCBSS)", LNCS 2255, Springer, pp 54–64

Burton S (2000) *Automated Testing From Z Specifications*, `http://citeseer.nj.nec.com/burton00automated.html`

Buxton J, Naur P and Randell B (eds) (1968) *Software Engineering Concepts and Techniques: 1968 NATO Conference on Software Engineering*

Carney D and Long F (2000) "What Do You Mean by COTS? Finally, a Useful Answer", *IEEE Software*, 17, pp 83–86, `http://dl.acm.org/citation.cfm?id=624636.626114`

Carrington D, MacColl I, McDonald J, Murray L and Strooper P (1998) *From Object-Z Specifications to ClassBench Test Suites*, Technical Report SVRC Technical Report No 98-22, The University of Queensland

Carvallo J P and Franch X (2006) "Extending the ISO/IEC 9126-1 quality model with non-technical factors for COTS components selection", *in* "WoSQ '06: Proceedings of the 2006 international workshop on Software quality", New York, NY, USA: ACM, pp 9–14

Carvallo J P, Franch X, Grau G and Quer C (2004) "COSTUME: A Method for Building Quality Models for Composite COTS-Based Software Systems", *in* "QSIC", IEEE Computer Society, pp 214–221

Cechich A, Piattini M and Vallecillo A (2003) "Assessing Component-based Systems", *In: Cechich et al (Eds) Component-Based Software Quality,*, LNCS 2693, pp 1–20

Cechich A and Polo M (2002) "Black-Box Evaluation of COTS Components Using Aspects and Metadata", *in* Oivo M and Komi-Sirviö S (eds) "PROFES", Springer, volume 2559 of *Lecture Notes in Computer Science*, pp 494–508

Cechich A, Requile-Romanczuk A, Aguirre J and Luzuriaga J M (2006) "Trends on COTS Component Identification", *in* "Proceedings of the Fifth International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems", Washington, DC, USA: IEEE Computer Society, pp 90–, `http://dl.acm.org/citation.cfm?id=1114286.1114655`

Chang K H, Liao S, Chapman R and Chen C (2000) "Test Scenario Generation based on Formal Specification and Usage Profile", *International Journal of Software Engineering and Knowledge Engineering*, 10(2), pp 185–201

Charette R N (1989) *Software Engineering Risk Analysis and Management*, Multiscience Press

Chau P Y K (1995) "Factors used in the selection of packaged software in small businesses: Views of owners and managers", *Information and Management*, 29(2), pp 71–78, `http://www.sciencedirect.com/science/article/pii/037872069500016P`

Chen Y and Yao Y (2008) "A multiview approach for intelligent data analysis based on data operators", *Information Sciences*, 178(2008), pp 1–20

Choi Y, Lee S, Song H, Park J and Kim S (2008) "Practical S/W Component Quality Evaluation Model", *in* "the 10th IEEE International Conference on Advanced Communication Technology (ICACT)",

Christiansson B and Christiansson M (2004) "An informal COTS Specification Model – Enabling Component Acquisition", *in* "International Workshop on COTS Terminology and Categories: Can We Reach a Consensus?, ICCBSS 2004",

Clark B and Clark B (2007) "Added Sources of Costs in Maintaining COTS-Intensive Systems", *CROSSTALK The Journal of Defense Software Engineering*, (June), pp 4–8

Collet P, Coupaye T, Chang H, Seinturier L and Dufrêne G (2007) *Components and Services: A Marriage of Reason*, Rapport de recherche ISRN I3S/RR–2007-17–FR, Laboratoire d'Informatique de Signaux et Systèmes de Sophia Antipolis

Comella-Dorda S, Dean J, Lewis G, Morris E, Oberndorf P and Harper E (2004) *Process for COTS Software Product Evaluation*, Technical Report CMU/SEI-2003-TR-017, Carnegie-Mellon University, `http://www.sei.cmu.edu/library/abstracts/reports/03tr017.cfm`

Comella-Dorda S, Dean J, Morris E and Oberndorf P P (2002) "A process for COTS software product evaluation", *in* "International Conference on COTS-Based Software Systems (ICCBSS)", LNCS 2255, Springer, pp 86–96

Computer History Museum (2009) *Computer Pioneers and Pioneer Computers [DVD]*

Coplien J, Harrison N and Bjørnvig G (2005) "Organizational Patterns: Building on the Agile Pattern Foundations", *Agile Project Management Report*, 6(6)

Cortellessa V, Crnkovic I, Marinelli F and Potena P (2008) "Experimenting the Automated Selection of COTS Components Based on Cost and System Requirements", *Journal of Universal Computer Science*, 14(8), pp 1128–1255

Councill W T (1999) "Third-Party Testing and Stirrings of the New Software Engineering", *IEEE Software*, 16(6), pp 76–79

Cover R (2011) *Extensible Markup Language (XML)*, `http://xml.coverpages.org/xml.html`

Creswell J W, Tashakkori A, Jensen K and Shapley K D (2003) *Teaching Mixed Methods Research: Practices, dilemmas and challenges.*, Thousand Oaks, CA: Sage

Crnkovic I, Schmidt H, Stafford J and Wallnau K C (eds) (2003) *Workshop on Component-Based Software Engineering: Automated Reasoning and Prediction*, IEEE Press

Crossan F (2003) "Research philosophy: towards an understanding", *Nurse Res*, 11(1), pp 46–55

Cubo J, Salaun G, Camara J, Canal C and Pimental E (2007) "Context-Based Adaptation of Component Behavioural Interfaces", *in* Murphy A and Vitek J (eds) "Coordination", volume LNCS 4467, pp 305–323

Cunningham W (2010a) *Component Definition*, `http://c2.com/cgi/wiki?ComponentDefinition`

Cunningham W (2010b) *Software Design Patterns Index*, `http://c2.com/cgi/wiki?SoftwareDesignPatternsIndex`

Curtis B, Kellner M I and Over J (1992) "Process modeling", *Communications of the ACM*, 35(9), pp 75–90

DACS (2011) *DACS Gold Practices Website: Goal-Question-Metric (GQM) Approach*, `http://goldpractice.thedacs.com/practices/gqm/`

Di Giacomo P (2005) "COTS and Open Source Software Components: Are They Really Different on the Battlefield?", *in* Franch X and Port D (eds) "ICCBSS 2005", LNCS 3412, pp 301–310

Dick J and Faivre A (1993) "Automating the Generation and Sequencing of Test Cases from Model-Based Specifications", *in* "FME '93: Industrial-Strength Formal Methods",

Dijkstra E W (1968) "Letters to the editor: go to statement considered harmful", *Communications of the ACM*, 11(3), pp 859–866

Dijkstra E W (1972) "The humble programmer", *Communications of the ACM*, 15(10), pp 859–866

Ding Y and Napier N (2006) "Measurement Framework for Assessing Risks in Component-Based Software Development", *in* "Proceedings of the 39th Annual Hawaii International Conference on System Sciences - Volume 09", Washington, DC, USA: IEEE Computer Society, `http://dl.acm.org/citation.cfm?id=1109717.1110102`

Dodig-Crnkovic G (2002) "Scientific Methods in Computer Science", *in* "Conference for the Promotion of Research in IT at New Universities and Colleges in Sweden",

Easterbrook S, Singer J, MStorey and Damian D (2007) *Selecting Empirical Methods for Software Engineering Research*, Springer

Esteves J and Porter J (2004) "Using a Multimethod Approach to Research Enterprise Systems Implementations", *Electronic Journal of Business Research Methods*, 2(2), pp 69–81

Fidge C J (2002) "Contextual matching of software library components", *in* Strooper P and Muenchaisri P (eds) "Asia-Pacific Software Engineering Conference", IEEE Computer Society Press, pp 297–306

Frakes W B and Kang K (2005) "Software Reuse Research: Status and Future", *IEEE Trans Software Eng*, 31(7), pp 529–536

Frakes W B and Pole T P (1994) "An Empirical Study of Representation Methods for Reusable Software Components", *IEEE Transactions on Software Engineering*, 20(8), pp 617–630

Frakes W B and Terry C (1996) "Software Reuse: Metrics and Models", *ACM Computing Surveys*, 28(2), pp 415–435

Freedman R S (1991) "Testability of Software Components", *IEEE Transactions on Software Engineering*, 17(6), pp 553 – 564

freshmeat (2007) *Freshmeat software index*, `http://freshmeat.net/`

Friedman-Hill E (2008) *Jess The Rule Engine for the Java Platform*, Technical Report Version 7.1p2, Sandia National Laboratories

Galorath D D and Evans M W (2006) *Software Sizing, Estimation, and Risk Management*, Boston, MA, USA: Auerbach Publications

Gamma E, Helm R, Johnson R and Vlissides J (1995) *Design patterns: elements of reusable object-oriented software*, Addison-Wesley Professional

Ganter B and Wille R (1997) *Applied Lattice Theory: Formal Concept Analysis*, Technical Report http://www.math.tu-dresden.de/ ganter/psfiles/concept.ps

Gao J, Gopinathan D, Mai Q and He J (2006) "A Systematic Regression Testing Method and Tool For Software Components", *in* "IEEE signature conference on Computer Software and Applications (COMPSAC)", pp 244–249

Gao J Z, Gupta K K, Gupta S and Shim S S Y (2002) "On Building Testable Software Components", *in* "Proceedings of the First International Conference on COTS-Based Software Systems", ICCBSS '02, London, UK: Springer-Verlag, pp 108–121, `http://dl.acm.org/citation.cfm?id=646852.707746`

Garcia V C, Lucrédio D, Durão F A, Santos E C R, de Almeida E S, de Mattos Fortes R P and de Lemos Meira S R (2006) "From Specification to Experimentation: A Software Component Search Engine Architecture", *in* et al G (ed) "CBSE 2006", LNCS 4063, pp 82–97

Garlan D, Allan R and Ockerbloom J (2009) "Architectural Mismatch: Why Reuse is Still so Hard", *IEEE Software*, pp 86–89

Garlan D, Allen R and Ockerbloom J (1995) "Architectural Mismatch or Why it's hard to build systems out of existing parts", *in* "Seventeenth International Conference on Software Engineering", pp 17–26

Gill N S (2004) "Few Imporant Considerations For Deriving Interface Complexity Metric For Component-Based Systems", *Software Engineering*, 29(2), pp 1–4

Gill N S and Grover P S (2004) "Few Imporant Considerations For Deriving Interface Complexity Metric For Component-Based Systems", *Software Engineering*, 29(2), pp 1–4

Glaser B G and Strauss A (1967) *Discovery of Grounded Theory. Strategies for Qualitative Research*, Sociology Press

Gnatz M, Marschall F, Popp G, Rausch A and Schwerin W (2002) "Common Template for Software Development Process Patterns", *in* "1st Workshop on Software Development Process Patterns", `www.forsoft.de/zen/sdpp02/authors/template.pdf`

Grundy J C (1999) "Aspect-Oriented Requirements Engineering for Component-Based Software Systems", *in* "Proceedings of the 4th IEEE International Symposium on Requirements Engineering", Washington, DC, USA: IEEE Computer Society, pp 84–91, `http://dl.acm.org/citation.cfm?id=647646.731259`

Grzymala-Busse J and Hu M (2001) "A Comparison of Several Approaches to Missing Attribute Values in Data Mining", *RSCTC 2000, LNAI 2005*, pp 378–385

Gui G and Scott P D (2006) "Ranking reusability of software components using coupling metrics", *The Journal of Systems and Software*

Gupta A, Conradi R, Shull F, Cruzes D, Ackermann C, Rønneberg H and Landre E (2008) "Experience Report on the Effect of Software Development Characteristics on Change Distribution", *in* Jedlitschka A and Salo O (eds) "Conf. on Product Focused Software Process Improvement (Profes 2008)", LNCS 5089, pp 158–173

Hall H, Frank E, Holmes G, Pfahringer B, Reutemann P and Witten I (2009) "The WEKA Data Mining Software: An Update", *SIGKDD Explorations*, 11(1), pp 10–18

Hall P A V and Hierons R (1991) *Formal Methods and Testing*, Technical Report 91/16, Computing Department, Open University

Hamlet D (2007) "Software component composition: a sub-domain testing-theory foundation", *J Software Testing and Verification Research*

Harker P T (1987) "Incomplete pairwise comparisons in the analytical hierarchy process", *Mathematical Modelling*, 9, pp 837–848

Harman M, Mansouri S A and Zhang Y (2009) *Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications*, Technical Report TR-09-03, Department of Computer Science, Kingís College London

Harrold M J, Orso A, Rosenblum D, Rothermel G, Soffa M L and Do H (2001) *Using Component Metadata to Support the Regression Testing of Component-Based Software*, Technical Report GIT-CC-01-38, College of Computing, Georgia Institute of Technology

Hartman R S (1969) *The Structure of Value*, Knoxville, Tennessee: Robert S Hartman Institute of Applied Axiology

Hasan H (2003) "Information Systems Development as a Research Method", *Australasian Journal of Information Systems*, 11(1), pp 4–13

Helke S, Neustupny T and Santen T (1997) "Automating test case generation from Z specifications with Isabelle", *in* "Proccedings of Z Users Meeting '97", `http://citeseer.nj.nec.com/article/helke97automating.html`

Hemer D (2003) "Specification matching of state-based modular components", *in* "Software Engineering Conference, 2003. Tenth Asia-Pacific", pp 446–455

Holz H and Melnik G (2004) "1-6 Research on Learning Software Organizations – Past, Present, and Future", *Lecture Notes in Computer Science*, 3096, pp 1–6

Horcher H M and Mikk E (1996) *Test Automation using Z Specifications*, Bremen, Germany: Shaker Verlag

Hunter J (2003) *A survey of metadata research for organizing the web*, `http://findarticles.com/p/articles/mi`$_{\mathrm{m}}$`1387/is`$_{25}$`2/ai`$_{1}$`12542835/`

IEEE (1990) *IEEE Standard Glossary of Software Engineering Terminology*, Technical Report

InternationalStandardOrganization (June, 2001) "Information Technology – Product Quality - Part1: Quality Model", International Standard ISO/IEC 9126

Ivers J and Moreno G A (2008) "PACC starter kit: developing software with predictable behavior", *in* "ICSE Companion '08: Companion of the 30th International Conference on Software Engineering", New York, NY, USA: ACM, pp 949–950

Jobs S (2006) "Steve Jobs Keynote: 86 million lines of source code that was ported to run on an entirely new architecture with zero hiccups", *in* "WWDC 2006", http://www.engadget.com/2006/08/07/live-from-wwdc-2006-steve-jobs-keynote

Johnson R and Onwuegbizue A (2004) "Mixed methods research: a research paradigm whose time has come", *Educational Researcher*, 33(7), pp 14–26

Karolak D W (1996) *Software Risk Management*, IEEE Computer Society Press (Los Alamitos, Calif.)

Kaur V and Goel S (2011) "Facets of Software Component Repository", *International Journal on Computer Science and Engineering*, 3(6), pp 2473–2476

Kitchenham B A, Pfleeger S L, Pickard L M, Jones P W, Hoaglin D C, Emam K E and Rosenberg J (2002) "Preliminary Guidelines for Empirical Research in Software Engineering", *IEEE Transactions on Software Engineering*, 28, pp 721–734

Klein M and Kazman R (1999) *Attribute-Based Architectural Styles*, Technical Report Technical Report CMU/SEI-99-TR-022, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Knox K (2004) "A researcher's dilemma – philosophical and methodological pluralism", *Electronic Journal of Business Research Methods*, 2(2), pp 119–128

Kontio J (1995) *OTSO: A Systematic Process for Reusable Component Selection*, Technical Report CS-TR-3478, University of Maryland

Korel B (1999) "Black-box Understanding of COTS Components", *in* "International Workshop on Program Understanding", Pittsburgh, Pennsylvania, pp 226–233

Kotonya G and Hutchinson J (2004) "Viewpoints for Specifying Component-Based Systems", *in* Crnkovic I e a (ed) "CBSE 2004", LNCS 3054, pp 114–121

Kotonya G and Hutchinson J (2005) "Analysing the Impact of Change in COTS-Based Systems", *in* "ICCBSS", pp 212–222

Kotonya G, Onyino W, Hutchinson J and Sawyer P (2001) *Component Architecture Description Language (CADL)*, Technical Report CSEG/57/2001, Computing Department, Lancaster University

Kuhn T S (1996) *The Structure of Scientific Revolutions*, The University of Chicago Press, 3 ed

Kunda D (2003) "STACE: Social Technical Approach to COTS Software Evaluation", *in* Cechich A, Piattini M and Vallecillo A (eds) "Component-Based Software Quality", Springer, volume 2693 of *Lecture Notes in Computer Science*, pp 64–84

Kunda D and Brooks L (1999) "Applying Social-technical Approach for COTS selection", *in* Brooks L and Kimble C (eds) "UKAIS 99", McGraw-Hill

Kunda D and Brooks L (2000) "Accessing Success of a Social-Technical Method for COTS Software Selection: A Survey Approach", *in* "ICEIS", pp 294–298

Kuo Y H, Hsu J P and Horng M F (1999) "Neuro-fuzzy Based Search Robot for Software Components", *International Journal on Artificial Intelligence Tools*, 8(2)

Lampson B (2004) *Computer Systems: Theory, Technology, and Applications, 2004, pp 137-146.*, Springer, chapter Software Components: Only the Giants Survive

Larman C and Basili V (2003) "Iterative and incremental developments. a brief history", *Computer*, 36(6), pp 47–56

Larsson M and Crnkovic I (2001) "Configuration Management for Component-based Systems", *in* "Software Configuration Management - SCM 10, ICSE '01: Proceedings of the 23rd International Conference on Software Engineering",

Lázaro M and Marcos E (2005) "Research in Software Engineering: Paradigms and Methods", *in* "CAiSE Workshops (2)", pp 517–522

Lee J and Ware B (2002) *Open Source Development with LAMP: Using Linux, Apache, MySQL, Perl, and PHP*, Addison-Wesley Professional

Lee J H, Kim M H and Lee Y J (1993) "Information Retrieval Based on Conceptual Distance in IS-A Hierarchies", *Journal of Documentation*, 49(2), pp 113 –136

Leedy P D (2002) *Practical Research: Planning and Design*, Check this one - may have used a paper

Legeard B, Peureaux F and Utting M (2002) "Automated Boundary Testing from Z and B", *in* "FME '02: 11th Conference on Formal Methods",

Lehman M M (1979-1980) "On understanding laws, evolution, and conservation in the large-program life cycle", *Journal of Systems and Software*, 1, pp 213–221, `http://www.sciencedirect.com/science/article/pii/0164121279900220`

Li J, Conradi R, Slyngstad O P N, Bunse C, Khan U, Torchiano M and Morisio M (2005) "Validation of New Theses on Off-the-Shelf Component Based Development", *in* "IEEE METRICS", p 26

Li J, Conradi R, Slyngstad O P N, Bunse C, Torchiano M and Morisio M (2006) "An empirical study on decision making in off-the-shelf component-based development", *in* "ICSE '06: Proceedings of the 28th International Conference on Software Engineering", New York, NY, USA: ACM, pp 897–900

Lill S, Olsen N and Loe K (2005) *The CoExSel Tool*, Technical Report, Norwegian University of Science and Technology

Liskov B and Zilles S (1974) "Programming with abstract data types", *SIGPLAN Not*, 9, pp 50–59, `http://doi.acm.org/10.1145/942572.807045`

Lootsma F (1999) *Multi-Criteria Decision Analysis via Ratio and Difference Judgement*, Dordrecht, The Netherlands: Kluwer Academic Press

Lovins J B (1968) "Development of a stemming algorithm", *Mechanical Translation and Computational Linguistics*, 11, pp 22–31

Lucredio D, Prado A F d and Almeida E S d (2004) "A Survey on Software Components Search and Retrieval", *in* "Proceedings of the 30th EUROMICRO Conference", Washington, DC, USA: IEEE Computer Society, pp 152–159, http://dl.acm.org/citation.cfm?id=1018420.1019676

Luhn H P (1960) "Keyword-in-context index for technical literature", *American Documentation*, 11(4), pp 288–295

Maiden N, Croce V, Kim H, Sajeva G and Topuzidou S (2003) "SCARLET: Integrated Process and Tool Support for Selecting Software Components", *in* Cechich A, Piattini M and Vallecillo A (eds) "Component-Based Software Quality", Springer Berlin / Heidelberg, volume 2693 of *Lecture Notes in Computer Science*, pp 85–98, http://dx.doi.org/10.1007/978-3-540-45064-1$_5$, 10.1007/978 − 3 − 540 − 45064 − 1$_5$

Malawski M, Bubak M, Baude F, Caromel D, Henrio L and Morel M (2007) "Interoperability of grid component models: GCM and CCA case study", *in* "Towards Next Generation Grids: Proceedings of the CoreGRID Symposium", Springer

Manes S (2007) *Dim Vista*, http://www.forbes.com/forbes/2007/0226/ 050.html

Martinez C P A (2008) *Systematic Construction Of Goal-Oriented COTS Taxonomies*, PhD thesis, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya

Massa P and Avesani P (2009) "Trust Metrics in Recommender Systems", *in* Golbeck J (ed) "Computing with Social Trust", Human-Computer Interaction Series, Springer London, pp 259–285, http://dx.doi.org/10.1007/978-1-84800-356-9$_1$0, 10.1007/978 − 1 − 84800 − 356 − 9$_1$0

Masterman M (1957) "The Thesaurus in Syntax and Semantics", *Mechanical Translation*, 4(1 and 2), pp 35–43

Maxville V (2002) "Intelligent Selection of Components", *in* "Young Researchers Workshop, 7th International Conference on Software Reuse: Methods, Techniques, and Tools (ICSR-7)",

Maxville V (2005) "Knowledge Representation for COTS Selection", *in* "Postgraduate Electrical Engineering and Computing Symposium (PEECS)",

Maxville V (2009) "Preparing scientists for scalable software development", *in* "Workshop on Workshop on Software Engineering for Computational Science and Engineering, ICSE09", Los Alamitos, CA, USA: IEEE Computer Society, volume 0, pp 80–85

Maxville V, Armarego J and Lam C (2004a) "Assessment Methods for Component Selection", *in* "Postgraduate Electrical Engineering and Computing Symposium (PEECS)",

Maxville V, Armarego J and Lam C (2009) "Applying a reusable framework for software selection", *IET Software*, 3(5), pp 369–380

Maxville V, Armarego J and Lam C P (2003a) "The CdCT Process for Component Selection and Evaluation", *in* "Postgraduate Electrical Engineering and Computing Symposium (PEECS)",

Maxville V, Armarego J and Lam C P (2004b) "Learning to Select Software Components", *in* Maurer F and Ruhe G (eds) "International Conference on Software Engineering and Knowledge Engineering (SEKE)", pp 421–426

Maxville V, Lam C P and Armarego J (2003b) "Selecting Components: a Process for Context-Driven Evaluation", *in* "Asia-Pacific Software Engineering Conference (APSEC)", IEEE Computer Society, pp 456–465

Maxville V, Lam C P and Armarego J (2004c) "Intelligent Component Selection", *in* "IEEE signature conference on Computer Software and Applications (COMPSAC)", IEEE Computer Society, pp 244–249

Maxville V, Lam C P and Armarego J (2008) "Supporting component selection with a suite of classifiers", *in* "IEEE Congress on Evolutionary Computation (CEC)", pp 3946–3953

McGregor J D, Stafford J A and Cho I H (2003) "Measuring and Reporting Component Reliability", *in* "In Proceedings of the 1st ACIS International Conference on Software Engineering Research and Applications",

McIlroy D (1968) "Mass-produced software components", *in* "In Software Engineering Concepts and Techniques, 1968 NATO Conference on Software Engineering, J. Buxton, P. Naur, and B. Randell, Eds. 88–98.", pp 88–98

Mendonca M and Sunderhaft N L (1999) *Mining software engineering data: A survey. A DACS state-of-the-art report*, Technical Report, Data Analysis Center for Software, Rome, NY

Meyer B (1992) "Applying "Design by Contract"", *IEEE Computer*, 25(10), pp 40–51

Meyer B (1997) "The Next Software Breakthrough", *IEEE Computer*, 30(7), pp 113–114

Meyer B (1999) "On to Components", *IEEE Computer*, 32(1), pp 139–143

Meyer B (2003) "The Grand Challenge of Trusted Components", *in* "ICSE'03: Proceedings of the 25th International Conference on Software Engineering)", USA, pp 660–667

Mili A, Yacoub S, Addy E and Hafedh M (1999) "Toward an Engineering Discipline of Software Reuse", *IEEE Software*, 16(5), pp 22–31

Mili R, Mili A and Mittermeir R T (1992) "A Formal Approach to Software reuse: design and Implementation", *in* "Proceedings of the 5th Workshop on Software Reuse",

Miller G A (1956) "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information", *The Psychological Review*, 63, pp 18–97

Min H G and Kim S D (2004) "Using Smart Connectors to Resolve Partial Matching Problems in COTS Component Acquisition", *in* Crnkovic I (ed) "CBSE 2004", LNCS 3054, pp 40–47

Mingers J (2001) "Combining IS Research Methods: Towards a Pluralist Methodology", *Information Systems Research*, 12(3), pp 240–259

Mittermeir R, Mili A and Mili A (2007) *Building A Repository of Software Components: A Formal Specifications Approach*

Mohamadali N A K and Garibaldi J M (2009) "A Review of Selected Multi-Criteria Decision Analysis Techniques and Applications", *in* "9th Annual Workshop on Computational Intelligence (UKCI 2009)",

Mohamed A, Ruhe G and Eberlein A (2007a) "COTS Selection: Past, Present, and Future", *in* "IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)", IEEE Computer Society, pp 103–114

Mohamed A, Ruhe G and Eberlein A (2007b) "MiHOS: an approach to support handling the mismatches between system requirements and COTS products", *Requirements Engineering*, 12, pp 127–143

Morisio M and Tsoukiàs A (1997) "IusWare: a methodology for the evaluation and selection of software products", *IEE Proceedings on Software Engineering*, 144(3), pp 162–174

Morris J, Lee G, Parker K, Bundell G A and Lam C P (2001) "Software Component Certification", *IEEE Computer*, 34(9), pp 30–36

Myers G J (1979) *The Art of Software Testing*, New York: John Wiley and Sons

Myerson M (1996) *Risk Management Processes for Software Engineering Models*, Norwood, MA, USA: Artech House, Inc., 1st ed

Nakkrasae S, Sophatsathit P and Edwards W R (2004) "Fuzzy Subtractive Clustering Based Indexing Approach For Software Components Classification", *International Journal of Computer and Information Science*, 5(1)

Navarrete F, Botella P and Franch X (2005) "How Agile COTS Selection Methods are (and can be)?", *in* "EUROMICRO Conference", Los Alamitos, CA, USA: IEEE Computer Society, pp 160–167

Ncube C and Dean J C (2002) "The Limitations of Current Decision-Making Techniques in the Procurement of COTS Software Components", *in* Dean J C and Gravel A (eds) "ICCBSS", Springer, volume 2255 of *Lecture Notes in Computer Science*, pp 176–187

Ncube C and Maiden N A (1999) "PORE: Procurement-Oriented Requirements Engineering Method for the Component Based Systems Engineering Development Paradigm", *in* "International Workshop on Component Based Software Engineering", pp 1–12

Negnevitsky M (2002) *Artificial Intelligence:A Guide to Intelligent Systems*, Addison Wesley

Neubauer T and Stummer C (2007) "Interactive Decision Support for Multiobjective COTS Selection", *in* "Proceedings of the Hawaii International Conference on System Sciences (HICSS)",

Nunamaker J, Chen M and Purdin T D M (1991) "Systems Development in Information Systems Research", *J of Management Information Systems*, 7(3), pp 89–106

Oberndorf T, Brownsword L and Sledge C (2000) *An Activity Framework for COTS-Based Systems*, Technical Report CMU/SEI-2000-TR-010, Carnegie-Mellon University

Ochs M, Pfahl D, Chrobok-Diening G and Nothhelfer-Kolb B (2001) "A Method for Efficient Measurement-based COTS Assessment and Selection - Method Description and Evaluation Results", *in* "IEEE METRICS", IEEE Computer Society, pp 285–285

Ochs M, Pfahl D, Chrobok-Diening G and Nothhelfer-Kolb B (2009) *A COTS Acquisition Process: Definition and Application Experience*, Technical Report IESE-002.00/E, Fraunhofer Institut Experimentelles Software Engineering

Orso A, Harrold M J and Rosenblum D (2000) "Component Metadata for Software Engineering Tasks", *in* "2nd International Workshop on Engineering Distributed Objects", Davis, CA: USA, pp 129–144

Orso A, Harrold M J, Rosenblum D S, Rothermel G, Soffa M L and Do H (2001) "Using Component Metadata to Support the Regression Testing of Component-Based software", *in* "Proceedings of the IEEE International Conference on Software Maintenance (ICSM 2001)", Firenze, Italy, pp 716–725

Page L, Brin S, Motwani R and Winograd T (1999) *The PageRank citation ranking: Bringing order to the Web*, Technical Report, Stanford University, `http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf`

Parnas D L (1972) "On the Criteria To Be Used in Decomposing Systems into Modules", *Communications of the ACM ACM*, 15(12), pp 1053–1058

Pedrycz W, Ekel P and Parreiras R (2011) *Fuzzy Multicriteria Decision-Making: Models, Methods and Applications*, Kluwer Academic Publishers

Petska-Juliussen K and Egil-Juliussen E (2009) *Worldwide PC Market*, `http://www.c-i-a.com/worldwideuseexec.htm`

Pfleeger S L (2001) "What Good Are Metrics? The Views of Industry and Academia", *in* "IEEE METRICS", IEEE Computer Society, p 146

Port D and Chen S (2004) "Assessing COTS Assessment: How Much Is Enough?", *in* "International Conference on COTS-Based Software Systems (ICCBSS)", pp 183–198

Prieto-Diaz R (1991) "Implementing faceted classification for software reuse", *Communications of the ACM ACM*, 34(5), pp 88–97

Quinlan J R (1993) *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Francisco

Quinn P (2009) *Radio Astronomy in the Petascale world*, Presentation, International Centre for Radio Astronomy Research

Rao D V and Sarma V V S (2003) "A Rough:Fuzzy Approach for Retrieval of Candidate Components for Software Reuse", *Pattern Recognition Letters*, 26(6), pp 875–886

Raymond E S (1997) "The Cathedral and the Bazaar", *in* "Linux Kongress", `http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/`

Reifer D J, Basili V, Boehm B and Clark B (2004) "COTS-Based Systems – Twelve Lessons Learned about Maintenance", *in* "International Conference on COTS-Based Software Systems (ICCBSS)", volume LNCS 2959, pp 137–145

Rifkin S (2003) "Why New Software Processes Are Not Adopted", *Advances in Computers*, 59, pp 83–126

Robb T N and Susser B (2000) "The Life and Death of Software: Examining the Selection Process", *CALICO Journal*, 18(1), pp 41–52

Robles G (2005) *Debian Counting*, `http://libresoft.dat.escet.urjc.es/debian-counting`

Rocha C R and Martins E (2008) "A method for model based test harness generation for component testing", *Journal of the Brazilian Computer Society*, 14, pp 7 – 23, `http://www.scielo.br/scielo.php?script=sci`$_{\text{a}}$`rttextpid` = S0104 − 650020080001000003nrm = iso

Rowe W D (1977) *An Anatomy of Risk*, John Wiley Sons

Roy B (1991) "The Outrank Approach and the Foundations of ELECTRE Methods", *Theory and Decisions*, 31, pp 49–73

Royce W W (1970) "Managing the Development of Large Software Systems: Concepts and Techniques", *in* "Proceedings of 26th IEEE WESCON", volume 26, pp 1–9

Ruhe G (2002) "Intelligent Support for Selection of COTS Products", *in* Chaudhri A B, Jeckle M, Rahm E and Unland R (eds) "Web, Web-Services, and Database Systems", Springer, volume 2593 of *Lecture Notes in Computer Science*, pp 34–45

Ruhe G, Eberlein A and Pfahl D (2003) "Trade-off analysis for requirements selection", *International Journal of Software Engineering and Knowledge Engineering*, 13(4), pp 345–366

Saaty T L (1990) *The Analytical Hierarchy Process*, McGraw-Hill, New York

Sankar S and Hayes R (1994) *Specifying and Testing Software Components using ADL*, Technical Report SMLI TR-94-23, Sun Microsystems, `http://labs.oracle.com/techrep/1994/abstract-23.html`

Sassi S, Jilani L and Ghezala H (2003) "COTS Characterization Model in a COTS-based Development Environment", *in* "Asia-Pacific Software Engineering Conference (APSEC)", Chiang Mai, Thailand, pp 352–361

Sassi S, Jilani L and Ghezala H (2004) "Modeling COTS-Based Development and Related Selection Methods Processes with MAP", *in* "Asia-Pacific Software Engineering Conference (APSEC)", Busan, Korea, pp 546–553

Schmid H A (1999) "Business Entity Components and Business Process Components", *Journal of Object-Oriented Programming (JOOP)*, (12), pp 6–15

Seacord R C, Mundie D and Boonsiri S (2001) "K-BACEE: Knowledge-Based Automated Component Ensemble Evaluation", *in* "Proceedings of 27th Euromicro Conference 2001: A Net Odyssey (euromicro'01)", Warsaw, Poland, p 56

Sedigh-Ali S, Ghafoor A and Paul R A (2001) "Software Engineering Metrics for COTS-Based Systems", *IEE Proceedings - Software*, pp 44–50

SEI (2011) `http://www.sei.cmu.edu/index.cfm`

Serban C and Vesca A (2007) "Metrics for component-based system development", *Creative Mathematics and Informatics*, 16, pp 143–150

Sharma A, Kumar R and Grover P (2008) "Estimation of quality for software components: an empirical approach", *SIGSOFT Softw Eng Notes*, 33, pp 1–10, `http://doi.acm.org/10.1145/1449603.1449613`

Shaw M and Garlan D (1996) *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall

Silaghi R and Strohmeier A (2003) "Integrating CBSE, SoC, MDA, and AOP in a Software Development Method", *in* "EDOC '03: Proceedings of the 7th International Conference on Enterprise Distributed Object Computing", Washington, DC, USA: IEEE Computer Society, p 136

Sim S E, Easterbrook S and Holt R C (2003) "Using benchmarking to advance research: a challenge to software engineering", *in* "Proceedings of the 25th International Conference on Software Engineering", ICSE '03: Proceedings of the 25th International Conference on Software Engineering, Washington, DC, USA: IEEE Computer Society, pp 74–83, `http://dl.acm.org/citation.cfm?id=776816.776826`

Singh H, Conrad M and Sadeghipour S (1997) "Test Case Design Based on Z and the Classification Tree Method", *in* "Proceedings of the 1st International Conference on Formal Engineering Methods", IEEE Computer Society, pp 81–, `http://dl.acm.org/citation.cfm?id=523981.852147`

Solberg H and Dahl K M (2001) *COTS Software Evaluation and Integration Issues*, Technical Report SIF8094, Norwegian Institute of Technology and Science, `http://www.idi.ntnu.no/grupper/su/sif8094-reports/2001/p14.pdf`

Sorensen R L (2004) *Systems Engineering in a COTS World*, Technical Report, Vitech Corporation

Sparck Jones K (1970) "Some thoughts on classification for retrieval", *Journal of Documentation*, 26(2), pp 89–101

Sparck Jones K (1972) "A statistical interpretation of term specificity and its application in retrieval", *Journal of Documentation*, 28(1), pp 11–21

Spivey J M (1992) *The Z Notation: a reference manual*, International Series in Computer Science, Prentice Hall, 2nd ed

Stafford J and Wallnau K C (2001) "Predictable Assembly from Certifiable Components", *in* Pulvermüller E, Speck A, Coplien J, D'Hondt M and DeMeuter W (eds) "Feature Interaction in Composed System, ECOOP 2001 Workshop 08", pp 35–41

Stocks P and Carrington D (1993) "Test Template Framework: A Specification-based Case Study", *in* "International Symposium On Software Testing And Analysis",

Stylianou C and Andreou A S (2007) "A Hybrid Software Component Clustering and Retrieval Scheme Using an Entropy-Based Fuzzy k-Modes Algorithm", *in* "19th IEEE international Conference on Tools with Artificial intelligence (ICTAI)", IEEE Computer Society, volume 1, pp 202–209

Szyperski C (1998) *Component software: Beyond Object-Oriented Programming*, New York: ACM Press

Szyperski C and Messerschmitt D G (2003) "The Flexible Factory", *Software Development*, 11(12), pp 30–34

Tahat L (2001) "Requirement-Based Automated Black-Box Test Generation", *in* "25th Annual International Computer Software and Applications Conference", IEEE Computer Society Press, pp 489–495

Taleghani A (2007) "Using Software Model Checking for Software Component Certification", *in* "ICSE Companion'07", pp 99–100

Tichy W (1997) "Should Computer Scientists Experiment More 16 Excuses to Avoid Experimentation. 16 Excuses to Avoid Experimentation", *IEEE Computer*, 31, pp 32–40

Tran V and Lin D B (1999) "Application of CBSE to projects with evolving requirements-a lesson-learned", *in* "Sixth Asia Pacific Software Engineering Conference (APSEC '99)", pp 28 –37

Triantaphyllou E (1995) "Linear programming based decomposition approach in evaluating priorities from pairwise comparisons and error analysis", *J Optim Theory Appl*, 84, pp 207–234, `http://dl.acm.org/citation.cfm?id=213441.213473`

Triantaphyllou E (2001) *Multi-Criteria Decision Making Methods: A Comparitive Study*, Kluwer Acedemic Publishers

Udell J (1994) "Componentware", *Byte*, 19(5), pp 46–56

Vallecillo A, Hernandez J and Troya J M (2000) "Component Interoperability", *in* "European Conference on Object-Oriented Programming (ECOOP '99) Reader", LNCS 1743, Springer-Verlag, pp 1–21

Voas J (1998a) "An Approach to Certifying Off-the-Shelf Software Components", *IEEE Computer*, 31, pp 53–59

Voas J (1998b) "The Software Quality Certification Triangle", *CROSSTALK The Journal of Defense Software Engineering*, 11(11), pp 12–14

Voas J (2000) "Developing a Usage-Based Software Certification Process", *IEEE Computer*, (33), pp 32–37

Voas J (2001) "Composing Software Component 'ilities"', *IEEE Software*, 18(4), pp 16–17

Voinea L and Telea A (2005) "Visual Assessment Techniques for Component-Based Framework Evolution", *in* "Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications", EUROMICRO '05, Washington, DC, USA: IEEE Computer Society, pp 168–179, `http://dx.doi.org/10.1109/EUROMICRO.2005.65`

Wallnau K C, Clements P and Zaremski A (1997) "Correcting, Identifying and Avoiding Interface Mismatch: Theory and Practice", *in* "ICSE '97: Proceedings of the 19th international conference on Software engineering",

Washizaki H (2003) "A Metrics Suite for Measuring Reusability of Software Components", *IEE Proceedings - Software*

Watkins D (1998) "Using Interface Definition Languages to Support Path Expressions and Programming by Contract", *in* "Technology of Object-Oriented Languages and Systems (TOOLS 26)", p 308

Wegner P (1996) "Interoperability", *ACM Computing Survey*, 28(1), pp 285–287

Weinberg G (1971) *The Psychology of Computer Programming*, Dorset House

Weyuker E (1998) "Testing Component-based Software: A Cautionary Tale", *IEEE Software*, 15(5), pp 54–59

White L and Leung H (1992) "A firewall concept for both control-flow and data flow in regression integration testing", *in* "Proceedings of International Conference on Software Maintainance", pp 262–271

Williams K B (2004) *Grace Hopper: admiral of the cyber sea*, Naval Institute Press

Wirth N (2008) "A Brief History of Software Engineering", *IEEE Annals of the History of Computing*, 30, pp 32–39

Witten I, Frank E and Hall M A (2011) *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann, San Francisco, 3 ed

Wohlin C and Runeson P (1994) "Certification of Software Components", *IEEE Transactions on Software Engineering*, 20(6), pp 494–499

Wohlin C, Runeson P, Host M, Ohlsson M C, Regnell B and Wesslen A (2000) *Experimentation in Software Engineering: An Introduction*, Kluwer Academic Publishers

Woodman M, Benediktsson O, Lefever B and Stallinger F (2001) "Issues of CBD Product Quality and Process Quality", *in* "The 4th ICSE Workshop on Component-Based Software Engineering (CBSE)", Canada, pp 55–57

Wu H, Luk R, Wong K and Kwok K (2008) *ACM Transactions on Information Systems*, 26(3), pp 1–37

Wu X, McMullan D and Woodside M (2003) "Component Based Performance Prediction", *in* Crnkovic I, Schmidt H, Stafford J and Wallnau K (eds) "6th ICSE Workshop on Component-Based Software Engineering: Automated Reasoning and Prediction", Carnegie Mellon University, USA and Monash University, Australia

Xie X and Zhang W (2007) "A framework for software component adaptation", *in* Jin H (ed) "ICA3PP", Springer-Verlag Berlin / Heidelberg, volume 4494, pp 153–164

XML (2004) *XML Schema Part 0: Primer Second Edition*, `http://www.w3.org/TR/xmlschema-0/`

Yacoub S, Ammar H and Mili A (1999) "Characterizing a Software Component", *in* "International Conference in Software Engineering '99",

Yamamoto K and Saeki M (2007) "Using Attributed Goal Graphs for Software Component Selection: An Application of Goal-Oriented Analysis to Decision Making", *in* "Proceedings of the 26th International Conference on Conceptual Modeling ER 2007 - Tutorials, Posters, Panels and Industrial Contributions", `http://crpit.com/confpapers/CRPITV83Yamamoto.pdf`

Yoon H, Choi B and Jeon J (1999) "A UML Based Test Model for Component Integration Test", *in* "Workshop on Software Architecture and Component", Japan

Zheng J, Robinson B, Williams L and Smiley K (2006) "Applying regression test selection for COTS-based applications", *in* "ICSE '06: Proceedings of the 28th International Conference on Software Engineering", New York, NY, USA: ACM, pp 512–522

Appendix A

Glossary

API An application programming interface (API) is an interface that a software program implements in order to allow other software to interact with it.
Source: en.wikipedia.org/wiki/Api

ARFF An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files have two distinct sections. The first section is the Header information, which is followed the Data information
Source: weka.wikispaces.com/ARFF

CBD Component-based development

CBS Component-based systems

CBSE Component-based software engineering

CdCT Context-driven Component Testing - previous name for CdCE

CdCE Context-driven Component Evaluation

Component-intensive systems Systems assembled from components, or a mixture of components and new code

COTS A COTS (commercial off-the-shelf) product is one that is used 'as-is'. COTS products are designed to be easily installed and to interoperate with existing system components.
Source: searchenterpriselinux.techtarget.com/sDefinition

DOM The Document Object Model (DOM) is a cross-platform and language-independent convention for representing and interacting with objects in HTML
Source: en.wikipedia.org/wiki/XML-DOM

DTD Document Type Definition (DTD) is a set of markup declarations that define a document type for SGML-family markup languages (SGML, XML, HTML).
Source: en.wikipedia.org/wiki/.dtd

GOTS A GOTS (government off-the-shelf) product is typically developed by the technical staff of the government agency for which it is created. It is sometimes developed by an external entity, but with funding and specification from the agency.
Source: searchenterpriselinux.techtarget.com/sDefinition

GPL GPL is short for General Public Licence and is a type of licence published by the GNU Project.
Source: cplus.about.com/od/glossar1/g/gpldefinition.htm

GQM The Goal—Question—Metric method is a structured approach to evaluation, discussed in Section 3.1

Ontology In computer science and information science, an ontology is a formal representation of the knowledge by a set of concepts within a domain and the relationships between those concepts.
Source: http://en.wikipedia.org/wiki/Ontology

swvML software verification Markup Language, previously cpML, component Markup Language

OSS Open-source Software - an approach to software development where source code is available

OTS Off the shelf (software) - software developed by a third party which may be purchased or used under licence

RDF Resource Description Framework relies on XML as an interchange syntax, creating an ontology system for the exchange of information on the Web.
Source: isp.webopedia.com/TERM/R/RDF.html

SAX SAX is a standard API for event-based XML parsing, and SAX implementations are available in different programming languages.
Source: dret.net/glossary/sax

SDM Spiral Development Method, an iterative software development method, developed by Barry Boehm

W3C World Wide Web Consortium - develops and maintains standards, such as XML

XML XML (Extensible Markup Language) is a set of rules for encoding documents electronically.
Source: en.wikipedia.org/wiki/XML

XML Schema An XML schema describes the structure of an XML document.
Source: www.w3schools.com/schema/schema_intro.asp

XSLT XSL Transformations (XSLT) is a declarative, XML-based language used for the transformation of XML documents into other XML documents.
Source: en.wikipedia.org/wiki/XSLT

WEKA Weka is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. Weka contains tools for data pre-processing, classification, regression, and clustering.
Source: http://www.cs.waikato.ac.nz/ ml/weka/

Appendix B

Code and Scripts

B.1 Scripts from Spiral 4

B.1.1 xml_exp_SEARCH

```
#!/bin/bash
echo
echo "############################################################"
echo
echo "                  CALCULATOR CASE STUDY (On the Mac!)"
echo
echo "############################################################"

base=$1

./process ${base}t5s10
./process ${base}t5s21
./process ${base}t5s22
./process ${base}t5s23
./process ${base}t5s31
./process ${base}t5s32
./process ${base}t5s33
./process ${base}t5s41

./grab_predict $base

echo "############################################################"
echo
echo " Don't forget to move the files before running the script again."
echo
echo " (grab_predict doesn't like it)"
echo
echo "############################################################"
```

B.1.2 process

```
#!/bin/bash

# laptop version 15/6/05
# Re-saved to run on Mac - CR/LF problem, 31/12/08
```

```
# Rejigged paths for Mac 31/12/08
# Needed old xerces - see http://archive.apache.org/dist/xml/xerces-j/ for version 1_4_4
# Had to change j48.J48 to J48 for Weka 31/12/08

echo
echo "############################################################"
echo
echo "            INTELLIGENT COMPONENT SELECTION"
echo
echo "############################################################"
echo

# see http://www.tldp.org/LDP/abs/html/io-redirection.html for
# output redirection options


if [ -z "$1" ]; then
  echo "ERROR: No file base given - exiting."
  echo
  echo "Usage: process file_base"
  echo
  exit 1
fi

echo "Setting up directory for running experiment on >> $1 <<"

EXPERIMENT=$1
WORKDIR="${1}_`date +%F_%H_%M`"
TEMPDIR="TEMP_`date +%F_%H_%M`"
WEKAHOME="/Users/valeriemaxville/_Thesis/dev_new/Weka-3-6-0/weka.jar"
#WEKAHOME="/Users/valeriemaxville/_Thesis/dev_new/Weka-3-4-14/weka.jar"
#WEKAHOME="/Users/valeriemaxville/_Thesis/dev_new/weka-3-0-6.jar"
JAVACLASSPATH=".:/Users/valeriemaxville/_Thesis/dev_new/FMfilter/classes:
      /Users/valeriemaxville/_Thesis/dev_new/xerces-1_4_4/xerces.jar:
      /Users/valeriemaxville/_Thesis/dev_new/xml-writer-0.2/xml-writer.jar:${WEKAHOME}"

echo "Directory name is $WORKDIR"

if [ -d "$WORKDIR" ]; then
  echo
  echo "ERROR: Directory >> $WORKDIR << already exists, "
  echo "        please delete/rename then re-run script."
  echo
  exit 1
fi


mkdir $WORKDIR
mkdir $TEMPDIR

echo "Copying files..."

if [ -e "*${EXPERIMENT}*.arff" ]; then   # Keep any arff files safe
    mv *.arff $TEMPDIR
fi

cp parameters_${EXPERIMENT}.xml $WORKDIR
cp ideal_${EXPERIMENT}.xml $WORKDIR
#  cd $WORKDIR
```

```
echo "Running programs..."

echo "   CdCETransformer"
java -classpath $JAVACLASSPATH fmfilter.CdCETransformer parameters_${EXPERIMENT}.xml
    1> ${WORKDIR}/transformer_output.txt 2> ${WORKDIR}/transformer_errors.txt
echo "   Intelligent"
java -classpath $JAVACLASSPATH fmfilter.Intelligent parameters_${EXPERIMENT}.xml
    1> ${WORKDIR}/generator_output.txt 2> ${WORKDIR}/generator_errors.txt

mv *${EXPERIMENT}*.arff $WORKDIR
if [ -e "${TEMPDIR}/*.arff" ]; then
  mv ${TEMPDIR}/*.arff .              # Bring back saved files
fi
rmdir $TEMPDIR

cd $WORKDIR

echo "    ...calling weka_train..."
../weka_train .
```

B.1.3 weka_train

```
#!/bin/bash

# Laptop version - altered path to J48 17/6/05
# 30Dec06 - changed *FM* to *fm* on line 32
# Re-saved to run on Mac - CR/LF problem, ::: 31/12/08
# Rejigged paths for Mac ::: 31/12/08
# Needed old xerces - see http://archive.apache.org/dist/xml/xerces-j/
#              for version 1_4_4 ::: 31/12/08
# Played with other weka versions, but didn't solve problem - using 3-6-0 now. ::: 1/1/09

echo
echo "############################################################"
echo
echo "               WEKA TRAINING AND TESTING"
echo
echo "############################################################"
echo

# Assumes that there's only one training file in the directory - could loop through later

if [ -z "$1" ]; then
  echo "ERROR: No directory given - exiting."
  echo
  echo "Usage: weka_train directory"
  echo
  exit 1
fi

WEKAHOME="/Users/valeriemaxville/_Thesis/dev_new/Weka-3-6-0/weka.jar"
#WEKAHOME="/Users/valeriemaxville/_Thesis/dev_new/Weka-3-4-14/weka.jar"
#WEKAHOME="/Users/valeriemaxville/_Thesis/dev_new/weka-3-0-6.jar"
JAVACLASSPATH=".:/Users/valeriemaxville/_Thesis/dev_new/FMfilter/classes:
   /Users/valeriemaxville/_Thesis/dev_new/xerces-1_4_4/xerces.jar:
   /Users/valeriemaxville/_Thesis/dev_new/xml-writer-0.2/xml-writer.jar"
```

```
WORKDIR=$1

pushd $WORKDIR

TRAININGFILE=‘ls *train.arff‘
TEST1=‘ls *test1.arff‘
TEST2=‘ls *test2.arff‘
REAL=‘ls *fm*.arff‘
# REAL=‘ls *FM*.arff‘

echo "#### WEKA TRAINING ####" >>train_output.txt
echo >>train_output.txt
echo "#### WEKA TRAINING ####"
echo
echo "Classpath is >> $JAVACLASSPATH <<"
echo "Wekahome is >> $WEKAHOME <<"
echo "Training file is >> $TRAININGFILE <<"

echo "Training file : ${TRAININGFILE}"  >> train_output.txt
echo "Test file 1   : ${TEST1}"  >> train_output.txt
echo "Test file 2   : ${TEST2}"  >> train_output.txt
echo "Real data     : ${REAL}"  >> train_output.txt
echo >>train_output.txt
echo "#### TRAINING ####" >>train_output.txt
echo "#### TRAINING ####" >>train_error.txt
echo "#### TRAINING ####"
java -Xint -classpath "${WEKAHOME}" weka.classifiers.trees.J48 -t $TRAININGFILE
    -d model.mod 1>> train_output.txt 2> train_error.txt
#java -Xint -classpath "${WEKAHOME}" weka.classifiers.trees.j48.J48 -t $TRAININGFILE
    -d model.mod 1>> train_output.txt 2> train_error.txt

echo "#### TEST1 ####" >>train_output.txt
echo "#### TEST1 ####" >>train_error.txt
echo "#### TEST1 ####"
java -Xint -classpath "${WEKAHOME}" weka.classifiers.trees.J48 -l model.mod -T $TEST1
    1>> train_output.txt 2>> train_error.txt
#java -Xint -classpath "${WEKAHOME}" weka.classifiers.trees.j48.J48 -l model.mod -T $TEST1
    1>> train_output.txt 2>> train_error.txt

echo "#### TEST2 ####" >>train_output.txt
echo "#### TEST2 ####" >>train_error.txt
echo "#### TEST2 ####"
java -Xint -classpath "${WEKAHOME}" weka.classifiers.trees.J48 -l model.mod -T $TEST2
    1>> train_output.txt 2>> train_error.txt
#java -Xint -classpath "${WEKAHOME}" weka.classifiers.trees.j48.J48 -l model.mod -T $TEST2
    1>> train_output.txt 2>> train_error.txt

echo "#### REAL ####" >>train_output.txt
echo "#### REAL ####" >>train_error.txt
echo "#### REAL ####"
java -Xint -classpath "${WEKAHOME}" weka.classifiers.trees.J48 -l model.mod -T $REAL
    1>> train_output.txt 2>> train_error.txt
#java -Xint -classpath "${WEKAHOME}" weka.classifiers.trees.j48.J48 -l model.mod -T $REAL
    1>> train_output.txt 2>> train_error.txt

popd
```

B.1.4 grab_predict

```
#!/bin/bash

# 17/6/05 - Laptop version - changed javaw to java and updated classpath
# 2/1/06 - Updated component filename
# 1/1/08 - Added lines in loop to generate separate shortlists for each set in suite

echo
echo "############################################################"
echo
echo "                   GRABBING RESULTS"
echo
echo "############################################################"
echo

base=$1
#WEKAHOME="/Users/valeriemaxville/_Thesis/dev_new/Weka-3-6-0/weka.jar"
#WEKAHOME="/Users/valeriemaxville/_Thesis/dev_new/Weka-3-4-14/weka.jar"
WEKAHOME="/Users/valeriemaxville/_Thesis/dev_new/weka-3-0-6.jar"
JAVACLASSPATH=".:/Users/valeriemaxville/_Thesis/dev_new/FMfilter/classes:
    /Users/valeriemaxville/_Thesis/dev_new/xerces-1_4_4/xerces.jar:
    /Users/valeriemaxville/_Thesis/dev_new/xml-writer-0.2/xml-writer.jar"

rm ${base}_predict.txt
rm ${base}_predict_detail.txt
rm ${base}_inst.txt
rm ${base}_shortlist.xml

for w_dir in `ls -d ${base}*`
do
   if [ -d $w_dir ]; then
      ./weka_predict $w_dir
      grep accept ${w_dir}/train_predict.txt >> ${base}_predict.txt
      grep accept ${w_dir}/train_predict.txt >> ${w_dir}/accept_predict.txt
      echo "Matches in ${w_dir}" >> ${base}_predict_detail.txt
      MATCHCOUNT=`grep accept ${w_dir}/train_predict.txt | wc -l`
      echo "Count : ${MATCHCOUNT}" >> ${base}_predict_detail.txt
      grep accept ${w_dir}/train_predict.txt >> ${base}_predict_detail.txt
      grep accept ${w_dir}/train_predict.txt >> ${w_dir}/accept_predict_detail.txt
      awk '{print $1}' ${w_dir}/accept_predict_detail.txt > ${w_dir}/inst.txt
      java -classpath $JAVACLASSPATH fmfilter.Grabber ${w_dir}/inst.txt fm_projects06_CdCE.xml
          ${w_dir}/shortlist.xml
   fi
done

#awk -F: '{print $2}' ${base}_predict.txt | awk '{print $1}' > ${base}_inst.txt
awk '{print $1}' ${base}_predict.txt > ${base}_inst.txt

java -classpath $JAVACLASSPATH fmfilter.Grabber ${base}_inst.txt fm_projects06_CdCE.xml
    ${base}_shortlist.xml

echo "Number of unique items is: "
echo "Number of unique items is: " > ${base}_unique.txt
grep "<swv:component>" ${base}_shortlist.xml | wc -l
grep "<swv:component>" ${base}_shortlist.xml | wc -l >> ${base}_unique.txt

WORKDIR="${base}_results_`date +%F_%H_%M`"
```

```
mkdir $WORKDIR
mv ${base}_predict.txt $WORKDIR
mv ${base}_predict_detail.txt $WORKDIR
mv ${base}_inst.txt $WORKDIR
mv ${base}_shortlist.xml $WORKDIR
mv ${base}_unique.txt $WORKDIR
```