A Distributed Solution to Software Reuse

James F. McGuigan

M.Sc. Thesis

Department of Computer Science
University of Durham

September 2008

Abstract

Reuse can be applied to all stages of the software lifecycle to enhance quality and to shorten time of completion for a project. During the phases of design and implementation are some examples of where reuse can be applied, but one frequent obstruction to development is the building of and the identifying of desirable components. This can be costly in the short term but an organisation can gain the profits of applying this scheme if they are seeking long-term goals.

Web services are a recent development in distributed computing. This thesis combines the two research areas to produce a distributed solution to software reuse that displays the advantages of distributed computing within a reuse system. This resulted in a web application with access to web services that allowed two different formats of component to be inserted into a reuse repository. These components were searchable by keywords and the results are adjustable by the popularity of a component's extraction from the system and by user ratings of it; this improved the accuracy of the search. This work displays the accuracy, usability, and speed of this system when tested with five undergraduate and five postgraduate students.

Acknowledgements

This MSc thesis is dedicated to Anne McGuigan. My life is her success.

Copyright

Declaration

No part of the material offered has previously been submitted by the author for a degree in the University of Durham or in any other university. All of the work presented here is the sole work of the author and no one else.

Contents

List of Tables

List of Figures

Chapter 1 Introduction

1.1 Background

The software crisis has been with us for quite some time [Paulk95], and is not diminishing. As hardware prices dramatically decrease, these days more people can own their own hardware systems. So the demand for software by which hardware systems operate is exploding while programmers' productivity is limited. Further evidence of this difference is the fact that many software projects finish over budget. This difference between demand and supply for software resulted in an enormous gap between hardware and software development during the past few decades.

Another aspect of the software crisis is the lack of quality. Although quality can be a subjective characteristic, overall system quality usually can be accessed in terms of providing the functionality expected by the customer, meeting customer performance requirements, and freedom from defects.

In addition to them, the quality factors of a software system also contains working as advertised, having acceptable use of time and space resources (efficiency), being composable with other components (composiability), being understandable by clients and maintainers, and being usable in a possibly different context (portability or rehostability) [Bator92].

In the former approach, many software engineers have focussed on improving the software development process. This approach usually includes the use of computer aided software engineering (CASE) tools. The hope is that improvements in how an organisation goes about managing software development will lead to better productivity and to higher quality systems [Paulk95].

In recent years, researchers have aimed at providing a means for organisations to integrate their processes together between multiple sites. Distributed technologies focus upon providing a means for interoperability between heterogeneous systems, and allows for the adoption of new software development processes such as reuse to be instated within a global institution. Web service technologies are a field within distributed computing that aims to accelerate application integration inside and outside enterprises by providing a language-neutral, environment-neutral programming model [Gotts00].

The following chapters will examine software reuse, reverse engineering, component based engineering, electronic data interchange (EDI), design patterns and distributed computing. From this analysis, a CASE tool is proposed and developed that aims to integrate software reuse across a distributed organisation.

1.2 The Criteria for Success

The main objective of this research is to propose a system that enables an organisation to introduce a reuse approach throughout the various stages of the software lifecycle. The criteria for the success of this system are the following:

- Suggest guidelines for an approach to code reuse.
- Identifying criteria that are used to select a component for reuse within a repository.
- Provide a distributed tool that enables many employees within an organisation to insert and search for reusable components.
- Within the distributed reuse system, design a search mechanism that will provide accurate search results that reflects upon the many façades a component can be viewed from.
- Validating the usefulness and usability of the distributed reuse system.

The above criteria will be judged in Chapter Chapter 7, Conclusion.

1.3 Outline of Thesis

The thesis is organised as follows. Chapter 2 introduces the general principles relating to software reuse and distributed computing. Within this chapter, consideration of areas that do not just involve software based, but other organisational based issues concerning reuse is undertaken. Chapter 3 describes the architecture and concepts behind the proposed system. Chapter 4 details how the system described in Chapter 3 is implemented. In chapter 5, the approach taken into how the system is measured for success is described. The following chapter evaluates the results obtained from chapter five using the criteria from chapter 2. Finally, chapter 7 discusses possible future work and the conclusions drawn from the work so far.

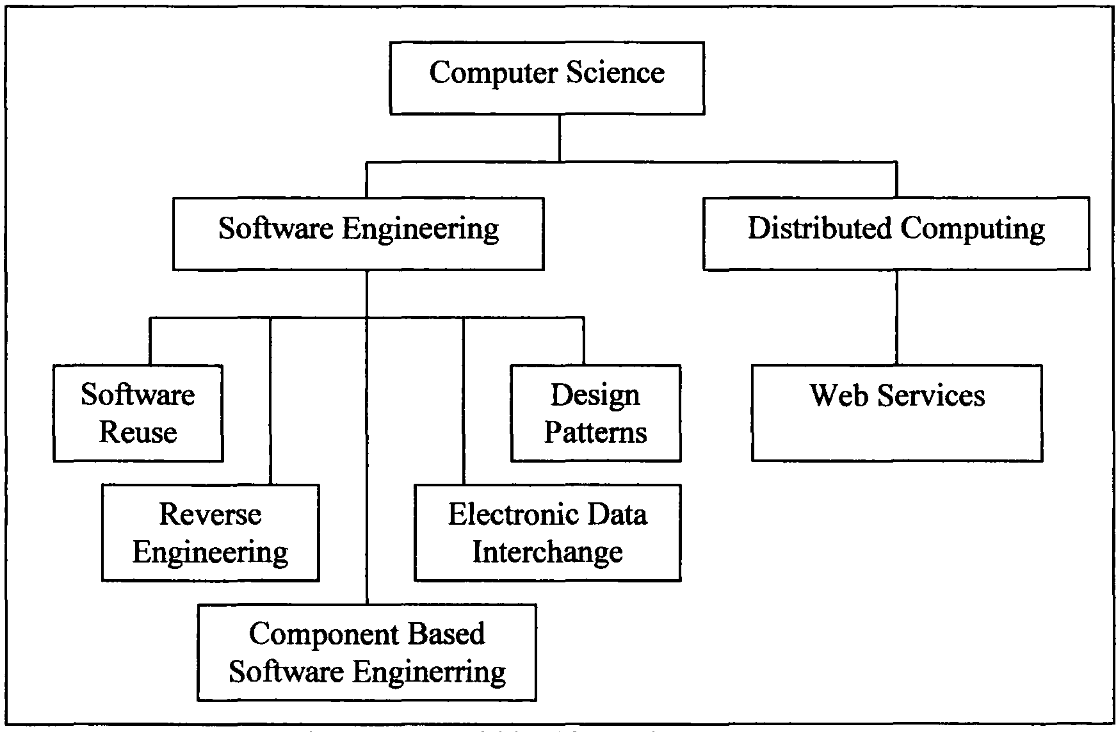


Figure 1.3-1: Research coverage within this work.

Figure 1.3-1 describes the research areas involved within this thesis, and displays how they fit together. The work presented in this thesis has links with other research topics in software engineering such as “software cost estimation”, “software safety” or “distributed transactions”.

Chapter 2 Literature Survey

2.1 Component Based Software Engineering

McIlroy [McIl68] foresaw software development becoming the process of constructing software from standard interchangeable building blocks. Component-based Software Engineering (CBSE) is a methodology that supports the compositional approach the compositional approach to building software applications involving 'plug-and-play' software components (custom-built or Commercial Off-The-Shelf) in a framework. Recent developments such as the shift from centralised mainframe-based to distributed applications and the need to reuse existing resources in the business and organisational contexts [Brown98] are accelerating the use of CBSE for application development. Morris et al [Morri03] defines how reuse in CBSE differs from conventional reuse. Components are:

- *Required to interoperate with other components as well as the frameworks.*
- *Required to hide their implementation details and thus their interfaces are separated from their implementations.*
- *Usually designed on a pre-defined architecture to permit interoperability.*

Component development and integration are the two key processes in CBSE. The component-based "*enterprise software process model*" [Aoyam95] for application development consists of the following sequential stages:

- *Analysis and Component Acquisition*
- *Component-Orientated Design*
- *Component Composition*
- *Integration Testing and System Testing*

Developers during component integration often never see the source code of the components being reused; therefore, a 'black box' approach to development is taken. With black box CBSE, a number of factors must be taken into account. Weyuker [Weyuk98] lists these factors as:

- *Mismatch which can arise between component from several sources*
- *Incomplete or incorrect behavioural specifications for the components*
- *Components are highly volatile as they are often upgraded – leading to cases where upgrades may not have the required capability or bug fixes*

All these factors contribute to making integration an error prone process producing systems that are difficult to test and debug [Morri03].

2.1.1 Software Crisis

The 'software crisis' [Paulk95] of the 60,70,80, and 90's often produced software systems; that were delayed in their delivery to the client, incurred escalated costs, had reduced functionality to which was previously planned for, and contained a high number of faults. It is seen as a long-term inability of organisations to create software in a predictable, efficient, and timely manner [Brook95].

80% of all embedded systems are delivered late, and that much of the delay arises in the software infrastructure of the system rather than the applications [Web03]. While functionality is common, the requirements differ greatly. Despite many attempts to create a "Silver Bullet" that solves the software crisis, no one simple solution has been found, and will likely to be found [Brook95]. Software development is a complex web of technical, business, personnel, and sociological factors that are difficult to balance [Dykma99]. Complex technical problems have to be addressed and resolved by the discovery of tools that address processes. Formal methods are applied towards system development to ensure a reliable system. More visual approaches such as UML are used to define a visual modelling language to attempt to capture component requirements and design component classes and interfaces more accurately. Use cases generated can be used to derive test cases [Morri03]. Another approach involves testing the components for each new environment so that developers and users can predict behaviour and performance. This is not a very feasible approach as it may incur significant cost [Weyuk98].

Once a correct approach to component development and reuse has been developed, CBSE systems have a very high reliability rate. Based on limited analysis of data from the Department of Defence, 99% of all executing instructions come from COTS components [Boehm99].

The tools discovered must be compatible with third-party systems or legacy systems. These tools are seen as the 'Golden Gun' of software development, and has the ability for people, software tools, and processes to be carefully combined together and managed to create quality software [Tracz95].

Brooks [Brook95] predicts "that no single tool or technology would provide an order-of-magnitude gain in software productivity, reliability or simplicity in the next 10 years". Tracz's [Tracz95] description of software reuse supports this Brooks's [Brook95] statement and also adds that "these tools can be expensive; a proper investment in tools has a positive return on investment and provides increased productivity and quality" [Tracz95]. He also believes that these tools are essential in creating high-quality software.

CBSE cannot be used effectively until it can be employed within the context of well-understood methods for designing, assembling and maintaining component-based systems [Weyuk98]. Frey and Rosvall [Web03] believes that this has led to a "very low level of standardisation and reuse in resource-constrained embedded systems" [Web03]. Embedded applications have become increasingly complex over the last decades. The increases in functionality and complexity are related to infrastructure rather than the actual applications [Web03]. Infrastructure functionality is often tightly coupled to the application due to pressed time schedules that do not permit a proper design where application and infrastructure are clearly separated. This greatly hinders software reuse.

2.2 Software Reuse

The fundamental unit of software reuse is a component [Bator92]. The identification of similar requirements and artefacts at an early stage can enable the reuse of components at early stages of the development process. However, Karlsson [Karls95] explains that the attitude in industry is for insufficient time to be spent in the earlier phases of the development process such as analysis and design, in which the possibility for reusing existing components and defining new reusable components is greatest. This strengthens the point that software reuse "is just not limited to source code fragments but may also include design documents, specifications..." [Czarn00, Krueg92], and is further supported by Select Business Solutions [Selec03a] which states that "reuse reduces the amount of work to be undertaken by a project; the earlier an asset can be reused, the larger the scale of saving to the project".

Software reuse is fundamentally...

"...a means to improve the practice of software engineering by using existing software artefacts during the construction of new software systems" [Krueg92].

Dykman [Dykma99] expands Krueger's [Krueg92] definition of software reuse to reflect upon the possibility of applying components within it by explaining reuse as...

"...the use and development of software artefacts that are used over and over again in a number of different but related software projects".

Prieto-Diaz and Freeman [Priet87] identifies that the identification of reusing existing software artefacts is through "a matching process between new and old situations and when matching succeeds, duplications of the same actions", and is supported by Select Business Solutions [Selec03a] statement that "reuse requires a memory - a memory of the intellectual property invested in each of the reusable assets so that the intellectual property can be maintained and expanded as the business changes. The memory is best maintained by the adoption of some level of tooling to support the development process".

2.2.1 Previous Applications and Advantages

Results from the field of business show that there were considerable increases in productivity, quality, and maintenance. These results are seen in Japan where Meyers [Meyer98] highlights that the standard productivity is approximately 3,600 lines of code lines per year while the total in Japan is 24,000. This figure correlates to the wider reuse of reusable software within Japan that has been shown to have reuse factors of 85% [Stand84, McNam84]. If an asset is reused then its lifetime will be extended, increasing the returns that can be achieved. Any organisations seek to achieve a rate of return by limiting the lifetime over which development costs are written-off. If the level of reuse is high enough, then the rate of return will be increased [Selec03a]. The rate of return generated by a solution is reduced by its cost of maintenance. Ultimately these costs may become so significant that they act as a significant brake on the rate of change within the organisation. The resulting paralysis will significantly increase operating costs and reduce the competitive advantage held by the organisation [Selec03a]. To follow the example of Japanese software factories, a change of western cultures must be undergone to gain the advantages of software reuse. These advantages have led to quicker delivery of systems that is essential in today's competitive markets. This allows organisation to be more responsive to commercial pressures and derive real value from new solutions more quickly [Selec03a]. Agresti [Agres99] lists a number of specific gains other than productivity that are gained from reuse.

- **Reliability**: *through the use of proven components*
- **Consistency**: *by using the same components in many places, this reduces the need for fresh and possibly idiosyncratic design.*
- **Manageability**: *using well-understood components as reuse reduces the likelihood of cost and schedule overruns by providing already developed components whose behaviour is understood.*
- **Standardisation**: *using libraries of components*

2.2.2 Incorporating a Software Reuse Program inside an Organisation

There is a need for high initial investments to implement an effective reuse programme because current business processes have to be reorganised and new roles created [Lim98]. Lim [Lim98] outlined these roles as being:

- **Influencer/Consultant**: *captures and transfers technology and knowledge through classes, tutorials, handbooks, and consulting.*
- **Producer/Business:** *delivers course on designing with reuse and produces reusable assets.*
- **Librarian/Broker:** *provides a library service for the storage of reusable components.*

Influencer/Consultant acts as a catalyst within the organisation and keeps abreast of reuse developments. It requires fewer resources to be required from the organisation, and provides a divisional reuse program. This may mean that there is potential for projects to deviate from standards and that future libraries may not have the ability to be integrated. Divisional reuse teams may be more aware and responsive to the needs of their consumers.

The focus for the producer/business role is the creation and maintenance of reusable assets. This will incur activities such as domain analysis and infrastructure review to produce assets that will be profitable. Expertise for this role is needed to produce highly generalised assets that can be reused across projects; however generalising programmable code reduces the efficiency of the program it lies in.

The librarian/broker provides information, and advice towards reuse within an organisation. Centralised reuse architecture provides the possibility of this role. However, the components produced and collected within this system are designed towards reuse across many projects throughout the organisation and are highly unlikely to be reused within a project. The ability for distributing reusable assets is provided by a library tool [Lim98].

Karlsson [Karls95] provides a more in-depth evaluation of the new roles and adaptations of existing roles for the integration of reuse into an organisation. He focuses upon three views within an organisation, development, management, and support.

Within the development view, Karlsson [Karls95] identifies that a developer could be developing for or with reuse. Different activities are needed in each area, and that a developer is placed into one of the following two categories, actual reuser or potential reuser. An actual reuser is classified as waiting for a developing component while a potential reuser is in the future. The requirements for a potential reuser are harder to predict because it is impossible to define exactly what will be required.

There are two different roles in the development for reuse. Firstly, integration aims their development on functionality, performance, and the quality of component for integrating the component into a subsystem or product [Karls95]. Secondly, an adapter role imposes requirements as the reusability of the component i.e. how easy it is to adapt.

Development with reuse involves the process of continually searching and evaluating components that may be reused to satisfy these requirements. There are two approaches for a developer:

- *Change requirements so that the component fits "as it is" into a subsystem.*
- *Adapt components to fit requirements*

Often it is beneficial for requirements to be altered so that components can fit seamlessly into a system. Introducing components like this can gain benefits such as "Qualification of the development process used to create and maintain it" [Kunda00] and can reduce the costs for "adapting and integrating the COTS, maintenance (upgrades) cost, training and support" [Kunda00]. However, the changing of requirements sometimes is not possible and adaptation of the component must be performed. Kwon [Kwon98] identifies the process modelling of maintenance with reuse, he identifies two approaches used for reusing components as:

- **Black Box reuse:** *a component is reused on an "as-it-is" basis.*
- **White Box reuse:** *it should be modified before reuse.*

The case for White Box reuse is strengthened by Select Business Solutions [Selec03a] view that partial reuse may be the most cost effective way of providing new services because in reality, the reused component or service is unlikely to provide a perfect match for the projects needs, and the need for testing of the new application is always essential.

Components can be reused either vertically in a project or horizontally across many projects. Vertical reuse is the reapplying of components in the same project or in the same domain. Horizontal reuse is the application of component in many different projects that may or may not be in the same domain. However, Tracz [Tracz95] and Griss and Wentzel [Griss94] say that software reuse is most effective when the reusable software artefacts are developed for and used in a specific software domain.

It has been recognised that there are several pre-conditions that must be met in order for a developer to be able to incorporate a reusable component into their software system. Frazer [Fraze92] lists these as:

1. The component must exist
2. The component must be available to the developer
3. The developer must be able to find the component
4. Once found, the developer must be able to understand the component.
5. Based on an understanding of the component, the developer must identify the component as being valid for the current system.
6. The developer must be able to successfully integrate the component into the current system.

During this section, a review of how to integrate reuse into an organisation was investigated, but outsourcing of a reuse is a possibility within business. The purchasing of functionality offers opportunities for cost savings, and can be considered to be a transfer of effort and risk from the organisation to a third party, but the opportunities for savings are reduced by the need for the third party to operate at a profit [Selec03a]. The failure to capture, manage and reuse these assets means that critical knowledge about the application and the business processes it supports are lost to the outsourcing organisation – making maintenance, upgrades and integration of these application more difficult [Selec03b].

2.3 Reverse Engineering

Maintenance activities are categorised by Lientz and Swanson [Lien80] as

- **Adaptive:** changes in the software environment
- **Perfective:** new user requirements
- **Corrective:** fixing errors.
- **Preventive:** prevent problems in the future.

Their investigation into the effort spent on maintenance showed that 75% of effort spent was on adaptive and perfective, while error correction consumed 21%. Lehman [Lehma80] gives an insight into the reasons why perfective and corrective maintenance takes up a large portion of maintenance, he states that "documentation for systems is often quite poor and lacks the quality that a maintainer needs to do their task. Over time, memories fade, software engineers leave, documents decay and thus complexity increases as the knowledge of the inner workings of a system slip away from the human domain". Bennett and Rajlich [Benne00] strengthens that argument by stating "that if changes can be anticipated at design time, they can be built in by some form of parameterisation" and that "many changes actually required are those that the original designers cannot even conceive of". It is vital that the transfer of architecture and design tradeoffs, engineering constraints, and the application domain of software engineers are transferred through well-transcribed and accurate documentation to define the architecture of a system and the dependencies between components.

The task of analysing systems in a subjective manner is called reverse engineering. This may include goals such as identifying the system's components and their inter-relationship, or creating representations and design information of a system in another form or at higher levels of abstraction. The primary goal of reverse engineering is the understanding of programming code. This is key when introducing a reuse system where there is potential for reusing already developed components that may not have any documentation associated with them.

The main application for reverse engineering is on legacy systems where an understanding gap arises between known, useful information and the required information needed to enable software change. Reverse engineering tools focus on bridging the understanding gap, and transferring this previously unknown information to the mind of software engineers. It is beneficial to an organisation for it to reverse engineer previously developed components when introducing software reuse. The extracted information is useful for classifying these components within any reuse tool, and gives the opportunity of reusing these previously developed components.

Program understanding plays an essential role during the phases after software development. Henninger reports that "40% of maintenance is spent understanding code" [Henni97]. As such, program understanding is the key activity during software maintenance. To aid a maintainer's task, automated tools or defined standards must be implemented to reduce the size of the task and to make maintenance work more efficient.

Reverse engineering tools provide software engineers the ability to analyse systems at various levels of abstraction and maintain mappings between these levels. The lowest level is the programmer's abstraction which is the identification of semantics via control flow and data flow analysis occurs. However, at these lower levels of abstraction, the big picture behind the evolution of a software system is missed.

For efficient reverse engineering, the tools deployed must be automated to save software engineers the time and effort of studying code. This is especially case when introducing a reuse repository into an organisation, where possibly thousands of previously developed components could be beneficial if introduced into the repository. This however is much harder to gain by the imperfect knowledge these tools have to tolerate. A serious solution of fixing this problem is through continuous application of reverse engineering. This would reconstruct the earliest design and architectural decisions at earlier stages of the lifecycle of a system.

2.3.1 Cognitive Models

Ramalingham et al [Ramal04] describes programming as "a highly cognitive activity that requires the programmer to develop abstract representations of a process in form of logic structures", and highlights that mental models "play an important role in program comprehension and correspondingly in comprehension related tasks, such as modification" [Ramal04]. A mental model is defined by Norman [Norma83] as "predictive representations of real world systems. People create internal representations of objects and information in the world, and use these mental representations to reason about, explain, and predict the behaviour of external systems". These features are of major significance towards reuse. A person who applies reuse to their work needs to gain substantial knowledge of what code actually does to identify whether it is suitable for their needs. Their internal representation is defined by Retkowsky [Retko99] as being a reuser's 'mental model'.

The mental model is defined by Timens [Timen89] as being a list of domains:

- *Task Domain.*
- *Intermediate Domains.*
- *Algorithms.*
- *Plans.*
- *Beacons.*
- *Programming Languages.*
- *Source Code.*

Knowledge about a specific mental model domain consists of information about the objects and the operations within that domain, as well as information between objects and operations of this domain to objects and operations to nearby domains. Brooks [Brook83] and Soloway [Solow84] describe the various domains, and the relationship between them.

2.3.1.1 Brooks's Model of Program Comprehension

Brooks's [Brook83] cognitive model of program comprehension takes a top down approach of mapping between domains. The understander develops a primary hypothesis; this is usually provided from the program name, and forms the root of the tree. A cascade of subsiding hypothesis follows from the basic understanding of the domain knowledge. This has been built from experience in the task domain, and experience from the programming domain. This process is completed via a depth first search. This cascading continues until the understander can verify the hypothesis against the program code and/or documentation.

To aid this process of identifying mappings between domains within code beacons are identified with it. These beacons describe those visible details that show the presence of a particular structure or operation, and provide an important first link between the top down hypothesis and the actual program text. Mittermeir et al [Mitte01] highlights that novices and experts both use beacons in program comprehension.

When scanning the program code, the understander is searching for a set of beacons dealing with the current hypotheses. When a hypothesis has been verified to the satisfaction of the understander than actual bindings between the hypothesis and the program code occurs. If the understander has created the correct primary hypothesis, as well as all the subsiding hypotheses, and is able to bind the program code completely and uniquely to these hypotheses, the understander is said to have comprehended the program completely.

The task of comprehension can vary greatly depending on a number of factors. Primarily, documentation explaining the functionality of the program is the most important. Usually it is rare to obtain documentation explaining these intermediate domains rather than the original program task. This increases the difficulty upon tracing the mappings from the programming level to the problem domain. Secondly, the ability for an understander to identify beacons within code is controlled by the quality of code, the amount of documentation, the individual's abilities, the task they are attempting, and the quality of the primary and higher level hypotheses. This is amplified by the programming domain knowledge of the understander, and affects the lower level bindings and beacon location process.

Since understanders can rarely generate large numbers of alternative hypotheses which have the same behaviour, it is most likely the understander simply repeatedly attempts to interpret and bin program code to existing hypotheses, rather than using know features, or beacons of the program to adopt different hypotheses.

2.3.1.2 Soloway's Model of Program Comprehension

Soloway's [Solow84] approach defines the process of program comprehension as being "the recognising of plans in code, combining these plans (by reversing the rewrite rules) to form sub goals, and combining into higher level goals" [Hoyda91]. This attempts to recover the intention behind the code; therefore, the goal denotes the intention, and plans denote techniques for realising those intentions. This is seen as a bottom-up approach as it maps from the programming domain up to the task domain. Rajlich and Wilde [Rajli02] describe Soloway's strategy as "the programmer piecing together his understanding of the program by combining chunks into increasingly large chunks".

The knowledge base used in the Soloway's model [Solow84] is:

- **Programming language:** *deals with understanders' knowledge.*
- **Goal knowledge**: *the encoding of the understanders' set of meaning for computational goals.*
- **Plan knowledge**: *the encoding of solutions to problems that the understander has solved or understood in the past.*
- **Efficiency knowledge:** *detect inefficiencies.*
- **Domain knowledge:** *understanders' knowledge of the world.*
- **Discourse rules:** *programming conventions attach greater meaning to the same code.*

2.3.1.3 Evaluation of Approaches

Von Mayrhauser and Vans [vonMa94] evaluates Brooks [Brook83] and Soloway [Solow84] models and states that each accommodates the following:

- A mental representation of code
- A body of knowledge stored in long-term memory
- A process for combining the knowledge in long-term memory with new external information into a mental representation.

The bottom-up orientation of the Soloway model [Solow84] is bound to fail. It simply creates too much data for a human can handle. The top-down generation of a human's mental model produced by Brooks [Brook83] ensures that human limitations are incorporated at every level of the understanding at every step of the understanding process ensuring that humans do not feel overwhelmed. This view is shared by Rajlich and Wilde who states that "complete comprehension of the whole program is unnecessary and often impossible" [Rajli02]. They add that "as programs become larger, it has become less feasible to achieve complete comprehension" [Rajli02]. They further add that these models "have been combined into unified models" [Rajli02] to include "an as-needed strategy in which they attempt to understand only how certain specific concepts are reflected in the code" [Rajli02].

2.3.2 Representations

2.3.2.1 Internal Representation

In Section 2.3.1, the cognitive perception of software comprehension was analysed. An analysis was performed on both Brooks [Brook83] and Soloway [Solow84] methods of program comprehension. This provided the different domains within software comprehension and defined how they were mapped and traversed for each method. The goal of which is to define an accurate mental model of software by filling the gaps missing in a programmers internal representation.

Maintainers when observing must gain a 'mental model' of what is happening in code. This involves the analysis of control flow and dataflow within an operation. This is helped through a set of guidelines for programming and documentation practices. Retkowsky lists the following guidelines:

- A class must have a multi-line prologue commentary preceding the class, indicating the purposes and goals of the class.
- Also included are annotations indicating author and version number.
- Program comments within and between modules and procedures usually convey information about how the program achieves the goals set out in the prologue comments.
- Information such as the functionality, any assumptions, declarations, algorithms, and reminder notes can be added 'in-line' to the class.
- A class/module should be responsible for one well-defined process [Retko97].

2.3.2.2 Externalising the Internal Representation

Externalisation is how programs externalise their internal representations via programming and how to describe a program.

When externalising the internal representations of a program, a programmer can describe either a possible solution, or their solution. This proves to be a grey area in software comprehension. Externalising internal representations can be done through a number of different approaches such as natural language keywords, or using tree architecture.

Natural language is used to give components textual descriptions. These are very difficult to develop. These are possibly inaccurate or inefficient due to the possibility of their descriptions containing useless information because of full textual description that programmers possible may add to components.

Keywords can be used to describe components and then can be matched using synonyms or equivalent. This approach requires a limited dictionary of words, and forces the programmer to really think about what he/she wants. Most keyword searches indifferently describe the problem and the solution; therefore, the results produced from this act as a first level filter. Further analysis of components is needed to make a judgement whether one fits the desired task.

2.3.2.3 External Representation

External representation can be in one of three forms, code, textual representation, and graphical representation. However varied these approaches maybe when compared to each other, the resulting mental model should be the same. The accuracy of these external representations can be judged between the mental models obtained, and of the "pseudo code" of the initial developer.

Code is the primitive method of representing a program. The understanding of code is enhanced through defining a structured layout within it. Examples of this are seen by indentations to define blocks of code, or lines in-between functions to break up code so that it is easier to a maintainer to analyse.

A new approach being undertaken by developers of programming environments is to introduce colouring to code so that various keywords are highlighted [Retko97]. Applications introduce colours automatically when displaying these scripts; however, the user does not have the ability to manually manipulate the colour of code. Important sections or significant keywords that a programmer would want to point out to the maintainer as being vital to the understanding process cannot be highlighted by this automation.

2.4 Electronic Data Interchange

As businesses identify the growing advantages of cooperating to streamline costs in this ever competitive environment, technologies such as EDI (Electronic Data Interchange) and Electronic Business using eXtensible Markup Language (ebXML) are becoming ever more desirable for companies to invest in. The application of these technologies is possible when considering the transferring of reusable assets between these organisations.

EDI allows the transfer of information between companies in a format that can be understood clearly and concisely. Before EDI was introduced, the transfer of information between organisations was in a raw format (it is still common practice for small companies to operate in this manner). For this raw data to be processed by the receiving company, it was often the case that manual data entry was needed. This took time and vital human resources; it also opened up the opportunity for errors to occur. EDI therefore allowed the automation of data entry into a system, and removed the possibility of errors and saved resources.

```
<?xml version="1.0"?>
<person>
  <name>Jim</name>
  <age>22</age>
</person>
```

Figure 2.4-1: A sample of an XML document

The growth of EDI saw many new techniques emerge. For EDI to become successful and worthwhile for a business to implement, its business partners must also implement in the same technique. As the web of business partners span worldwide, the need for just one technique was apparent. eXtensible Markup Language (XML) ISO 8879 became the de-facto standard for all EDI over the web. XML is a text-based language that displays data in tags that defines structure within the document (shown in Figure 2.4-1).

2.4.1 ebXML

ebXML is a global business standard that is sponsored by UN/CEFACT (United Nations Centre for Trade Facilitation and Business (Organisation for Advancement of Structural Information Standards) [Irani01]. The goal of this standard is to bring about the integration of small and large businesses into one business environment that enables inter-company processes based on a common protocol. The standards for global electronic business are defined in a framework that is based "upon well-defined XML messages within the context of standard business processes" [Irani01].
The advantages of implementing this framework are:

- *A reduction in the cost of implementation since only one global standard needs to be implemented within a system.*
- *Businesses are not restricted to who their trading partners are. This opens up more competition in the marketplace.*
- *Businesses are integrated more easily due to them implementing the same standards*

2.4.2 How it works

The primary underlying function of ebXML can be split into three abstract categories. These are publishing, finding and binding. Publishing involves giving the ability to companies of disclosing the services they can offer to a potential partner. These are services such as common business transactions e.g. sales ledgers, or profiling their capabilities. These details then could be discovered by other organisations searching a data repository containing these details. During the binding stage, negotiation and transactions are performed. Once a search of the ebXML repository produces results that highlight a number of companies that are valid for the desired business collaboration, a Collaborative Partner Agreement (CPA) business contract is negotiated to agree the terms and conditions of the transaction. On agreement, business transaction can be performed between the two corporations.

The current feedback from industry concerning the use of ebXML within industry is positive. Jennifer Hamilton CEO of RosettaNet highlights the company's drive towards ebXML with "plans to integrate support for the ebXML Messaging Services Specification in future releases of RosettaNet's Implementation Framework" [Web08]. David Russell CTO of Bind Systems is also very positive upon the uptake of ebXML which he sees as "a pivotal component enabling the delivery of 'business ready' Web Services" [Web08].

The growth in the uptake of business collaboration between small and medium sized business using ebXML highlights that the economics of integrating their legacy systems and business processes with current business integration frameworks gives a positive of investment. This is of vital importance in this sector where profit margins are much smaller then larger corporations.

2.5 Web Services

Web service technology encourages the distribution of business processes, such as reuse across physical boundaries, which have prevented processes to be streamlined or accessible to off-site entities. This emerging area has many definitions associated with it that attempts to explain this.

"Web services are not EAI in and of themselves. Rather, Web Services are just another technology that enables EAI, and can significantly change the traditional point-to-point approach" [Samta02].

Dogac et al [Dogac02] gives a particular interesting explanation that gave an insight into the possibility for web services becoming an international standard in distributed computing.

"*A web service is a programmable entity that provides a particular element of functionality, such as application logic and is accessible to any number of potentially disparate system through the use of Internet standards, such as XML and HTTP*" [Dogac02].

Ceremi [Cerem02] provides details into why web services are accessible over the Internet.

"A web service is any service that is available over the Internet, uses a standardised XML messaging system, and is not tied to any one operating system or programming language" [Cerem02].

Cauldwell et al [Cauld01] expands upon the points of accessibility of web services from [Dogac02, Cerem02] by mentioning the structure of coupling within web service application, and the procedures used the infrastructure.

"*Web Services are modular, self describing applications that can be published, located, and invoked from just about anywhere on the Web or local network*" [Cauld01].

Chaudhary et al [Chaud02] extends Caudwell's [Cauld01] point by highlighting the fact that a web service is packaged as a single entity to a network.

"*A web service is a programmable application logically accessible using standard Internet protocols, having a collection of functions that are packaged as a single entity and published to the network for use by other programs*" [Chaud02].

Gottschalk [Gotts00] quite simply describes web services as being "an interface that describes a collection of operators that are network accessible through standardised XML messaging" [Gotts00].

2.5.1 Current Situation

Traditional distributed architectures incorporate relatively brittle coupling between various components in the system. Over the past few years, businesses have interacted using ad hoc approaches that take advantage of the basic Internet infrastructure [Jepso01]. These are sensitive to change, so as the scale, demand, volume, and rate of business change, the brittleness of these systems increase and becomes a crisis [Chaud02]. A number of problems can occur through this crisis such as unresponsive/unavailable websites, lack of speed to market, inability to rapidly shift to new business opportunities or competing against threats.

The high coupling of components in traditional systems ensures that the management of these architectures is virtually impossible. To replace current models of application design, a new generation of distributed applications have been designed to provide an architecture that is more flexible [Chaud02].

In previous years, server based applications such as Common Gateway Interface (CGI) technologies have dominated solutions to reusable libraries, but with the advent of web services many previous solutions to the reuse paradigm are lacking functionality and efficiency that web services can provide [Cerem02].

2.5.2 Challenges with Existing Protocols

The heterogeneous network environments of the web provide a challenge to existing protocols. Distributed technologies such as Common Object Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM) or Remote Method Invocation (RMI) are required to have present symmetrical requirements where "both ends of the communication link are implemented in the same distributed object model" [Cauld01]. This problem is amplified because of these distributed technologies relying upon single vendor solutions; thus generating compatibility problems between different programming languages and operating systems, and always relying upon their protocols being better than competitors. It is this lack of universal acceptance throughout industry, which has encouraged the search for another solution to distributed computing.

2.5.3 Strategies of Integration for B2B Commerce of Reusable Assets

The integration of businesses to form partners among each other has allowed businesses to specialise in certain areas of business processes. This expertise allows companies to focus on their primary objectives/goals, and not to be distracted by menial objectives. It is more desirable for other organisations to be producing higher quality components and integrating these organisations into their own business processes. This strategy opens up the possibility for 'Business-2-Business' (B2B) commerce amongst business partners. Samtani and Sadhwani [Samta02] states that this form of integration among companies can increase growth and success, and that this includes all sizes of organisations. He also states that strengthening relationships between business partners and producing seamless integration can "increase operational efficiencies and reduce costs". This is of relative importance in the current economic environment.

Samtani and Sadhwani [Samta02] defines a number of conventional patterns for integration of B2B commerce that depends upon the trading agreement chosen by trading partners.

2.5.3.1 Portal-Orientated Integration

This approach is highly suited towards small to medium sized companies because of the reduced amount of investment needed. A portal is established by the development of a web application that gives data access to trading partners. However, this approach does not offer seamless integration between businesses, and the delicacies of business processes are not analysed to gain maximum efficiency.

2.5.3.2 Data-Orientated Integration

Data-Orientated Integration involves the sharing of data between two different partners. This is in the form of replicating data sources via synchronous or asynchronous updates, or the merging of data sources into one data warehouse.

Data is a vital commodity in business and has a significant value in specific contexts. The sharing of data between partners can be of a significant advantage but it can also be a threat. To remove this threat, the identification of whether this will create competition has to be analysed.

2.5.3.3 Application-Orientated Integration

Application integration involves a group of organisations working closely together to form software that communicates via RMI or API to each other's software components. This form of integration provides the least amount of automation; however, it does offer synchronous data retrieval and updating.

2.5.3.4 Business Process-Orientated Integration

Ideally for a company to progress in the CMM, business processes must be fully understood and described in a non-ambiguous language [Caput98]. This gives the ability for automation to occur within processes. Business Process-Orientated Integration "provides process interface abstraction that maintains the integrity of business rules" [ORior02]. Integration of this nature gives companies complete autonomy in terms of how they want to conduct their business, although predetermined standards must first be agreed and met by both companies for complete autonomy to occur.

2.5.4 Service-Orientated Architecture (SOA)

For true dynamic integration, software resources such as applications, objects, and programs should be loosely coupled [ORior02]. For integration to occur between businesses, public interfaces of these entities are provided to describe their actions. The presence of these resources and interfaces should be made available to application developers through searching mechanisms that involve sifting for multiple repositories. The successfully undertaking of these actions requires that these resources are to be built to open standards.

SOA provides a framework and architecture "that enables seamlessly interconnecting applications and software components" [ORior02]. The invocation or installation of remote business services into a different application can now be applied without composing a single line of programming code. SOA focuses on how service components are described and organised to support dynamic, automated discovery and use [Flurr01]. To make this possible, SOA has a defined architecture. Samtani and Sadhwani [Samta02] outlines a number of roles and operations within service orientated architecture.

Components

- **Service provider**: *creates and publishes interfaces of services.*
- **Service requester:** *registers and categories published interfaces.*
- **Service requestor:** *an actual user is aiming to discover services by searching a repository storing the published interfaces.*

Operations

- **Publish.**
- **Find.**
- **Bind.**

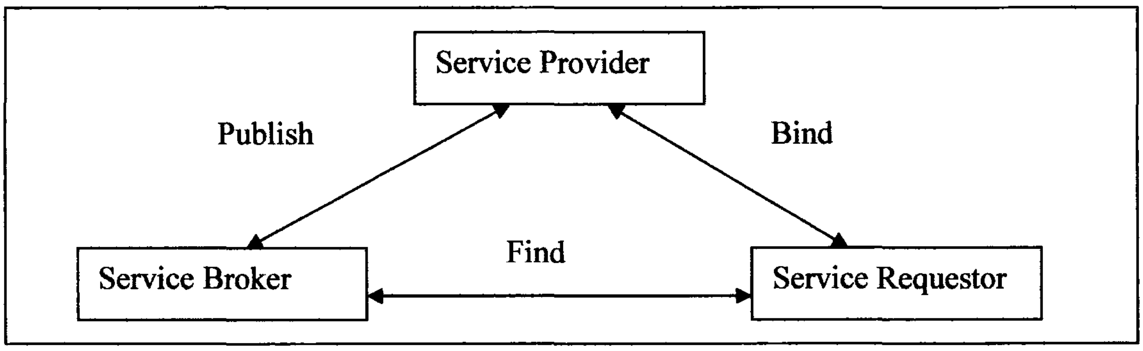


Figure 2.5-1: Service Orientated Architecture

Gottschalk et al [Gotts02] provides additional details to this architecture by outlining a number of objects that operations are performed upon.

Objects

- **Services.**
- **Services descriptions.**

The operations performed within this architecture are done so by actors. These actors are a list of Simple Object Access Protocol (SOAP) service nodes that defines a message path. Each intermediate node can perform some processing before the message is forwarded to the next node [Cerem02].

Chaudhary et al [Chaud02] expands Gottschalk et al's [Gotts02] list of possible operations within the service-orientated architecture to be applicable to web services.

- *Web services are created and interfaces and invocation methods defined.*
- *Web services needs to be published to one or more intranet or Internet repositories for potential users to locate.*
- *Web services needs to be located in order to be invoked by potential users.*
- *Web services needs to be invoked to be any use.*
- *Web services needs to be unpublished when it is no longer available or needed.*

This description provides addition actions such as the creation of interfaces and methods, and the need for authors to remove interface entries from repositories if there is no need for them. Doing this is a form of maintenance and ensures that results from search queries are accurate, and improves the responsiveness of search the repository.

Chaudhary et al [Chaud02] produces summarised explanation following their previous definition of the fundamental concepts of web services. These are:

- *Encapsulation.*
- *Message passing.*
- *Dynamic binding.*
- *Service description and discovery.*

2.5.5 Concepts of Web Services

The concepts behind web services are aimed at reducing complexity through encapsulation; this enables web services to be easily understood. This also represents black-box functionality that can be reused without worrying about how the service is implemented because service requesters do not need to understand underlying implementations when accessing interfaces. Service providers also have no idea of how a service requester uses its service. Encapsulation and the need not to know the underlying implementations promote the easy learning curve of web services [Chaud02].

These fundamental concepts also aid Just-In-Time Integration of web services. Collaborations in web services are bound dynamically at runtime. Dynamic service discovery, invocation and message-orientated collaboration yield applications with looser coupling, enabling just-in-time integration of new applications and services [Chaud02]. Glass [Glass00] points out that these features "yield systems that are self configuring, adaptive and robust with fewer single points of failure".

Web services are composed of three components. These are Simple Object Access Protocol (SOAP), Universal Descriptions Discovery and Integration (UDDI), and Web Services Description Language (WSDL). Each component aids the 'publishing, finding, binding' philosophy highlighted in Figure 2.5-1. Each of these are analysed further on in this section.

Before web services, the vast majority of enterprise scale developments platforms were rather limited with Java Application only being accessible via Java programming language, CORBA applications only being accessible through using the CORBA framework. With web services, an integration channel between the various applications and programming languages is present. This allows methods from different programming languages to be invoked by each other. This compatibility is possible because web services are developed with open standards in mind. Not only are different applications and programming languages possible, but they are also platform independent.

Web services provide a solution to the problems of distributed computing by bridging "the differences that exist between systems that use incongruent component models, operating systems and programming languages." [Chaud02] Cauldwell et al [Cauld01] highlights that the provider or consumer of a web service does "not have to worry about the operating system, language environment, or component model used to create or access the XML Web service, as they are based on ubiquitous and open standards, such as XML, HTTP, and SMTP". The use of HTTP in the transport layer of the infrastructure enables communication to pass through firewalls or proxy servers easily. A system may have severe restrictions upon the accessibility of ports through a firewall with the only ports being accessible being the ones used for HTTP and SMTP communications. This removes any need for processes to open sockets and listen for requests that may be blocked by firewalls or proxy servers.

The contents of the web can be separated into to groups "eyeball web" and "transaction web". A collection of human readable pages that are virtually unintelligible to computer programs are described as being eyeball web while pages that can be interpreted by computer programs are denoted as "transactional web" [Sycar03]. The "eyeball web" is dominated by program-to-user business-to-consumer interactions. Transaction web is mainly involves program-to-program business-to-business interactions. The transformation of eyeball web computer programs to transaction web versions is being fuelled by the program-to program communication model of web services built on Hypertext Transfer Protocol (HTTP), eXtensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL) and UDDI (Universal Description, Discovery, and Integration [Gotts02].

2.5.6 Web Service Programming Stack

Given the Web's intrinsically distributed and heterogeneous nature, communication mechanisms must be platform independent, international, secure, and as lightweight as possible [Jepso01]. Gottschalk et al [Gotts02] provide an insight into how the collection of standardised protocols and APIs used within web services are coherent to Jepson's [Jepso01] requirement, and how applications and users can access and utilise these services. They state "at each layer of the web service programming stack is the standardisation of simple, open protocols and APIs" [Gotts02]. The use of standardised, simple, open protocols and APIs in web service components enables communication between levels of the programming stack; it is also the "key to the ubiquitous deployment of web services architectures, and the ubiquitous deployment of the infrastructure is the key to the network effect of web service adoption" [Gotts02]. This feature ensures that "Web services can be accessed by any language"

[Chaud02], and "accessed by any component mode, running on any operating systems" [Chaud02]. Web services achieve high levels of interoperability compared with previous programming languages [Gotts02].

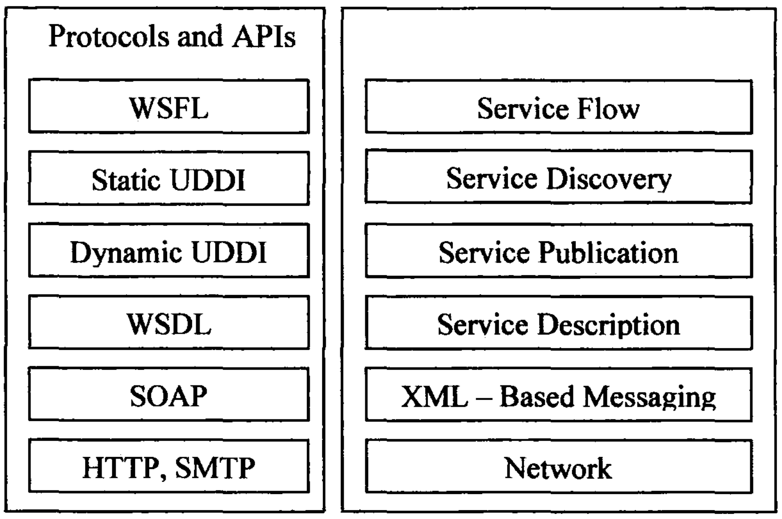


Figure 2.5-2: Web services programming stack

Network, XML based messaging, and service descriptions are needed to have interoperable web services. They also create a low cost entry for leveraging by allowing these services to be deployed over the Internet. The remaining layers in the programming stacks are optional and will be used as businesses need them [Gotts02].

Open standards such as SMTP, HTTP and File Transfer Protocol (FTP) are listed in the network layer of the Web services programming stack (Figure 2.5-2). The use of these protocols enables advantages such as the passage of messages through firewalls and compatibility with other business networks [Chaud02]. However, the reliability of using HTTP is questionable. The use of this protocol does not guarantee the delivery or the order of packets at the destination. Using message queuing can increase reliability but this is at the cost of the response time. To overcome this problem, new protocols such as reliable HTTP (HTTPR), Blocks Extensible Exchange Protocol (BEEP) and Direct Internet Message Encapsulation (DIME) are used.

For the networking layer of the model standards, Gottschalk et al [Gotts02] outlined a number of standards:

- *Share a common networking protocol.*
- *Use a protocol converter to convert between the networking protocols each uses.*

2.5.7 SOAP

The messaging layer of the web services programming stack (Figure 2.5-2) is based on SOAP. SOAP is a standardised packaging protocol for the messages shared by applications [Snell02]. This is an XML protocol, which facilitates the publishing, finding, binding and invoking operations [Gotts02]. SOAP works on existing transports, such as SMTP, HTTP [Jepso01], and ensures that messages are not uniquely tied to just one operating system or programming language [Snell02].

Simple Object Access protocol (SOAP) enables communication among web services. It was initially created by Microsoft and later developed in collaboration with developers IBM, Lotus and UserLand. SOAP is an XML-based protocol for messaging and RPCs. At the basic functionality level, you can use SOAP as a simple messaging protocol [Jepso01]. It can also be used as a method for extending the usage of legacy applications. A SOAP wrapper can enclose a legacy application. This casts the application as a web service. This would allow these dying legacy applications to be used in interesting new ways [Dogac02].

SOAP specification defines a model that dictates how recipients should process the SOAP messages. The message model also includes actors, which indicate who should process the message. Actors indicate a series of intermediaries that process the message parts meant for them and then pass on the rest [Jepso01].

Business messages typically originate deep inside one enterprise and go deep inside another. Additional security is needed such as Secure MIME (S-MIME), HTTP Secure (HTTPS) or Kerberos. Mechanisms such as Secure Socket Layers (SSL) are great for confidentiality of information between two machines in a direct connection. SSL does not operate in an in-direct connection [Dogac02]. To enforce security, SOAP security extensions are being developed that detail specifications for authentication, confidentiality, and authorisation at the SOAP level [Jepso02].

The World Wide Web Consortium (W3C) schema specification provides a standard language for defining the document structure and the XML structures' data types. XML has gained widespread acceptance as a standard specification for data markup, validity checking, and tagging. XML greatly aids in the generation, validation and machine interpretation of complex data structures or documents. SOAP is built on top of XML [Gotts02]. SOAP assumes a type system based on the one in XML schemas and defines it canonical encoding in XML to produce an XML encoding for any type of structured data. XML and SOAP are the base technologies of Web Services architectures [Cauld01].

SOAP offers basic communication, but it does not tell us what messages must be exchanged to successfully interact with a service [Jepso02]. For successful interactions to occur, both the service providers and requester must agree to a common format for the messages [Gotts02]. Messages can fall into one of two categories. Gottschalk et al [Gotts02] describe these methods:

- *Composed primarily of a document that is to be processed remotely.*
- *Contain components and parameters that are used to directly invoke a RPC and return values in XML.*

Gottschalk et al [Gotts02] identifies that is was until recently that there was "no common protocol for handling both types of messages". Applications such as IBM MQ series handled the formatting and delivery of documents within transactions during Electronic Data Interchange (EDI), while Java Remote Method Invocation (RMI) and Distributed Component Object Model (DCOM) is used to format components and parameters. SOAP standardizes both types of messages, document-centric messages and RPC using XML. SOAP implementations exist for several programming languages, including C, Java, and PERL, which automatically generate and process the SOAP specifications; they thus be exchanged by services implemented in different languages [Jepso01]. This greatly enhances the ability of service providers, and users to properly interpret the messages [Gotts02].

SOAP messages have a common structure. An envelope encloses a SOAP header and SOAP body blocks. The SOAP header block has child blocks within it holding information relevant to how the message is to be processed. These blocks control the routing and delivery settings, authentication or authorisation assertions, and transaction contexts. The blocks within the SOAP body block contains the actual message that is to be delivered and processed [Snell02].

Dogac et al [Dogac02] identified that the SOAP performance is degraded within its use inside the web services architecture because of a number of reasons:

- *SOAP uses XML instead of binary data that makes the size of the data almost 400% larger.*
- *Extracting the SOAP envelope from the SOAP packet is time expensive*
- *Encoding binary data in a form acceptable to XML is time expensive*
- *XML parsers support a number of features that makes them resource intensive. Not all of these features may be used by SOAP.*

Chaudhary et al [Chaud02] provides a number of solutions for this performance issue:

- *Some applications can consider compressing XML when CPU overhead required for compression is less than the network latency*
- *Consider using stripped down versions of XML parsers*
- *Current SOAP implementations are Document Object Model (DOM) based*
- *DOM parsers are slow to parse messages.*

- *Simple API for XML (SAX) based SOAP implementations can be used to increase through put and reduce memory overhead.*

2.5.8 WSDL

Within the web service programming stack (Figure 2.5-2), the ability for describing services to clients is provided by the WSDL specification. SOAP offers basic communication, but it does not tell us what messages must be exchanged to successfully interact with a service [Jepso01]. A web service is described using a standard, formal XML notation called its service description that provides all of the details necessary to interact with the service; this includes message formats, transport protocols, and location. Standardization of service descriptions to support web services is achieved via Web Services Description Language (WSDL) [Gotts02]. WSDL was developed by IBM and Microsoft to describe web services using a common semantic understanding of the context of these messages [Gotts02]. This forms collections of communication end points that can exchange certain messages [Jepso01].

For a web service to be invocated, a potential requester must know what services are available from a service provider. WSDL forms an integral component of the discovery process, by providing a formal, computer readable description of web services [Snell02]. These descriptions enable the ability to use inputs to dynamically invocation proxy's which can generate the correct service requests at runtime. This relieves the user and developer of the need to remember or understand all the details of service access [Jepso01]. This also reduces the need of maintenance within SOAP clients if changes to web service are made. The dynamic discovery of WSDL descriptions within a repository ensures that links between SOAP clients and web services are not lost and expensive changes to client code is not needed [Snell02].

A WSDL document describes a web service's interface and provides users with a point of contact [Jepso01]. This defines the interface required for interaction between a requester and a service provider and identifies the location of the service provider [Gotts02]. It also defines a service's abstract description in terms of messages exchanged in a client service interaction. WSDL is the de-facto standard for providing these descriptions. Other, less popular, approaches include the use of the W3C's Resource Description Framework (RDF) and the DARPA Agent Markup Language (DAML), both of which provide a much richer (but far more complex) capability of describing web services than WSDL [Snell02].

Jepson [Jepso01] describes that a complete WSDL service description provides two pieces of information:

- *An application level service description (abstract interface).*
- *Specific protocol-dependent details that users must follow to access the service at concrete service end points.*

Jepson [Jepso01] identifies that there are three main components of an abstract interface within WSDL service descriptions:

- *Vocabulary.*

- *Message.*
- *Interaction.*

Jepson [Jepso01] determines that "the agreement on a vocabulary is the foundation of any type of communication". WSDL uses external type systems to provide data-type definition for the information exchange. It can support any type system, most services use XSD. This is possible because XML schemas are platform neutral [Jepso01].

"*XSD: xml schemas are an application of XML used to express the structure of XML documents*" [Snell02].

WSDL defines message elements as aggregation of parts, each of which is described by XSD types or elements from a predefined vocabulary. Messages provide an abstract, typed data definition sent to and from the services [Jepso01]. The sequence or possible patterns of message exchange between services and invokers are clearly defined within this section of the WSDL interface.

An interaction is simply a combination of messages labelled as input, output, or fault to indicate what part a particular message plays in the interaction [Jepso01].

Gottschalk et al [Gotts02] makes a point of indicating that this area of web services has not been standardized yet, but highlights the importance of doing this to ensure interoperability among web service repositories and web services for the future. This accounts for the fact that similar application-level service functionality is often deployed at different end points with slightly different access protocol details, this helps WDSL represent common functionality between seemingly different end points [Jepso01].

Snell et al [Snell02] provides an in-depth analysis of the structure of WSDL by categorising five sections to an abstract interface instead of Jepson's [Jepso01] three point definition. Snell et al [Snell02] categorises these as:

- **Data types:** *both parties involved must agree upon the data types being used before the services description can be analysed.*
- **Messages:** *defines the sequence or possible patterns of message exchanged between service and invoker.*
- **Interfaces:** *identify the port types. A port element describes a single endpoint as a combination of a binding and a network address. Service elements group a set of related ports.*
- **Binding:** *defines the methods of how messages will be transmitted over the network, and includes the communication protocol (such as SOAP over HTTP), and data format specification that is being transmitted. It also describes how to accomplish individual service interaction over this protocol, and where to terminate communications (net address)*
- **Services:** *lists the network location of the service.*

2.5.9 UDDI

The discovery and publication of a service within the web services programming stack (Figure 2.5-2) is provided by Universal, Description, Discovery and Integration (UDDI) directory. UDDI is a registry of web services descriptors. UDDI specifications offer users a unified and systematic way to find service providers through a centralised registry of services. This is roughly equivalent to an automated online "phone directory" of web services [Jepso01].

A service provider must first register the service with a registry; this enables a service requester to discover the service using UDDI [Gotts02]. Jepson [Jepso01] describes how UDDI provides two basic specifications that define a service registry structure and operation:

- *A definition of the information to provide about each service and how to encode it.*
- *A query and update API for the registry that describes how this information can be access and updated.*

Jepson [Jepso01] explains how access to the UDDI registry is accomplished using "a standard SOAP API for both querying and updating". A Service provider first registers the required technical specifications, and then assigns it to a unique identifier key so that encoding is possible. Cauldwell et al [Cauld01] describes how they also define the three different types of encoding that are used within UDDI.

- **White pages:** *name and contact details.*
- **Yellow pages:** *categorization based on a business.*
- **Green pages:** *technical data about the services.*

The UDDI registry is organized around two fundamental entities that describe businesses and the services they provide. Both business and service entities can specify a 'categoryBag' to categorize the business or service [Jepso01].

*"***CategoryBag element:** *A list of name-value pairs that tag the business entity with specific classification information. This could be in the form of industry taxonomy or geographical classifiers"* [Cauld01].

The design of a UDDI entry is described [Web02]. Jepson [Jepso01] describes the number of key elements in it:

- *Unique keys identify each data entity: businesses, services.*
- *Long hexadecimal strings generated when the entity is registered.*
- *The keys are guaranteed to be universally unique identifiers (UUID).*
- *Business key attribute maps to business entity.*
- *Service key attribute maps to service entity and business key attribute.*

UDDI enables business and service descriptions using arbitrary external information (not defined by UDDI) to find the expected unique key (binding key) and a cross-reference to the service key. Replacing the information itself with a unique key provides a reference to arbitrary information types. The location types of businesses and services depend on the ability to qualify the directory business and service entities. In addition to the three kinds of data published within a UDDI registry, the standards bodies and businesses also register information about their service types. Cauldwell et al [Cauld01] calls this Service Type Registration; however, in the UDDI white paper these are more commonly known as tModels [Web02]. Jepson [Jepso01] takes a different approach into the describing of tModels. He states that "tModel mechanisms are 'simple and powerful'. To adequately describe a service, there is often a need to reference information whose type or format cannot be anticipated. Users and implementers of compliant services must be aware of the registered tModels and their keys".

Human readable description, name, and categorization, the service entity contains a list of binding templates that encode the technical service access information. Jepson [Jepso01] details the actions of these binding templates as:

- *Representation of an access point to the service.*
- *Same service can be provided at different end points (might have different end points, this will have different technical characteristics).*

Service endpoints that support the specification can then imply the addition of the corresponding reference to their tModelInstanceDetails list. tModelInstanceDetails provides the services technical description (green pages) [Jepso01]. This field will then contain a list of references to the technical specifications with which the service compiles.

When a service requester identifies a suitable service via the UDDI interface, the interface provides the service requestor with a WSDL interface and an URL pointing the requestor to the service itself [Gotts02]. The WSDL descriptions outlines how exactly invoker methods can interact with it using SOAP messaging and SOAP RPC calls [Jepso01]. The service requestor may then use this information to the service and invoke it [Gotts02]. SOAP API is used for both querying and updating [Jepso01].

A standard is needed for publishing and finding web services. UDDI emerged to help this problem. This provides sets of APIs for publishing and finding services. A service provider creates a web service and its service definition, publishes the service with a service registry based on UDDI [Gotts02]. This point has already specified business context descriptions for services specified by UDDI categorizing information on the type of business, location, and contact info. This facilitates further discovery and usage of appropriate services, however Gottschalk et al [Gotts02] identifies that "further standardization is needed".

A solution provided by Gottschalk et al [Gotts02] advises "businesses to provide standard APIs so that partners can publish and find services". Gottschalk et al [Gotts02] also identifies that "for a company to create their own APIs for finding and publishing services is a costly venture. It would also create a burden on the merchants and suppliers who are dealing with multiple service provides. These partners would have to customize their applications to work with each service provider". Jepson [Jepso01] explains that the development of web services is not only effective in inter-organizational dealings but also within business processes that should not be accessible outside the organization. They highlight that "several individual companies and industry groups are also starting to use "private" UDDI directories to integrate and streamline access to their internal services" [Jepso01]. These maintain "private UDDI registries and control what service data is registered and who can access data" [Gotts02]. A public UDDI registry is located at http://www.uddi.org. This registry is synchronized and maintained by IBM and Microsoft.

2.5.10 WSFL

IBM has produced the web service flow language (WSFL) as it input in the standardisation process [Gotts02]. Service flow layer of the stack facilitates the composition of web services into workflow and the representation of this aggregation of web services as a high-level web service [Jepso01].

2.5.11 Advantages of Web Services

Web services are emerging to provide a systematic and extensible framework for application-to-application interaction built on top of existing web protocols and based on open XML standards. Jepson [Jepso01] divides the web services framework into three different areas:

- *Communication protocols.*
- *Service descriptors.*
- *Service discovery.*

The Web services framework is modular. The advantage of modularity for developers with this framework is to gain from the availability of specifications and tools now and incorporate more modules as the technology matures [Jepso01]. However these reasons are not the only reasons why the uptake of Web services is growing, Gottschalk et al [Gotts02] highlights a number of reasons why:

- *Web services provide a language-neutral, environment-neutral programming model that accelerates application integration inside and outside an enterprise.*
- *Web services yields flexible loosely coupled business systems.*
- *Web services are easily applied as a wrapper technology around existing applications.*
- *New solutions can be deployed quickly.*
- *Pool of services is growing due to the increase in the uptake of web services within industry.*
- *Aids development of more dynamic models such "Just-in-time applications" and business integration over the web.*

With these reasons in mind, Chaudhary et al [Chaud02] predicts Web services technology to be "both an evolutionary and revolutionary step forward in the domain of distributed computing". They state that this it is evolutionary because "the next step in abstraction is beyond object orientated technology" [Chaud02], while the term revolutionary is used to indicate the "catalytic effect web services have upon the shift away from traditional client-server architectures to peer-to-peer architectures"

[Chaud02]. This has been achieved by "combining the best aspects of component-based development and object orientated approach, geared towards the architecture, design, implementation and deployment of e-business solutions" [Chaud02].

2.5.12 .NET Framework

Web services are components that facilitate the sharing of data and functionality through heterogeneous protocols. This is achieved by using open standards such as XML, SOAP and HTTP. The .NET framework provides a developer with the ability of developing these web services. [Web04] lists these key benefits and goals of using the .NET framework and are listed below.

- *Shared code and increased efficiency.*
- *Robust code.*
- *Secure execution.*
- *Support for encryption.*
- *Automatic deployment.*
- *Rapid application development that requires fast time to market.*
- *Ability to call Win32 DLL (direct link libraries) without having to rewrite them.*
- *Debugging and development can be used by Microsoft Visual Studio .NET 2003.*
- *Code is not prone to fail due to un-initialised variables.*
- *JIT compilation is not interpreted.*

- *Garbage collection greatly minimises memory leaks by cleaning up objects no longer in use.*

The .NET framework consists of two major areas of focus; these are common language runtime, and a unified set of class libraries.

The common language runtime (CLR) allows the developer the flexibility to develop in one of many languages that are classified as 'Managed Code'. These languages exhibit features such as strong type-safety, no bad pointers or create memory leaks. Languages that conform to theses features are C#. NET, VB.NET, C++. NET, J#. NET, and ASP.NET. The CLR is responsible for managing the execution of managed code. The first stage of compiling managed code involves parsing it into MSIL (Microsoft Intermediate Language) and metadata, and packaging both languages into a Pre Execution file (PE). The JIT compiles the PE down to a native code when it is being requested. The result of this is that all .NET framework components run as native code and increase the performance of a service.

2.5.13 XML

Extensible Markup Language (XML) provides a way to describe structured data. Unlike HTML tags, which are primarily used to control the display and appearance of data, XML tags are used to define the structure and data types of the data itself. XML uses a set of tags to delineate elements of data. Each element encapsulates a piece of data that may be very complicated or very simple.

As XML tags are adopted throughout an organization and across organizations, data from all kinds of different data stores will be easier to exchange and manipulate. XML is simple, platform-independent, and a widely adopted standard. The advantage of XML over HTML is that it separates the user interface from the structured data. This separation of data from presentation enables the integration of data from multiple code repositories. These can be sited off site and accessed via web services.

To maintain constancy between the XML tags adopted by the different organisations to define their data sources, XSD scripts are applied to XML datasets.

2.5.14 XSD

The XML Schema Definition language (XSD) allows constraints to be specified on the elements and attributes it defines. When mapping on XML schema to relational schema in a Dataset, XML schema constraints are mapped to appropriate relational constraints on the tables and columns within the dataset.

The Microsoft's developer's network has defined a number of advantages that XML Schemas have over previous technologies, such as Document Type Definitions (DTD):

- *XML Schemas use XML syntax, so there is no need to learn a new syntax to define your data structure.*
- *XML Schemas support reusable types and allow the creation of new types using inheritance.*

- *XML Schemas allows the grouping of elements to control the recurrence of elements and attributes* [Web06].

XML Schema guarantees consistency among certain types of XML data that is shared between applications and organizations. XML schemas are used within the ReSULT architecture to verify the structure of the information being passed to it. These XML schemas could be published over the web to promote software reuse between two different sites or organisations. By publishing theses schemas, organisations that wish to exchange data can then build their applications around these schemas so their xml messages will be understood. This is of great importance when considering that the ReSULT is a distributed system that has the opportunity of interacting with other systems during business-to-business commerce.

2.5.15 XSLT

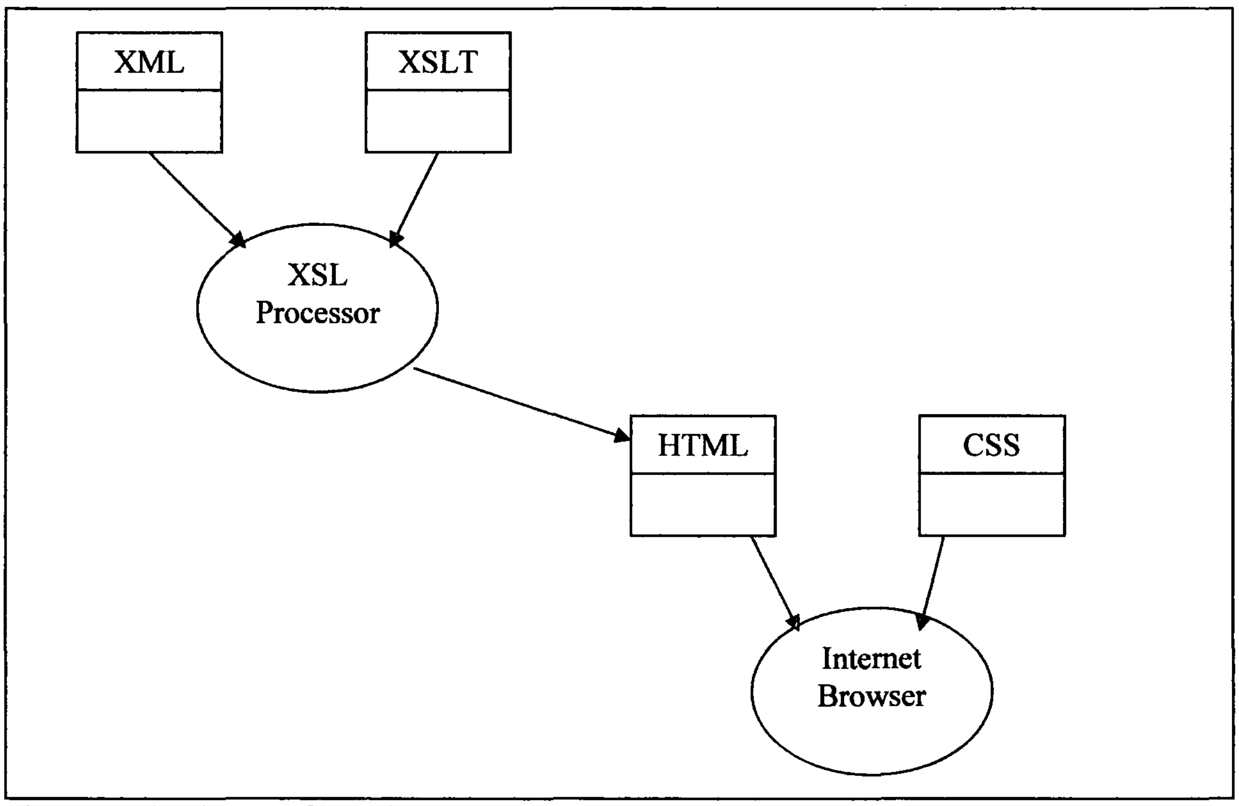


Figure 2.5-3: The transformation of XML to HTML.

As described in Section 2.5.13, the data held within XML data structures does not contain any specific information. The process of converting raw XML data into HTML is displayed in Figure 2.5-3. The process of formatting XML data is achieved through a XSL processor. This takes the XML data and applies an XSLT transformation to the data. An XSL processor is embedded within Microsoft IIS. The output of this is plain HTML with styling applied from the XSLT transformation, the application of any generic styling cannot be applied during the XSLT processing. This can only be applied at the internet browser. Generic styles are applied in the ReSULT system using Cascading Style Sheets (CSS).

2.5.15.1 CSS

CSS provide additional formatting at the end of the process for converting raw XML data into presentable HTML (Figure 2.5-3). When a request is made by an Internet browser for a web page that contains an externally linked CSS script, the web server extracts the class featured in the HTML document and replaces these declarations with their fully declaration stored inside the CSS script. This provides the advantage of avoiding the repetition of HTML elements and tags, thus saving the developer time and effort when developing.

2.6 Design Patterns

Larman defines that "in object orientated design, a pattern is a named description of a problem and solution that can be applied to new contexts; ideally, a pattern advises us on how to apply its solution in varying circumstances and considers the forces and trade offs" [Larma05].

Gamma et al describes a design pattern as "One person's pattern is another person's primitive building block" [Gamma95].

They elaborate further on the definition of design patterns as "a description of communicating objects and classes that are customised to solve a general design problem in a particular context" [Gamma95].

Design patterns enforce users to program to an interface, not an implementation [Larma05]. Many patterns guide the assignment of responsibilities to objects, and often are loosely coupled; therefore, a client component is unaware of the specific object types they use, and are unaware of the classes that implement these. Larman states that "a good pattern is a named and well-known problem/solution pair that can be applied in new contexts with advice on how to apply it in novel situations and discussion of its trade offs" [Larma05]. There are a large number of design patterns available to designers and maintainers. Gamma et al describes the design and application for twenty three different design patterns in [Gamma95].

The ability to control how a system evolves during its lifetime is another key advantage of design patterns. Design patterns allows requirements to be anticipated and for changes to be made significant redesigning of the system to be performed. [Larma05] defines a number of different causes for redesign and how these changes can be performed using design patterns.

Creating an object by specifying a class explicitly: this commits to an implementation and not an interface. Factory Method design pattern [Larma05] would be used to allow objects to be created indirectly instead.

Dependence on specific operations: this commits to a way of handling a request. To modify this execute path a command design pattern would be used.

Dependence on platform: components may be reliant upon on APIs that differ between platforms. The abstract factory design allows flexibility on design on a system that has to interface with different APIs.

Dependence on object representation or implementation: clients that know how an object is represented, stored, located or implemented may need to change when the object changes. The proxy design pattern provides a surrogate or placeholder for another object to control access to it.

Algorithmic dependencies: objects that depend on an algorithm may need to change if the algorithm changes. Ideally, algorithms should be isolated using a design pattern such as Iterator or Visitor.

Tight coupling: inter-class dependency causes problems when maintaining code. To reduce problems the Command or Observer design pattern is used.

Inability to alter classes conveniently: it may be difficult to modify a class because the source code is unavailable, or it may require modifying sub classes. A Decorator or Visitor design pattern should be applied [Larma05].

2.7 Summary

In this chapter, background topics relating to software engineering and distributed computing were discussed. The software crisis was the main motivation for which the idea of reusing software was born. The benefits which can arise from reusing components were also discussed. After that, an insight of how reuse can be introduced into a company was performed. Within this three important roles were identified as being influencer/consultant, producer/business, and librarian/broker. Methods of how reusable components can be integrated into existing systems were identified as adapt components to fit requirements, and change requirements so that the component fits as it is. The scope at which these components could be used within an organisation was identified as horizontal and vertical reuse.

The following section introduced reverse engineering. This provides the means of replacing gaps of understanding within legacy systems. Automated reverse engineering tools are best to reduce the amount of time and effort spent analysing code. Following on from this, two approaches were described that are used for modelling program comprehension when analysing code; these are Soloway and Brooks. There are three different types of representation for the understanding of program comprehension. They were internal, external, and externalising the internal representation. After that an introduction into the concepts and workings of EDI were also discussed.

A discussion of the concepts and advantages web services bring to a distributed organisation and the challenges they face followed. A number of strategies are described into the integration of Business-2-Business commerce of reusable assets; these are portal-orientated, data-orientated, application-orientated, and business process-orientated integration. After that, an analysis of service orientated architecture follows introducing the various components involved with web services such as SOAP, WSDL, UDDI, WSFL, XML, .NET, XSD, XSLT and CSS. Finally, design patterns were introduced giving the reasons why they are useful when designing or applying changes to a system. The following section will take the findings from this chapter, and produce a design proposal for such a system.

Chapter 3 Design

3.1 Introduction

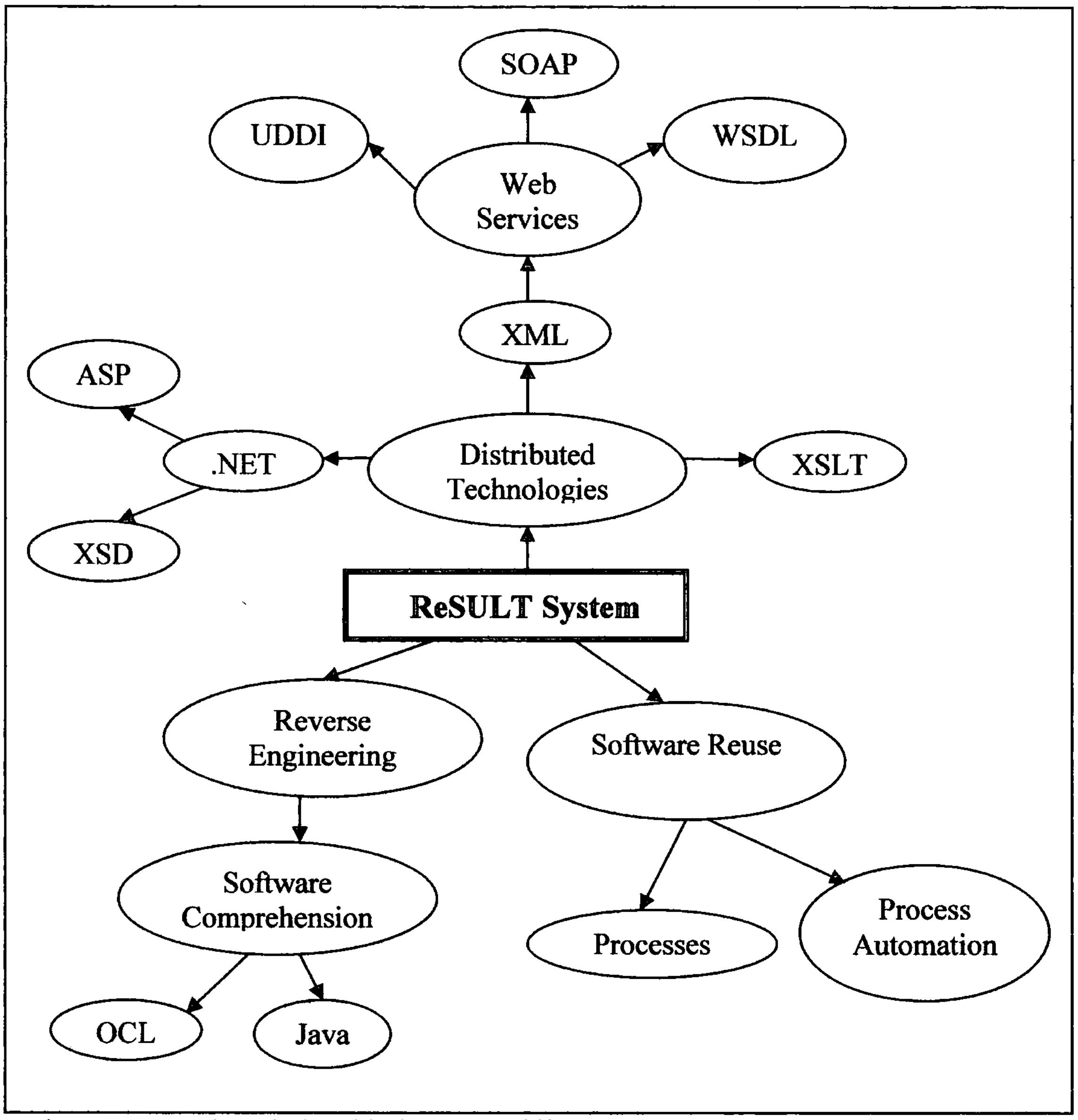


Figure 3.1-1: The relationship between different research areas that are considered during this work.

The literature survey in Chapter 2 highlighted that there are clear advantages to companies implementing software reuse strategies. Furthermore, their strategies have differed over time based on changes that have occurred within organisations. Organisations are frequently located on many sites and any strategy and support tools must support the distributed nature of the organisation. This chapter will describe a support tool which reflects these changes within organisations while supporting a reuse process.

This chapter proposes a distributed solution to software reuse. The proposed system is called Reusable Source code Units Library Tool (ReSULT). ReSULT is an electronic reusable library that provides a distributed organisation with the ability for the sharing of entities; therefore enabling the spreading of knowledge within it. This tool is Internet based to promote the distribution of information across company and nation boundaries. To achieve this goal of developing this system, a number of different areas were considered and reviewed during Chapter 2.

From the different research areas reviewed during Chapter 2, a map that defines the relationships between these areas is shown in Figure 3.1-1. This figure provides an abstract representation of this chapter.

The first section in this chapter is reverse engineering. Taking the findings from the analysis of program comprehension during Section 2.3.1, the software and design languages of Java and OCL are used to apply these findings. The core of this work is to identify useful beacons within these languages that will maximise the understanding of code in either language. The conclusion of this subsection will highlight how all this data is to be stored in an efficient manner inside a database.

Section 3.3 proposes how this system is integrated with the current techniques and processes of software developers.

For the rest of this chapter, an increasing focus on practical issues will be taken towards this research.

An important feature to any system is usability. In this chapter, an insight is provided in how web service technology can be transcribed into a usable feature of a software development workplace. This will include how XML data that is produced from web services is interpreted by web applications. Web applications have many methods of dealing with XML. Features such as Extensible Stylesheet Language Transformation (XSLT) and Cascading Style Sheets (CSS) are approaches used by web applications to transform XML data into organised information that can be transmitted to a user's web browser, using HTML. These technologies mentioned above will provide the majority of the work in this chapter.

3.2 Reverse Engineering

Within the development of any software reuse system, reverse engineering is of significant importance. As described in Section 2.3.1, a reuser needs to fully understand the functionality of a component to identify whether it satisfies their 'mental model'. The goal of reverse engineering during software reuse is to obtain an accurate representation of a component. This representation must have the ability to become externalised to provide a method for reusers to search a collection of components.

When developing a reuse system, an analysis of the current processes involved within it must be considered as possible areas for automation. Processes such as the inserting, searching and extraction of components are likely candidates for automation with a reuse system. By providing the ability of automation within the system, the time taken to complete tasks is reduced and the degree of accuracy achieved is significantly increased by removing the opportunity for human error.

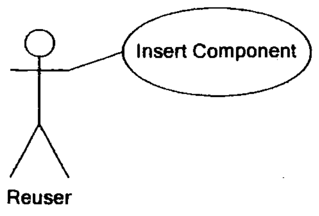


Figure 3.2-1: The 'Insert Component' use-case.

During this research, a use-case (Figure 3.2-1) will be used to identify individual automated processes involved within the system. The first use-case designed for the ReSULT will give the user the ability to insert components into the system. The components will be in Java or OCL. The user is classified as a producer of reusable assets (Section 2.2.2), or possibly a librarian (Section 2.2.2) who is given these assets from a producer. The following sub-sections look into how the system deals with these languages and describes the rationale for comprehension.

3.2.1 Software Comprehension

To achieve any degree of automation, it is first necessary to create a model of what understanding is all about. In Section 2.3.1 of the literature survey, two models of describing program comprehension were analysed; these models were Brooks's [Brook83] and Soloway's [Solow84]. It was concluded in Section 2.3.1.3 that the bottom-up orientation of the Soloway model is bound to fail because it creates too much data for a human to handle. The top-down orientation of Brooks [Brook83] incorporates these human limitations at every step of the understanding process, and will be used for the ReSULT system.

3.2.1.1 Internal Representation

For the ReSULT reuse library, the analysis of comments and program structure will provide the key areas of research. These factors will help the reuser build up a satisfactory mental model for a piece of software. By generating this model, the reuser can then start to map between the domains from the top downwards to identify whether this is ideal to their desired usage.

```
package Organisation;

import java.util.*;

public class Company{

	private int numberOfEmployees=0;
	private Person manager;
	private TreeSet employees=new TreeSet();
	private List topTenEmployees=new ArrayList();
	private Person[] topTwentyEmployees=new Person[20];
	private Company(String description, Person manager)
	{
		super(description);
		this.manager=manager;
		employees.add(manager);
		manager.employers.add(this);
		numberOfEmployees=employees.size();
		topTenEmployees.add(manager);
		topTwentyEmployees[0]=manager;
	}

	public Person getOldestEmployee()
	{
		return null;
	}

	public int getOldestEmployeeAge()
	{
		return 0;
	}

	public void employ(Person p)
	{
		employees.add(p);
		p.employers.add(this);
		numberOfEmployees=employees.size();
	}

	public boolean assertTrue()
	{
		return true;
	}
}
```

Figure 3.2-2: An example of Java code.

In the ReSULT reuse system, it was decided to define a reuse system for the scripting languages Java and Object Constraint Language (OCL). The languages chosen for ReSULT reflect the possibilities of identifying traceability between the design and implementation stages of development. Java was chosen due to a large repository of code available for testing. OCL is a scripting language that adds semantic details to UML structured models that cannot express statements, which should be part of a thorough specification. These statements should migrate through to implementation code, and provide a method of traceability and reusability.

```
@invariant numberOfEmployees:
Self.numberOfEmployees=employees->size

@invariant manager_is_employee:
@element-type Person
employees->iterate(
p:Person ; b:Bag(Person)=Bag{} |
b->including(p)
b->includes(manager)
)
@invariant manager_is_employee2:
manager.employers->includes(self)
@invariant manager.oclIsKindOf(Person)

@invariant topTenTwenty:
topTenEmployees->first=topTwentyEmployees->first
```

Figure 3.2-3: An example of Object Constraint Language (OCL).

OCL and Java are situated in different areas of the software lifecycle; OCL is situated in the design while Java is found during the implementation stage. Traceability between related documents can be reinforced between related documents to promote reuse throughout the software lifecycle. This approach was taken when designing and implementing the ReSULT reuse library to aid reuse within an organisation by providing a continuous application of reverse engineering to reconstruct the early design decisions in the lifecycle of a system.

- *Class name*
- *Package name*
- *Imports*
- *Interfaces*
- *Methods (including parameters and return types)*
- *Fields*

Figure 3.2-4: A list of similar structures identified between OCL and Java scripts.

Examining the code transcriptions of Java (Figure 3.2-2) and OCL (Figure 3.2-3) brings about the identification of similar elements displayed in both. This commonality aids the development of the automated process for analysing components. Figure 3.2-4 displays the information that is held in both languages. It is noticeable that Java contains more structures within its code than OCL; examples are the use of keywords static, abstract etc. These details are also included into the system, so that this increase in the levels of information will provide a greater success rate for the ReSULT system, by providing the ability for a reuser to gain a greater understanding of a component. The inclusion of class and package names gives the reuser an initial indication of what exactly the code does. This acts as a first level of understanding before the reuser continues to observe possible interactions with other classes via interfaces and import declarations. Specific information about functionality is identified by analysing methods and the annotations that lie within methods.

These factors help the reuser build up a satisfactory mental model of a piece of software. By generating this mental model, the reuser can then start to map between the individual software comprehension domains (Section 2.3.1), starting from the top downwards to identify whether this is ideal to their desired usage.

3.2.2 Developing the 'Insert Component' Use-case

The goal of the use case 'insert component' is to place a component into the reuse library. The ReSULT functionality will process this component, identify important features of this code, and place these details with the component itself into a database.

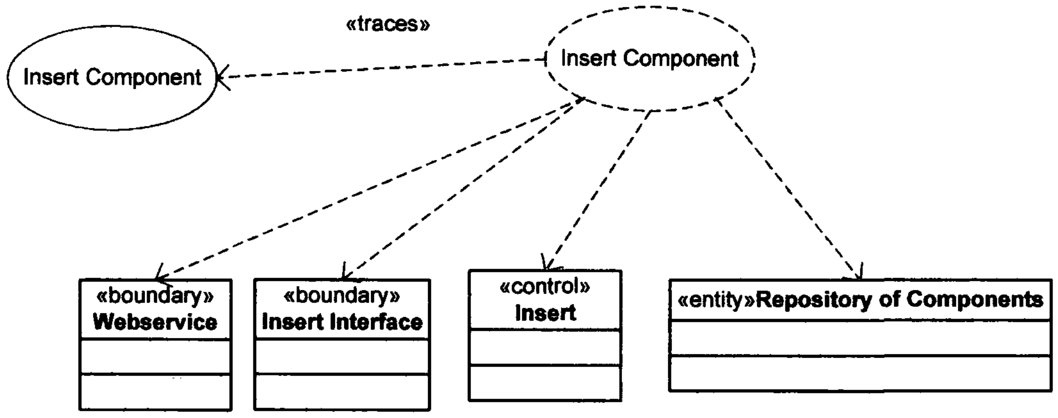


Figure 3.2-5: A diagram displaying the trace between the use case and the analysis classes.

Figure 3.2-5 displays a number of roles that are needed to fulfil the use-case.

Jacobson et al [Jacob99] describes how these roles fall into three categories. These are:

- **Boundary class:** *acts as an entry point for an interaction.*
- **Control class:** *coordinates interactions between boundaries and entities.*
- **Entity class:** *storage of state.*

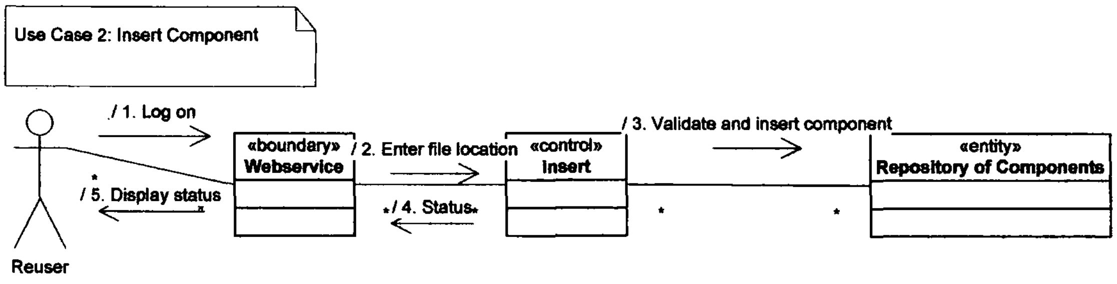


Figure 3.2-6: Collaboration between analysis classes.

How the classes in Figure 3.2-5 collaborate is displayed in Figure 3.2-6. Each class must fulfil all its collaboration roles. A collaboration role describes the type of object that may play the role and describes its relationships to other roles.

If a class is changed, the developer of the class must verify that the class can still fulfil its roles in use-case realisation. If a role in a use-case realisation is changed, the use-case developer must convey the change to the class developer. The roles thus help both the developer of the classes, and the developers of use-cases, to maintain the integrity of the analysis [Jacob99].

Analysis classes when designed give rise to more refined design classes that are adapted to the implementation environment.

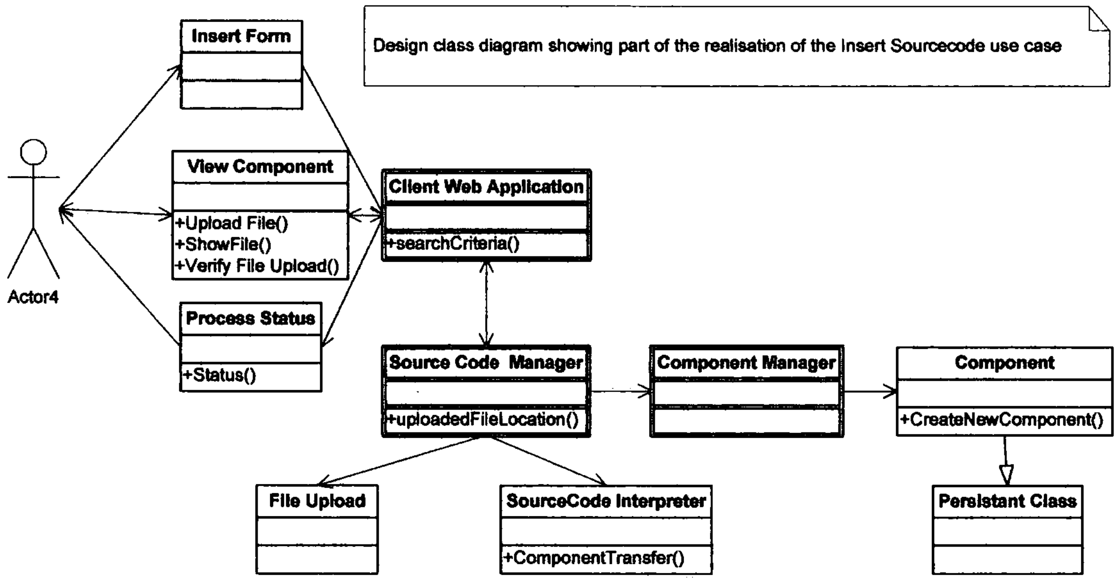


Figure 3.2-7: A design model showing the interactions between design classes in the 'Insert Code' use-case.

Within Figure 3.2-7, the 'Source Code Manager' coordinates the actions beneath the web service. Once the details of the uploaded file are passed to it from the web service, this component invokes the 'File Upload' to upload the binary from the client's specified location. The binary content of the file is passed towards the source code interpreter. For each language catered for within the ReSULT, a different implementation class is used to analyse the different language formats. The outcome of this analysis is the formation of a generic 'Sourcecode' class. This is constructed from a list of similar structures identified between the OCL and Java in Figure 3.2-4. The component manager negotiates the interactions with the database. For this application, this class inserts the component, the abstract representation, and the keywords that are associated with this component into the persistent class.

3.3 Software Reuse Techniques and Processes

The introduction of any new system is based upon the fact that it will do the job of the current system and more. It would not be beneficial for a system to be replaced by one that just replicated its current features, or does not improve dependability, or reduce the time taken for an operation to be executed. To provide the additional functionality, changes to current processes are needed to allow for the added features. This section will look towards the processes currently used within software reuse, and how they will be integrated with the current practices of software developers.

3.3.1 Techniques

3.3.1.1 Internal Memory Reuse Techniques

A programmer develops their own approaches to reuse. One factor that inhibits the success of systematic software reuse is the problem of 'no attempt to reuse' [Yunwe02]. This involves developers constructing new systems from scratch rather than reusing existing software components from a reuse repository. Fisher categorises this into two cognitive difficulties; firstly, developers are unaware of the existence of reusable components, and secondly there is a lack of means to locate the wanted components [Fishe87]. Further studies from Rosenbaum and DuCastel [Rosen95] conclude that most software developers only anticipate the existence of a limited portion of components within a repository, and that they are would not actively seek the reuse of components whose existence that did not know.

The CodeBroker system [Yunwe02] attempts to amend these difficulties by offering reusable components to a developer to import whenever a prologue comment is inserted into the editing space. This comment is parsed into a query for matching against the reuse repository. The output from this was a selection of components; however, Yunwen acknowledges that this was prone to identifying irrelevant components to a developer [Yunwe02].

Without a developed integrated support tool such as CodeBroker [Yunwe02] that users are familiar with and trained to use, programmers have their own interpretations of how to reuse source code. There are a number of approaches outlined in [Retko97] that document how a programmer thinks about reuse. These are 'Write/Copy/Paste', code scavenging, and design scavenging.

A reuser at an early stage may identify similarities in functionality at different stages of a program. It is at this stage that the reuser prepares a generic section of code that may be copied to other locations in the code with the possibility of slight alterations to adapt to differences in data types or functionality. Within the process of code scavenging, programmers may identify relevant pieces of code within previous programs they have implemented or have identified through program analysis. These sections of code are then copied and pasted for further instances, modifying these sections if necessary. Design scavenging involves the reusing of code abstractions, rather than reusing code.

After analysis of the current practices, and performing a judgement upon the size of this master's work, it has been decided that a code and design scavenging approach is taken within this work. An extracted component will be displayed on the screen to the user. The user is then free to copy and paste this component to their work.

3.3.1.2 External Memory Reuse Techniques

During the identification stage of software reuse, an external entity must be used to store components in an organised manner. The manners of which these components are organised are important for the efficiency and accuracy of the system. In this modern era, the storage and efficient retrieval of these components is gained through a database.

3.3.2 Processes

3.3.2.1 Identify Reusable Component

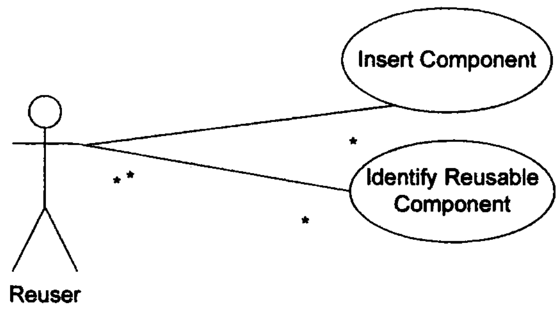


Figure 3.3-1: Realisation of use-cases with ReSULT system.

The primary goal of any reuse system is the identification of components that have the opportunity of being reused. Identifying these components is realised as a use-case because this process directly interacts with a reuser (Figure 3.3-1). This is a crucial stage within the process of software reuse. Any failings within this process may mean a substandard final product being produced or an increase in effort to develop the product. There are two independent approaches towards the identification of reusable components. The engineer may rely upon their own experiences and internal representations (Section 2.3.2.1) to produce sufficient external representations, or they may place their faith into an external memory system (Section 2.3.2.3).

For the ReSULT reuse library, the analysis of comments and program structure will provide an accurate external representation to help a reuser define a realistic mental model of a component. These factors will help the reuser build up a satisfactory mental model of a piece of software from its external representation displayed by the system. By generating this model, the reuser can then start to map between the domains from the top downwards to identify whether this is ideal to their desired usage.

3.3.2.1.1 External Representation

The inserting of a component into the system relies heavily upon how the ReSULT interprets the external representation of the component. External representation can be identified as being a method of transferring the "pseudo code" of the initial software engineering into a form that can be interpreted by other engineers to identify decision decisions and architectures within a system. This can be via notes produced while developing or through documentation produced during the design phase.

To avoid this initial learning curve, the ReSULT system externalises scripts in a textual format. It also ensures that when externalising the problem using database servers, textual search criteria will identify matching text from external representations of components. If these scripts were pictorial, an approach into translating textual search criteria to effectively search the database server would have to be designed.

3.3.2.1.2 Externalising the Internal Representation

The ReSULT reuse library takes into consideration the factors mentioned above. A keyword approach to searching for a component acts as a first level filter. Other details must be given to the reuser to aid further levels of filtering, and because different types of users need different types of information to reuse a component using different kinds of representation. This places a great emphasis on how a component is classified or selected and is of vital importance for a success reuse scheme.

3.3.2.1.3 Identifying and Classifying a Component

A programmer when reusing one of his/her programs will make use of their old internal representations as well as their external representations. These internal representations can be abstracted into the form of programming concepts or patterns that are used as searching criteria to filter their own long-term memory for reusable objects. The internal representations of the components found can be evaluated against each other. If an engineer's own internal representation of components does not reflect well upon their own desired solution of the problem, an external memory can be used to act as a first stage of filtration during the selection process. This external memory is in the form of a library. The automated search tool that examines this library must tackle two cognitive issues.

- *To help the programmer externalise the problem or requirements*
- *Help him select some solution*

ReSULT provides early results within the selection process from which the reuser must evaluate between to gain the most desirable component. To help them make this decision, the results produced from searching the system are provided by entering search criteria concerning the properties of the component i.e. class name, package, imports etc.

This method of finding a component has its advantages due to the ability for an engineer to have access to an infinitive number of components. However, it is of primary importance that the reuser has sufficient knowledge of what that component actually does and how to apply it to his solution. Cross evaluation of a solution is used to identify the best solution but the reuser's confidence of using another engineer's code may be low and may prove to be a de-motivational factor towards software reuse within an organisation.

To reduce any de-motivational factors caused through user confidence, the ReSULT reuse library incorporates feedback from reusers, and the number of times the component has been selected into the searching algorithm. High ratings from reuser feedback and more extractions a component has will provide it with a higher rank when being compared to components with similar criteria that matches the reusers search criteria.

3.3.2.1.4 Developing the 'Identify Reusable Component' Use-case

As discussed in the design Section 3.3.2.1, the use-case 'identify and select component' concerns the realizing of possible reusable components from a reuse library. This involves using search criteria in the form of keywords to generate a list of results. From these results, a software developer or librarian (Section 2.2.2) can view details stored in the database and extract the component if it is desirable.

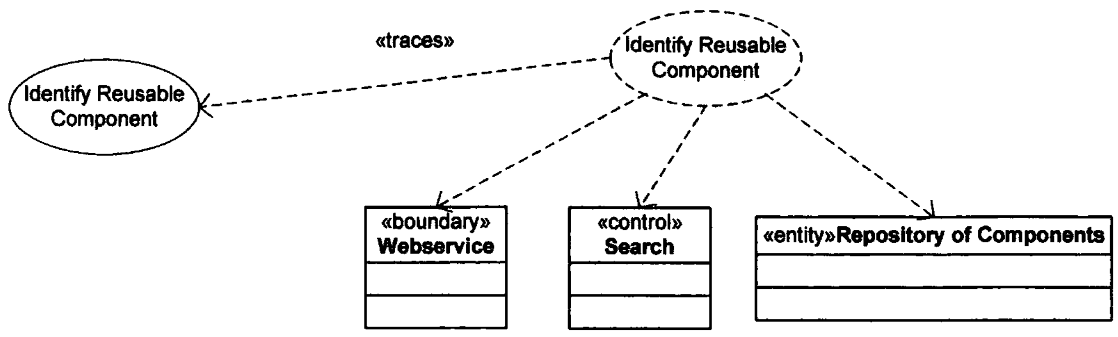

«traces»
Identify Reusable Component
Identify Reusable Component
«boundary» Webservice
«control» Search
«entity»Repository of Components

Figure 3.3-2: The realisation of analysis classes from the use-case.

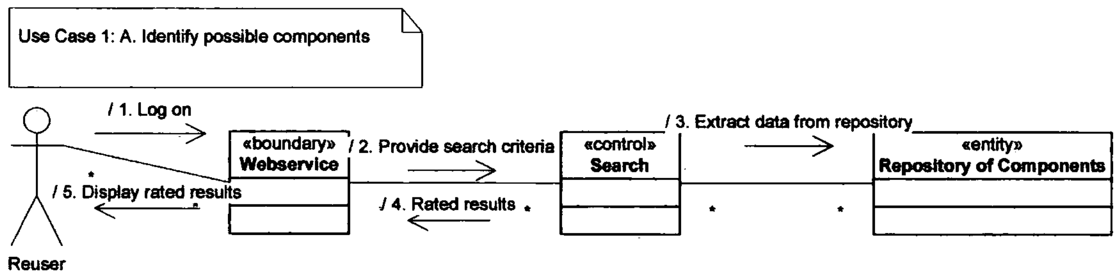

Use Case 1: A. Identify possible components
/ 1. Log on
«boundary» Webservice
/ 2. Provide search criteria
«control» Search
/ 3. Extract data from repository
«entity» Repository of Components
/ 5. Display rated results
/ 4. Rated results
Reuser

Figure 3.3-3: The collaboration between analysis classes.

Figure 3.3-2 identifies the classes involved in the 'identify reusable component' use-case. Figure 3.3-3 displays how these analysis classes interact and collaborate with each other during the use-case. The control class within these diagrams, 'Search', is the focal point for the rest of this section.

The goal of the 'Search' control class is to produce search results from querying the repository of components. The traceability from the analysis model to the design model is display in Figure 7.4-3 of Appendix Section 1. In this figure, the search manager controls interactions between "search keywords", "rate keywords", and "process keywords".

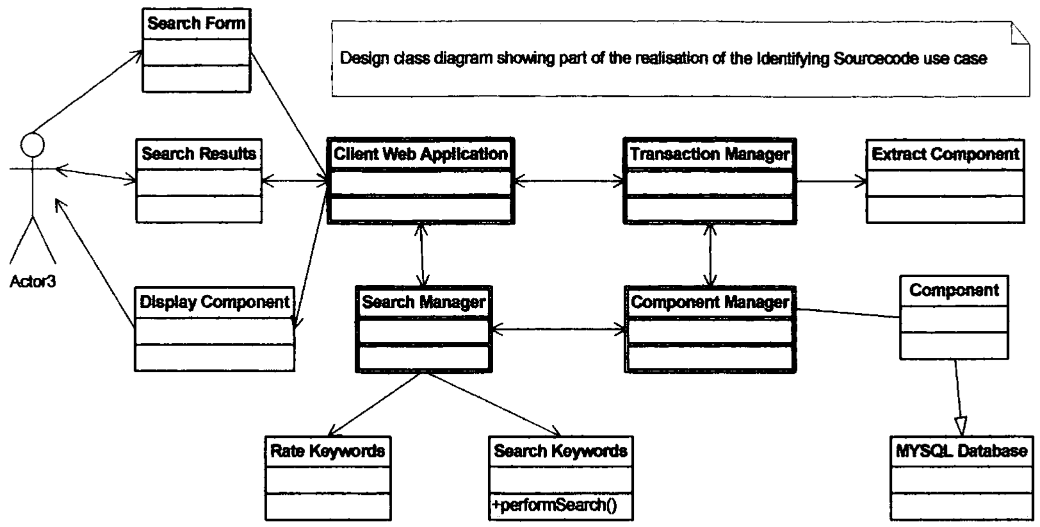


Figure 3.3-4: A Design Class Diagram displaying the classes involved in the use-cases 'Identify and Select Component' and 'Extract Component'.

Figure 3.3-4 identifies how the design classes interact with each other to fulfil the use-case. Once the user has entered the search criteria, these parameters are passed to the web application and then towards the search manager. This design class coordinates the interactions between the database and the components that produce the search results.

The design class 'Search keywords' has the functionality of producing search results from a number of individual searches. The goal of taking this response is to view the code repository using a faceted classification. The facets that are looked upon are:

- *Class Name*
- *Field Name*
- *Method Name*
- *Class Keyword*
- *Method Keyword*

These results are returned back to the search manager where they are placed into ranking by the design class 'Rate Keywords'.

3.3.3 Extract Component

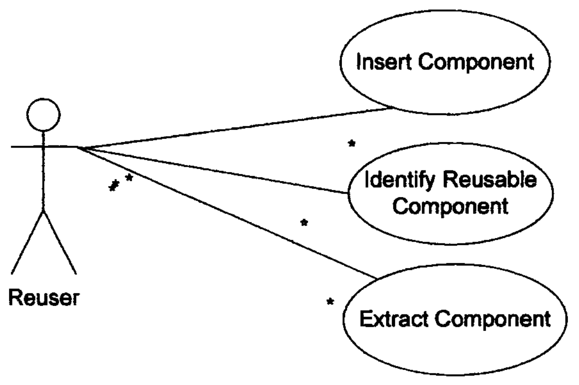


Figure 3.3-5: Realisation of use-cases with ReSULT system.

There is a strong possibility that when a reuser i.e. a software developer or librarian (Section 2.2.2) identifies a component that is suitable to their needs, it will not be 100% compatible with the current system. It is therefore likely that disruption will occur when integrating a component into a system. This 'black box' reuse concept of components is not reusable enough in a current working environment. The 'white box' concept allows the reuser to 'open' the component to automate the specialisation of the code at a low level. The ability for this specialisation allows a general piece of code to be adopted to apply to one's precise problem.

Section 2.6 describes how design patterns fit into software engineering and highlights the positives for use within the development of systems. Design patterns are the easiest to integrate into a design change and are without the demand for any specialisation. This factor often produces a general viewpoint that design patterns are the most successful method of reuse.

There are a number of different levels to which code is observed. Three levels can be defined reflecting upon the tasks needed. The highest level reflects upon management based issues that orientate around making decisions or analysing relationships between elements in a system. A desire to analyse components and data structures within a system is allocated towards the middle layer of a system where system architects. The middle layer also the detailed the algorithms used. A low level approach is undertaken by programmers. Their aim is to understand the semantics of the code, and identify control flow and data flow through the code. ReSULT takes these matters into consideration and allows users to copy and paste code (Section 3.3.1.1). This takes into consideration the fact that the possibility of code being seamlessly integrated is low. It is best giving the reuser the ability to be selective upon the code they reuse to allow for this.

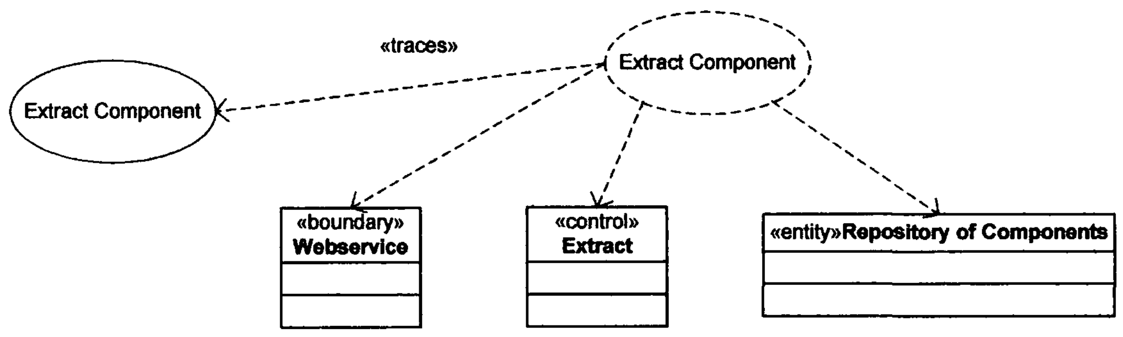


Figure 3.3-6: The realisation of the analysis model from the use-case.

Figure 3.3-6 shown above identifies the classes involved in this use-case. In this section, the focus is on the control class ‘extract’. This will provide the functionality of extracting the component from the database.

3.4 Distributed Technologies

One of the goals of producing the ReSULT reuse library is the ability to provide effective reuse features over the Internet or a company's Intranet. These environments provide a broad variety of implementations, platforms and devices. XML web services provide the ability to exchange messages in a loosely coupled environment using standard protocols such as HTTP, XML, XSD, SOAP, and WSDL. SOAP and WSDL are both based on XML. These XML messages can be structured and typed or loosely defined.

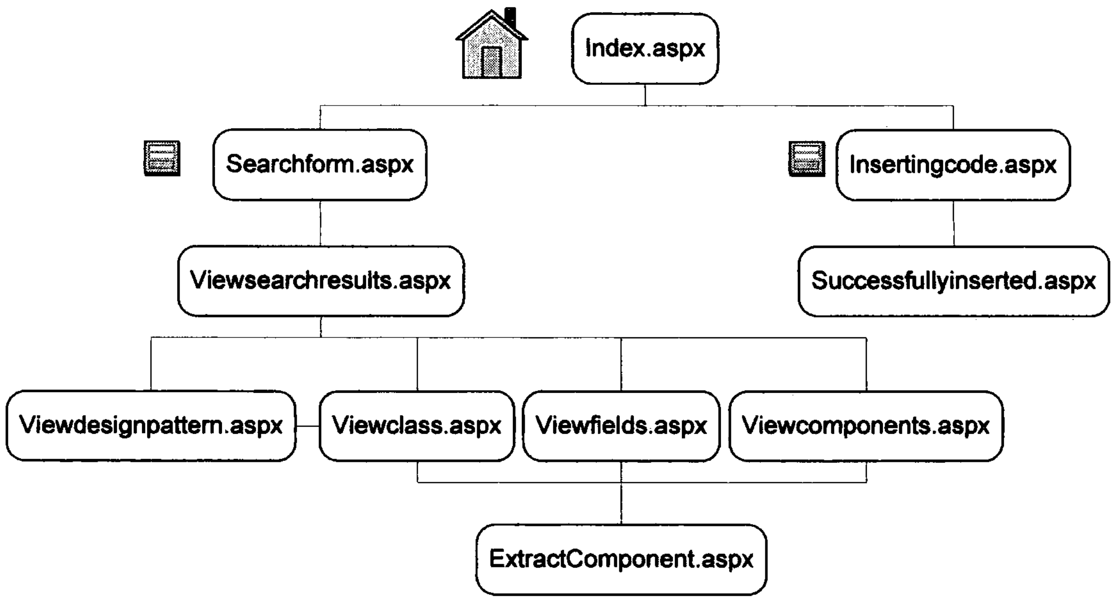


Figure 3.4-1: Web application structure.

In Section 3.3.2 and 3.3.3, a number of web services were defined. The web services designed are:

- *Insert Component*
- *Identify Reusable Component*
- *Extract Component*

The implementation of these web services will be undertaken in the .NET environment. .NET has a number of advantages over other web service architecture; these are listed in Section 2.5.12.

For a user to access these web services, a web application has to be constructed to aid in the usability of the system. The building of this application provides a front end to these services. The user will interact with this system without the knowledge of the complexities hiding behind it. The design for this web application is displayed in Figure 3.4-1. There is a distinct hierarchical structure to the web application. The control flow through the web applications incorporates two operations, inserting components into the system and searching for reusable components. A number of branches to the control flow propagate once search results have been generated. This allows individual elements (such as fields) of component to be examined more closely. The following chart displays the interactions web pages have with web services defined in Section 3.3.2 and 3.3.3.

Web Page	**Web Service Interacted With**
Searchform.aspx	Identify and Select Components
Insertingcode.aspx	Insert Component
Viewsearchresults.aspx	Identify and Select Components
Successfullyinserted.aspx	Insert Component
Viewdesignpattern	Identify and Select Components
Viewclass.aspx	Identify and Select Components
Viewfields.aspx	Identify and Select Components
Viewcomponents.aspx	Identify and Select Components
Extractcomponent.aspx	Extract Component

Table 3.4-1: Web page interaction with web services.

There are two main paths of execution within the system that correspond with the two main processes involved in reuse, identifying a component, and inserting a component. As detailed in Section 3.3.2.1, the more information a reuser is given helps them choose a component; however, users do not want to be overloaded with too much information. It has been decided that the information given to the user is categorised into structures that are identified in Figure 2.5-3. These enable reusers to view fields, classes, and design patterns to help them generate an internal representation of the component.

To communicate between the web application and the underlying web services, data is passed in the format of XML.

3.4.1 XML Encodings

3.4.1.1 Identify and Select Components

```
<Search>
	<Component>
		<ID></ID>
		<Rating></Rating>
		<Name></Name>
		<Package></Package>
		<Interface></Interface>
		<Inherits></Inherits>
		<Designpattern></Designpattern>
		<Abstract>
		<Static></Static>
		<Comments></Comments>
		<Field>
			<Name></Name>
			<Type></Type>
			<Accessibility></Accessibility>
			<Static></Static>
		</Field>
		<Method>
			<Name></Name>
			<Returntype></Returntype>
			<Accessibility></Accessibility>
			<Static></Static>
			<Parameters>
				<Name></Name>
				<Type></Type>
			</Parameters>
		</Method>
	</ Component>
</Search>
```

Figure 3.4-2: The XML encodings for sending a response for a search request from the identify components web service.

Figure 3.4-2 displays the design for XML encoding when from the identify component web service when responding to a message from the web application that contains search criteria. This response is changed to a dataset that is cached within the web application. The data is initial displayed in the Viewclass.aspx (Figure

3.4-1), and is recovered from cache for pages such as Viewfields.aspx, Viewcomponents.aspx and viewdesignpatterns.aspx (Figure 3.4-1).

3.4.1.2 Insert Component

```
<Insert>
        <Language></Language>
        <Designpattern></Designpattern>
        <Trace></Trace>
        <Component></Component>
</Insert>
```

Figure 3.4-3: The XML encoding when sending a request for inserting a component from the web application to insert component web service.

When inserting a component into a ReSULT, a user will be prompted for the file location of the component. In addition to this, the user will have to input the component's language, design pattern, and links to components that it traces from. Figure 3.4-3 displays how this data is encoded into XML at the web application and sent to the insert component web service.

3.4.1.3 Extract Component

```
<Extract>
        <Component></Component>
</Extract>
```

Figure 3.4-4: The XML encoding when requesting to extract a component from the extract component web service.

When a user extracts a component, they have already analysed the structure of the component and related information returned from the identify component web service (Figure 3.4-2). Figure 3.4-4 displays the message returned from the extract component web service; this returns just the code for the component.

3.4.2 Web Application Design

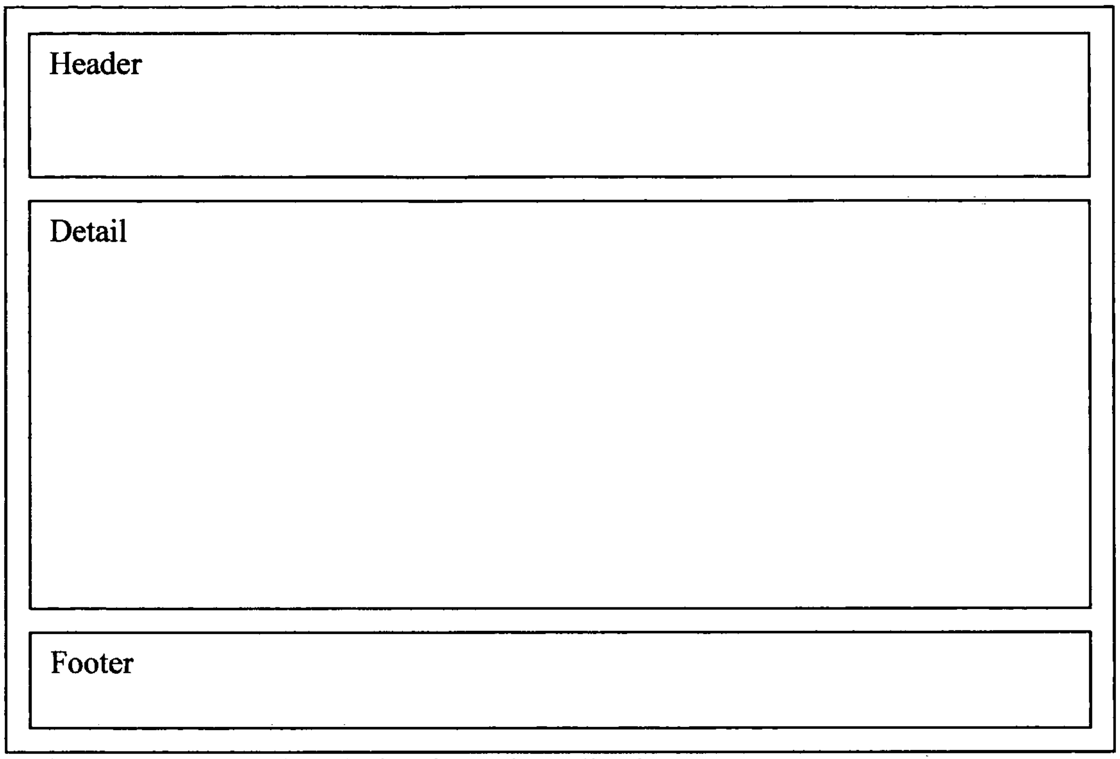


Figure 3.4-5: Template design for web application pages.

The design template for ReSULT's web application is shown in Figure 3.4-5. The header will display the title of a page. The footer for pages that are children of Viewsearchresults.aspx (Figure 3.4-1) will have links to return the user to search results and to search again. All other pages within the web application will contain a hyperlink to direct the user back to the index.aspx (Figure 3.4-1). Formatting within the web application will be designed using CSS scripting (Section 2.5.15.1).

When displaying results, the XML produced by web services 3.4.1 will be translated into displayable HTML using a combination of XSLT (Section 2.5.15) and CSS (Section 2.5.15.1) within the detail section of the template. This will provide organised, human readable information on components.

3.5 Summary

In this chapter, a design for the ReSULT system has been outlined. This design is split into two sections. Firstly, the processes featured within the system are identified and defined using use-cases. These are obtained from examining the processes within current reuse approaches. The identified use-cases are 'Inserting Code', 'Identifying Components', and 'Extracting Component'.

After that, the three use-cases are applied to distributed system architecture. Within Section 3.3, an examination of the technologies involved within the proposed distributed system is examined and the use-cases developed into a workable solution.

Chapter 4 Implementation

4.1 Introduction

In the previous chapter, a number of use-cases were defined for the ReSULT system. These reflected upon how the system would behave towards the user. In this chapter, the findings from the design are expanded to explain 'how' this system is implemented.

The first aspect of the system discussed is the basic functions that were identified from the use-cases found during the design of this system. These functions are examined during Section 4.2, and during Section 4.3, the underlying architecture and development tools needed to fulfil these functions are examined.

From Section 4.4, the details of the system are described in their full context. This includes a number of areas to discuss; the data structures used within the system, the methods used to insert data into the system, the algorithms used to select appropriate components, and how this data is displayed to the user.

4.2 Functions and Development

During Section 3.3.2 and 3.3.3, the use-cases 'Insert Components', 'Identifying Reusable Components', and 'Extract Component' were identified within the proposed design of the ReSULT system. In this section, an introduction into the proposed fulfilment of these use-cases is described, and how these will be developed into a distributed architecture.

By having the function for inserting components into the system, the system provides the user with the ability to distribute their knowledge and expertise through an organisation by allowing users access to their work. ReSULT also helps a reuser filter through many transcripts to identify possible components that may fit into their mental model of their solution. It does this by searching for beacons within the code that fit the search criteria. There are many stages to a reuser selecting a component for reuse within in their own development. During Section 2.3.2 a discussion is made of how reusers define internal representations of code to evaluate their value for development with. Within Section 3.3.2.1.2 descriptions of the various methods for externalising these internal representations are placed. The ReSULT system is designed to aid the identification of possible reusable components that match the reuser's needs by acting as an initial filter of accurate internal representations that are produced by ReSULT parsing components. By these actions, ReSULT aims to reduce the number of components a reuser has to examine before proceeding with development.

Web driven technologies aim to promote distribution within a business. This is based upon open standards for communication, such as HTTP, to provide the ability for servers to talk with other servers and to individual clients. By defining a communication standard, such as HTTP, allows the breakdown of communication barriers that may exist with heterogeneous networks.

4.3 ReSULT Architecture

The ReSULT tool is designed for a distributed heterogeneous network. It is a Web based application that provides the ability to communicate to clients using the standard HTTP protocol. This functionality is provided using a number of servers that offer unique services to client requests.

4.3.1 Servers

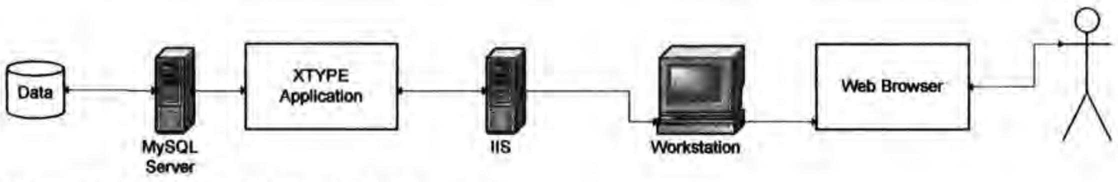


Figure 4.3-1: ReSULT architecture.

Figure 4.3-1 displays the architecture of ReSULT. In this figure, a number of servers are communicating with other servers and other external entities. A database server acts as a wrapper around databases. This controls a number of features such as maintaining consistency of the data, serialisation with atomic transactions, and fault tolerance. In this implementation, it was decided that a MySQL database would provide this service over other database server applications such as Microsoft's SQL server or Oracle because of its availability over the web and the licensing costs that are incurred for using SQL server or Oracle.

Servers often have designated processors. By having these individual processors, multiple requests are dealt with effectively and efficiently. The ReSULT prototype is an experimental system designed to demonstrate a possible solution to a reuse library; therefore it was decided that there would be no loss in performance if both the MySQL server and the Microsoft Internet Information Server were to be located on the machine, and operating from the same processor.

A web service deals with requests from Internet browsers and finds the file or program requested. The chosen web server for ReSULT was the IIS 6.0 (Internet Information Services) by Microsoft. This is because IIS has the ability to handle requests for active server pages (ASP) by implementing an ASP. NET worker process, and dealing with web service interactions. IIS is more desirable to developers working in Microsoft operating systems. Other products such as the Apache Web Server are aimed towards non Microsoft operating systems where web applications such as Tomcat are being used to process Java Servlets and Server Pages.

1.1.1 Software

Windows XP Pro
Internet Information Services 6.0
MySQL Server
.NET Framework 1.1
C# .NET within the IDE Visual Studio .NET 2003
Internet Explorer 6.0

1.2 Hardware

Pentium IV 2.66MHz
256Meg RAM
Ethernet 10/100Meg

Figure 4.3-2: Details of software and hardware development environment.

Figure 4.3-2 displays the details of the hardware and software that was used in the development of the ReSULT system. It was decided that this project would be developed using mainly Microsoft products. Over the past years, Microsoft has developed a stronghold within the software development market. Recently the emphasis has moved towards the distributed computing area. With the launch of .NET, Microsoft has developed an approach that encompasses this area. One feature that .NET has is the ability to reduce the number of requests made to services. Microsoft developed within its .NET framework an approach that drastically reduces the number of requests made between a web application and a database server. They identified that by taking a subset of data from the appropriate data sources at the beginning of the process and storing the data locally within a table (in effect caching data) inside the web application reduces the number of connections being opened and closed between the web service and database server. This feature becomes of great significance because it reduces the demand on the database server when an application is scaled up to cater for 10,000 transactions. The capability of implementing this feature may not be of much significance during this research because of the number of users, but it this functionality will be implemented and evaluated within the ReSULT system.

4.3.2 ASP.NET

ASP.NET is a unified Web development platform that provides the services necessary for developers to build enterprise-class Web applications. It is a compiled, .NET-based environment that allows authoring in any of the .NET compatible languages and access to the framework classes.

ASP.NET takes advantage of performance enhancements found in the .NET Framework and common language runtime (listed in section 2.5.12). These features offer significant performance improvements over ASP and other Web development platforms. Other than the Just-In-Time (JIT) compiling of managed code into native code, ASP.NET offers a number of performance enhances by offering:

- *Extensive caching services (both built-in services and caching APIs).*
- *Factorability, meaning that developers can remove modules (a session module, for instance) that are not relevant to the application they are developing. ASP.NET also provides.*
- *Performance counters that developers and system administrators can monitor to test new applications and gather metrics on existing applications.*
- *Provide default authorization and authentication schemes for Web applications.*

- *Configuration settings are stored in XML-based files, which are human readable and writable. Each application can have a distinct configuration file and can be extended to requirements.*

4.3.3 Microsoft Visual Studio

Microsoft Visual Studio .NET 2003 packages all the features of the .NET framework into an integrated desktop environment that allows seamless editing of HTML editing, compiling source code and other programming tools such as handling ADO interactions, xml handling but to name a few. Not only does this make Web development easier, but it also provides all the benefits that these tools have to offer, including a GUI that developers can use to drop server controls onto a Web page and fully integrated debugging support.

4.4 Data Structures

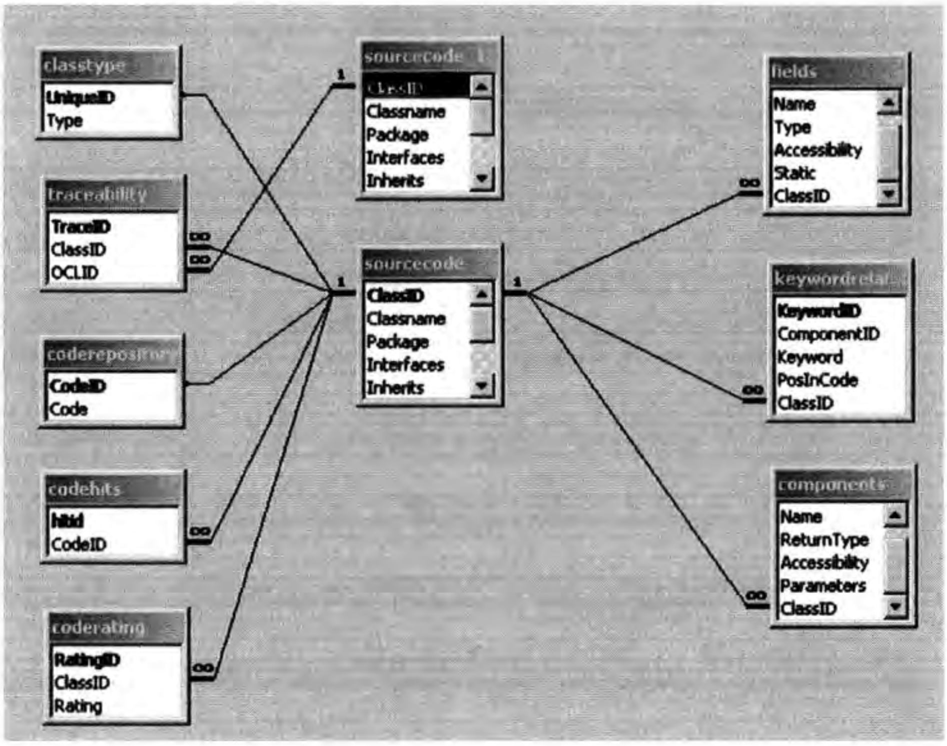


Figure 4.4-1: Table layout used in ReSULT.

Section 2.3.2.1 defines what features of a component are needed when identifying its internal representation, and are listed in Figure 3.2-4. These features are displayed within the '*Sourcecode*' database table in Figure 4.4-1. The internal representation of a component is separated from its actual content (stored inside the table '*coderepository*') to maintain efficiency in the searching repository. Additional properties, such as fields and components, are grouped together into individual tables. This is to cater for the need to search amongst these properties individually during the identification of suitable components (Section 3.3.2.1.3).

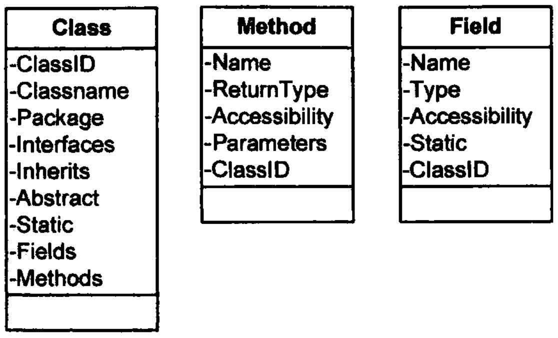


Figure 4.4-2: Tables involved in the development of the '*SourceCode*' Object.

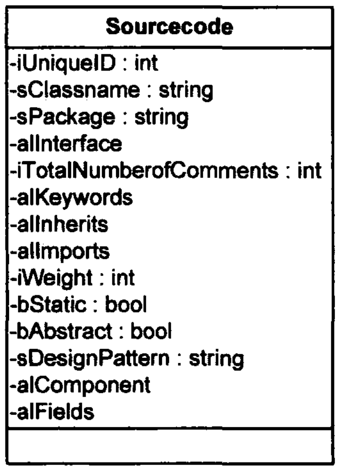


Figure 4.4-3: '*Sourcecode*' class (mutator[1] and accessor[2] methods are not shown).

OCL and Java scripts are converted into one homogeneous '*Sourcecode*' object (Figure 4.4-3). The components of this object are displayed in Figure 4.4-2. This promotes the possible expansion of the system to include different scripting languages, and aids in the transferring of objects between classes that are involved in a process. The work needed to do this consists of only defining a process of converting these new scripts into the homogeneous object. The programmer does not need to focus upon the underling details of ReSULT programming. '*classtype*' data table stores the original types of these homogeneous objects.

[1] A mutator method enables a private field variable within a class to be changed.
[2] An accessor method enables a class to obtain the value for a field variable.

Data involving the number of hits and the ratings a reuser gives to a component (3.3.2.1.3) are stored inside the tables '*codehits*' and '*coderatings*' respectively.

Traceability between OCL scripts and Java source code (as mentioned in Section 3.2.1.1) are stored inside the 'traceability' data table.

4.5 Fulfilling Use-cases

4.5.1 Inserting Code

During a reuser's interaction with the ReSULT system, he/she will use various forms that help users insert data and receive data from the system that will satisfy the 'Insert Component' use-case. These forms are part of the web application. The web application is a boundary class because of its interaction with entities outside the ReSULT architecture.

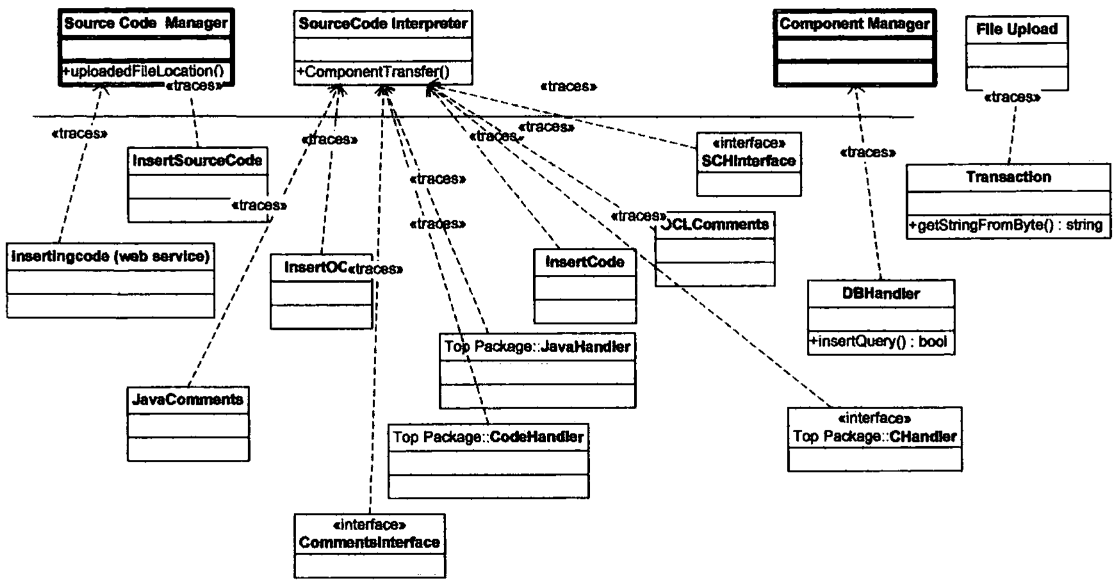


Figure 4.5-1: The traces between the design model in Figure 3.2-7, and of the implementation classes.

Figure 4.5-1 displays the implementation objects present within ReSULT for the inserting code use-case. When a user has selected the file that they wish to insert into the system on the web form "*insertingcode.aspx*", an ASP function called "*System.Web.UI.HtmlControls.HtmlInputFile*" is used to coordinate the uploading of this file from the client's location. The result of performing this action is a binary array. The "*insertcode.aspx*" passes this binary array as a parameter to the web service "*insertingcode.asmx*" which inherits the object 'Insertingcode' (Figure 4.5-1).

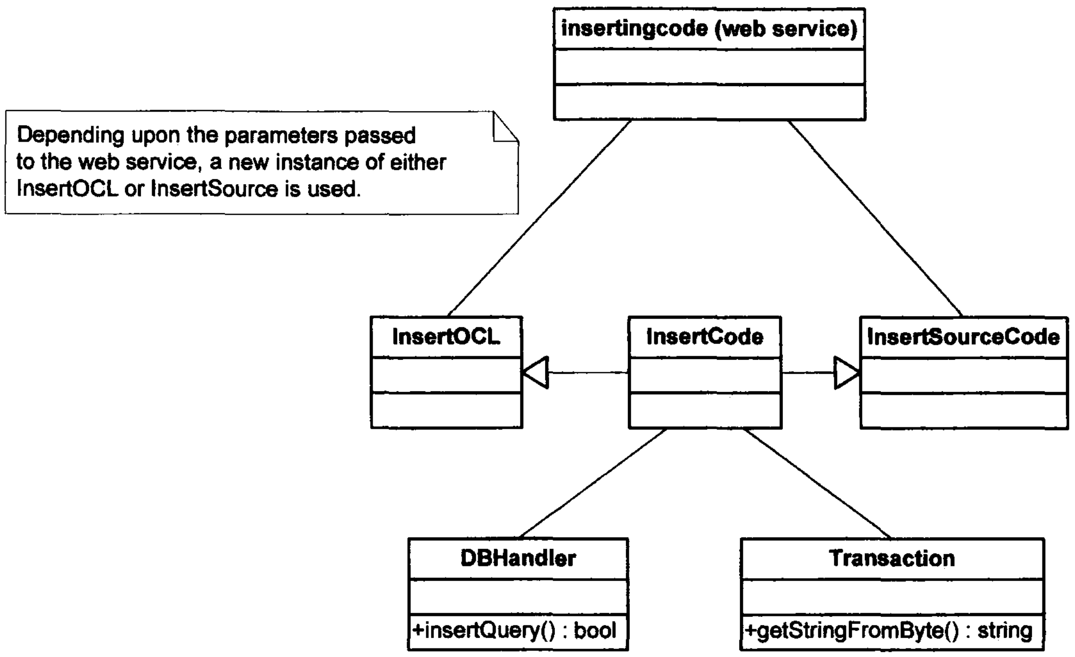

insertingcode (web service)
Depending upon the parameters passed to the web service, a new instance of either InsertOCL or InsertSource is used.
InsertOCL
InsertCode
InsertSourceCode
DBHandler
+insertQuery() : bool
Transaction
+getStringFromByte() : string

Figure 4.5-2: Implementation classes in the 'Inserting Source Code' use-case.

Figure 4.5-2 displays the relationship between implementation classes (Figure 4.5-1). 'InsertOCL' and 'InsertSourceCode' both inherit from 'InsertCode' because there are common functional elements when inserting code such as the facets that the search mechanism uses (Section 3.3.2.1.4)

Once the web service receives the binary contents of the file, it must be converted back into a string format; the method *Transaction.getStringFromByte()* (Figure 4.5-2) performs this action.

```
private string getStringFromByte(byte[] filecontents)
{
        string sResult="";

        for(int i=0;i< filecontents.Length;i++)
        {
                if(filecontents [i]!=0)
                {
                        byte byTmp =0;
                        byTmp filecontents [i];
                        sResult += System.Convert.ToChar(byTmp);
                }
        }
        return sResult;
}
```
Figure 4.5-3: A C# method that converts a binary array into a string.

After the conversion of binary array data into a string that reflects the contents of the uploaded file, this string is then analysed. The discussion of the analysis process follows on in the next section.

4.5.1.1 Code Analysis Framework

Figure 4.4-3 displays the objects that are returned after the operation of *ComponentTransfer()* in the '*SCHInterface*' class. The goal of this class is to analyse the inserted code and identifying the information that aids successful code reuse (as discussed in Section 3.2.2). The process of interpreting source code produces the '*Sourcecode*' object that contains in its fields '*Field*' objects and '*Method*' objects; both of these are stored in individual Arraylists. This design provides the idea that code (either Java or OCL) when translated produces one homogeneous '*Sourcecode*' Object. This reduces complexity that may incur if different codes are interpreted into their different '*Sourcecode*' objects. The Class '*Object*' is then passed to the '*DBHandler*' class that converts it into SQL statements for insertion into the relevant tables.

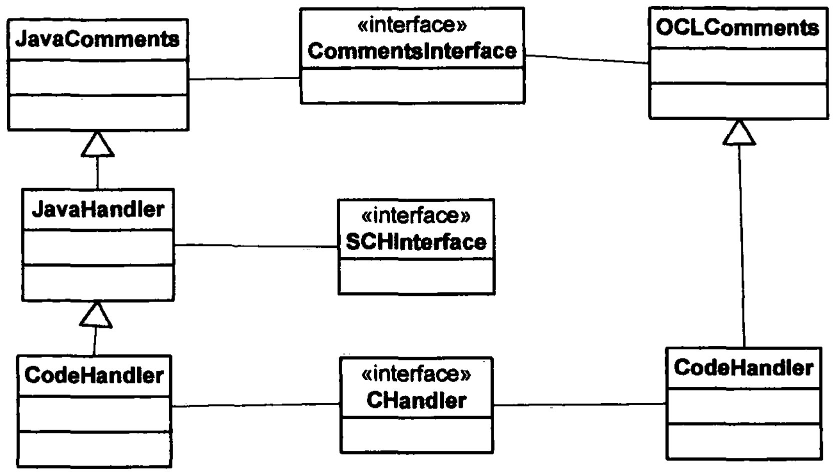


Figure 4.5-4: Inserting code framework.

Figure 4.5-4 displays the framework implemented in the design class '*SourceCode Interpreter*'. This framework promotes extensibility of the ReSULT system to other programming languages by providing interfaces for programmers to implement.

In Section 3.2.2, an analysis of the scripting languages Java and OCL was undertaken. The findings made from the analysis identified certain features each possessed. From these features, three interfaces were designed to give the opportunity for languages to be integrated into the system (full transcripts of these interfaces are found in Appendix Section 8). The features identified consisted of a basic structure of blocks that distinguished segments of code. Similar features were apparent in both languages such as fields and methods, and comments. Although, these two languages contained the same properties, they were defined in different ways; therefore, interpretation classes for both languages were needed. To ensure that these properties were implemented for both languages, programmers need to implement the framework that is defined in Figure 4.5-4.

```
string [] identifyMethod (string sSegment, int iComponentID);
ArrayList identifyBlocks(string sTemp);
```

Figure 4.5-5: Segment of '*SCHInterface*'.

The '*SCHInterface*' within the '*Inserting code*' framework provides the ability for the analysis of object orientated scripting languages. If code is not based around blocking and object orientation e.g. OCL, this interface needs not be implemented (as shown in Figure 4.5-5). A key feature with the processing of scripts in ReSULT is the blocking mechanism. Lines of code are processed together i.e. methods are extracted and analysed, during this analysis comments and structure about that method are recorded. This interface also extends the features identified in the '*Codehandler*' interface by allowing the user to identify advanced features that may not appear in other languages such as accessibility (public, private etc.), inheritance (abstract, final etc) but to name a few.

```
ArrayList identifyComments(int iComponentID,string sSegment);
void addLineComments(string sComment, int iComponentID);
void checkExpelledWords(string item,int iLocationOfWord,int iComponentID);
```

Figure 4.5-6: Methods in the '*CommentInterface*'.

One feature observed was that of different styles techniques of commenting are seen. As noted in Section 2.3.2.1, there are two possible types of commenting inline and prologue. Some languages do allow prologue comments while others do not, such as OCL. The starting signature that a line maybe a comment also varies from language to language, and thus must be implemented differently for these languages using the interface '*CommentInterface*' (Figure 4.5-6).

The goal of *checkExpelledWords()* in the *CommentInterface* is to allow the programmer the ability to analyse comments to identify whether they are nouns or commented out code. If these are not removed, they may affect database efficiency, or produce unwarranted results when searching for components.

4.5.1.2 Keyword Analysis

Keywords are a method of externalising the internal structure of a piece of software (Section 2.3.2.2). The keywords selected must give an accurate representation of this internal structure. The selection process for keywords differs for both scripting languages because their internal structures do not resemble each other's. OCL describes constraints about a system during the design process while Java describes and implements these constraints. There must be an approach towards the analysis of these scripts to extract information from both sources that resembles each language's internal structure; thus promoting traceability inside the system.

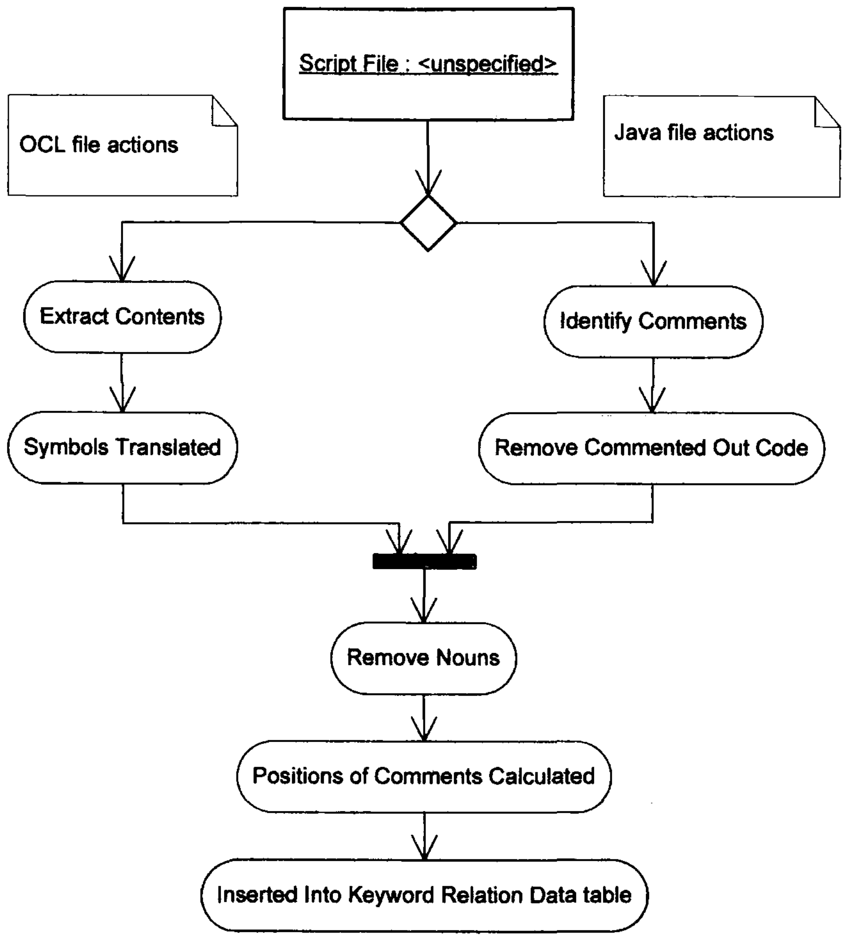

Script File : <unspecified>
OCL file actions
Java file actions
Extract Contents
Identify Comments
Symbols Translated
Remove Commented Out Code
Remove Nouns
Positions of Comments Calculated
Inserted Into Keyword Relation Data table

Figure 4.5-7: The process of identifying and inserting comments from Java and OCL files.

In the ReSULT system, it was identified in Section 2.3.2.3 that reusers when understanding software define an abstract representation of it. To automate this process of abstraction, the ReSULT system must analyse code for beacons (as identified in Section 2.3.1.1). An OCL script is observed as being abstract of what it implements in a programming language; therefore, it was decided that all the data in this scripting language is analysed as keywords. When analysing Java source code, the data extracted must roughly resemble that of the OCL files. As seen in

Figure 3.2-3, OCL contains information concerning packages, classes, methods, fields, and comments; this same information is identified in Java.

Figure 4.5-7 displays the different routes of analysis taken in the interpretation of Java and OCL. OCL files contain large amounts of symbolic representation. The translation of these symbols into words ensures that there is no confusion when producing search results. A list of these translations is found in the Appendix Section 7.

```
"}","{"," ", ", ", ".", ":", "and", "the", "a", "to", "is", "at",
"this", "all", "\r", "\n", "\t", "on", "of", "", "*", "break;"
```

Figure 4.5-8: A collection of expelled words or characters that will not appear as keywords.

In Figure 4.5-7, the execution path for the process of keyword analysis separate and merges for the different scripting languages. Once the comments in each language are identified, the path merges. At this stage, all expelled words and characters that may reduce the effectiveness of the searching capabilities of the ReSULT system are removed from the comments. A list of these is displayed in Figure 4.5-8.

The next stage of keyword analysis involves the searching mechanism as discussed in the following section. By identifying the location of where each keyword appears in a script and placing those details into a separate table with the keyword, the amount of data that one search has to analyse is reduced significantly.

4.5.2 Identify and Select Component

The search algorithm consists of observing the repository at different façades. It was decided in Section 3.3.2.1.4, to construct this approach to produce results that took into consideration a wider search space, which returned results that would be more accurate to the reuser. The concept behind this evolves around searching a number of database tables. These are entered into the system using the approach described in Section 3.3.2.1.3. The tables searched are '*Keywords*', '*Sourcecode*', '*Field*', '*Method*', and '*Class*'. The searches performed on these tables are categorised into two groups:

- Structured Search.
- Keyword Search

A structured search operates on the tables '*Fields*', '*Method*', and '*Class*' tables whereas the keyword search consist of the search results from tables '*Class*' and '*Method*'.

The following sections describe how each search produces results, and how they are amalgamated together to form one set of ranked results.

4.5.2.1 Structure of SQL Query Searches

A simple SQL query is used to identify whether any of the search criteria match records in any of the three tables. The SQL function 'Like' is preferred to '=' because it allows the conception that a reuser does not exactly know what they want, so solutions that approximately fit the search criteria should be allowed as a possible result.

The results returned from these queries are not in any ranking order. When performing search queries on the 'keywords' table '=' is used instead of 'like'. The idea behind this is to minimise the result set that is returned from executing the query. Keyword searching when compared to structured searches produces a larger results set. If the function 'like' is used when querying this result set would be even larger, and thus would reduce efficiency, and may cause incorrect components to be displayed in the final set of amalgamated results.

4.5.2.2 Keyword Ranking Algorithm

The class '*Ratekeywords*' contains the algorithm for calculating rank. This is involved when calculating distance between multiple search criteria for example.

Example 1

// *searching* these words to match specific *criteria* provides so much benefit to reusers

Example 2

// *searching criteria* for a reuser

Figure 4.5-9: Demonstrating ranking of the keyword algorithm.

If two components both contain the keywords specified in the search criteria, there must be a way of specifying whether one component is more desirable then the other. This is done by examining the distance between where the words occur within the script. In Figure 4.5-9, if the search criteria are 'searching' and 'criteria', Example 2 would be ranked higher than 'Example 1'. This is because both elements of the search criteria can be found closer than they can be found in Example 1.

This approach is simple in theory for working with only two words, but what happens if more than two words are entered as search criteria, or there are multiple occurrences of a word? The approach taken to tackle this problem involves three stages of processing.

- *Grouping together keywords from the same class*
- *Ordering keywords into the order they appear in the script*
- *Calculate the distances between each word in a serial fashion and obtaining an average from these values.*

$$\text{Average Weight} = \frac{\text{Sum distances between all elements}}{\text{Number of Elements} - 1}$$

Figure 4.5-10: Method of calculating average weight for keyword distances in a class.

For each class that appears in the results, an average is obtained from the calculation shown in Figure 4.5-10. A record is also kept of how many keywords (including replications) appear inside a class.

This class returns an Arraylist of '*Sourcecode*' Objects sorted by their average weighting. If averaged result scores obtained are equal, the numbers of keywords that appear in the script are used to define which one gets the higher ranking. The returned Arraylist will then be passed to the rating algorithm where it will be used in the rating algorithm.

4.5.2.3 Rating Algorithm

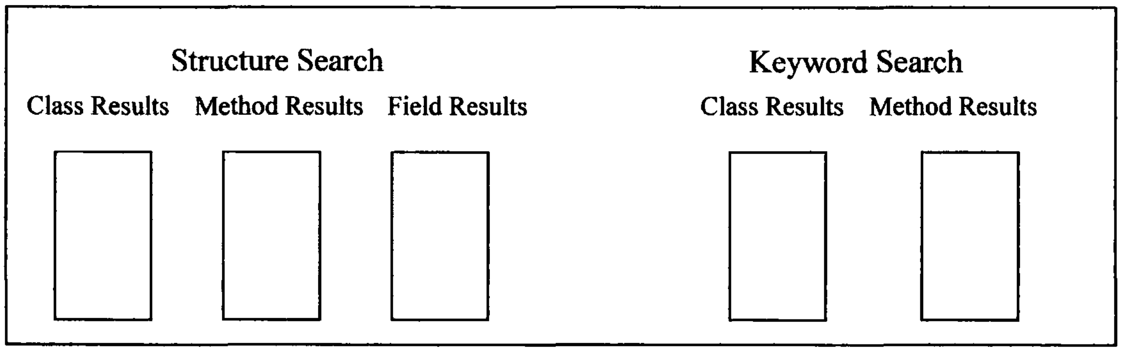


Figure 4.5-11: Arraylist fields found in '*Ranking*' class.

From the five different façades observed of the reuse repository, five arraylists containing ranked results are stored as fields in the '*Ranking*' class (shown in Figure 4.5-11). These five data collections are grouped together to form a list of class ids that have been reported as being relevant to the search. For each class id stored in this data collection, each result's arraylist is queried to identify the ranking, if any. If no rank is obtained from an arraylist, the rank is identified as being 0. Once all ranking scores are identified from the five data collections, the scores are processed by *calculateScore()*. When the rating is returned, the '*SourceCode*' Object is created by querying the *'SourceCode'* table with the class's id, and then is passed to the method *InsertIntoRanked()*.

x^W	= Weighting
x^R	= Ranking
CKeyword	= $\text{CKeyword}^W * \text{CKeyword}^R$
MKeyword	= $\text{MKeyword}^W * \text{MKeyword}^R$
Hits	= HitsR*HitsW
Ratings	= RatingsR*RatingsW
X	= $\text{CKeyword}+\text{MKeyword}+\text{CStructure}^R+\text{MStructure}^R+\text{FStructure}^R$ – Hits - Ratings
Rating	= 100.001 – x

Figure 4.5-12: Calculating rating formula in *calculateScore()*.

The method *calculateScore()* calculates a rating for a script. Scores for the ratings of components are in percentages. This reflects upon the ideology that it is highly unlikely that a component will fit exactly into the mental model the reuser has for the desired solution, and therefore the likelihood of scoring 100% is low.

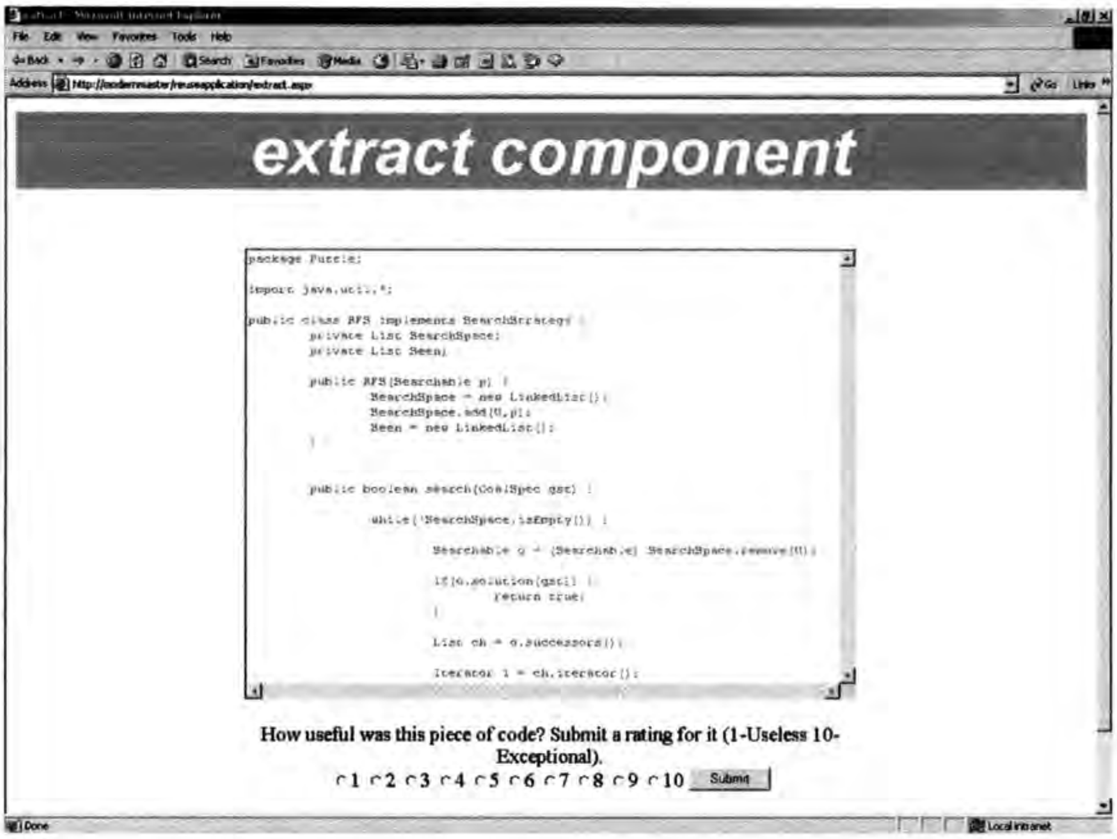
To produce dynamic ratings that are not just based on scripts but from feedback from reusers, data concerning the number of extractions ('*Hits*') and ratings submitted are used in *calculateScore()*. Section 3.3.2.1.3 discusses why this approach was taken.

Structure searches populate fields in the '*Ranking*' class, such as CStructure^R (class), MStructure^R (method) and FStructure^R (field). These collections are unordered, whereas searches concerning Keywords are ordered. These are also stored as fields of the Ranking class (CKeyword for classes and MKeyword for methods).

Figure 4.5-12 displays the formula that is used for calculating ratings. This produces results that are highly unlikely to be 100%. For a script to be rated as 100%, it must have result entries found in CStructure, MStructure, FStructure, CKeyword, and MKeyword. This would result in a static score of that would be no less than 4 (when weightings for keywords equal 0.5). With weightings for dynamic scores such as for hits and ratings, both equal to 0.001. 1000 hits and 1000 '1' ratings (with no other ratings) would provide a deduction of 4. This implies that x = 0 and nothing is taken away from the value 100.001. It is highly unlikely for this outcome to occur, and highlights the points made above about the possibility of components fitting seamlessly into a reuser's mental model of their solution.

4.5.3 Extract Component

The goal of the '*extract component*' use-case is for the user to be able to gain access to components that are stored within the reuse repository. As detailed in Section 3.2.2, the actual storage of the component is separated from its external representation. This component is stored as a BLOB (large binary object) because the size of components can be infinite; therefore, components could not be stored in a normal text field.



```
package Puzzle;

import java.util.*;

public class BFS implements SearchStrategy {
        private List SearchSpace;
        private List Seen;

        public BFS(Searchable p) {
                SearchSpace = new LinkedList();
                SearchSpace.add(0,p);
                Seen = new LinkedList();
        }


        public boolean search(GoalSpec gst) {

                while(!SearchSpace.isEmpty()) {

                        Searchable q = (Searchable) SearchSpace.remove(0);

                        if(q.solution(gst)) {
                                return true;
                        }

                        List ch = q.successors();

                        Iterator i = ch.iterator();
```

How useful was this piece of code? Submit a rating for it (1-Useless 10-Exceptional).

○1 ○2 ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10 Submit

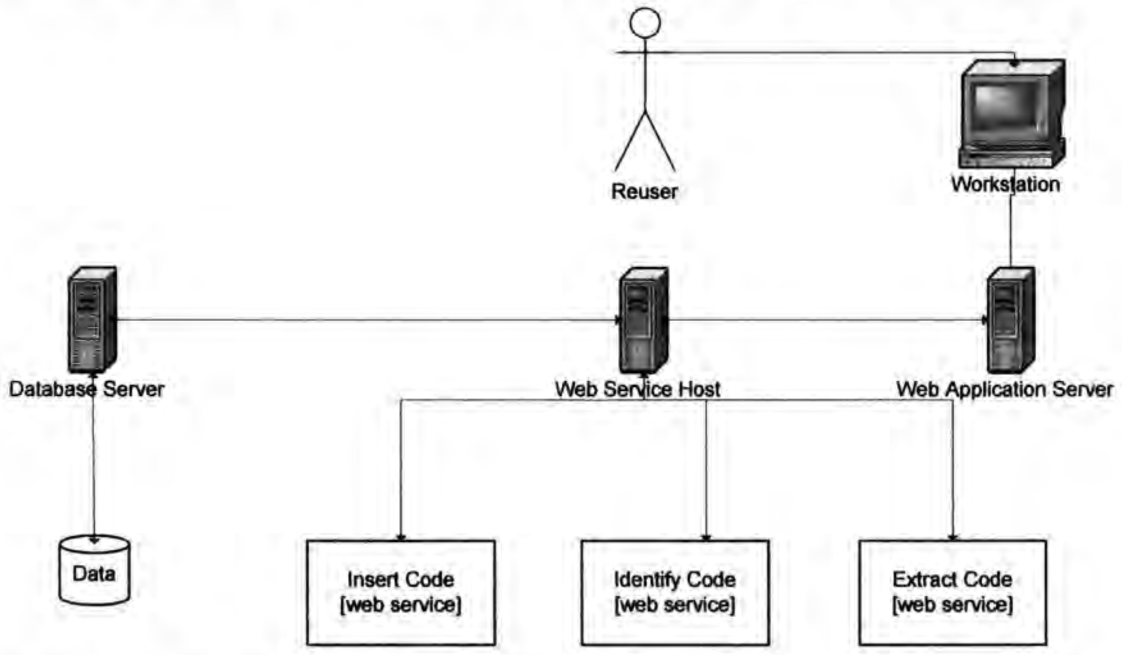
Figure 4.5-13: View of extracted component on the ReSULT system.

Once a user identifies a component that they wish to analyse to a greater depth or use it in their program. They can extract the code from the code repository table within the system's database. A component is extracted by selecting the extract hyperlink. This data is transferred to '*extractcomponent.aspx*' that initiates and passes the component id over to the web service '*extractcomponent.asmx*'. This web service obtains the component from the repository. The component is held inside the database as a BLOB object. When this database is queried for this BLOB object, a binary array is returned. To convert this to a string format, a new instance of the '*transaction*' class is created. This same mechanism was used to input data into the system. The code for this is displayed in Figure 4.5-3. After conversion from byte array to string, the component is displayed on the resulting web page (Figure 4.5-13).

4.6 Web services

4.6.1 Architecture

For the three use-cases identified during Chapter 3, individual web services will be used to represent each of these use-cases. This ensures that the responsibility for satisfying these use-cases solely relies upon these services. If the qualities of these services are of a high standard, it can be assured that the use-cases are satisfied.



Reuser
Workstation
Database Server
Web Service Host
Web Application Server
Data
Insert Code [web service]
Identify Code [web service]
Extract Code [web service]

Figure 4.6-1: ReSULT web service architecture.

The web services in Figure 4.6-1 are located on a web service host. These services do not have to be on the same host, but for practicality issues they were for this project.

The ReSULT web application is deployed on the web application server. However, this web application must firstly identify these services using WSDL descriptions that are located in UDDI directories. These directories contain a list of services that are registered with them. Querying a UDDI directory will result in the ability for a programmer to analyse WSDL descriptions, and select services that would be of benefit to them. The ReSULT web services are registered in a UDDI directory. This opens up the opportunity for the integration of another organisation's reuse system to enlarge the current knowledge base (Section 2.4), and allows the opportunity for Business-2-Business commerce of reusable assets (Section 2.5.3). The full WSDL transcripts are found in the Appendix Section 2.

4.6.2 Data Transfer inside the ReSULT Architecture

Communication between the web application and the three services is coordinated using SOAP messaging. As discussed in Section 2.5.7, SOAP is an extension of XML. .NET allows the programmer to take advantage of SOAP messaging using the Dataset class that is found in the API of .NET.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<Sourcecode>
        <Table>
                <ClassID>12</ClassID>
                <Classname>CLASSNAME1</Classname>
                <Package>PACKAGENAME1</Package>
                <Imports>
                        <Import>
                                <Name>Import1</Name>
                        </Import>
                        <Import>
                                <Name>Import2</Name>
                        </Import>
                </Imports>
                <Inherits>
                        <Inherit>
                                <Name>INHERITS1</Name>
                        </Inherit>
                        <Inherit>
                                <Name>INHERITS2</Name>
                        </Inherit>
                </Inherits>
                <Interfaces>
                        <Interface>
                                <Name>INTERFACE1</Name>

                        </Interface>
                        <Interface>
                                <Name>INTERFACE2</Name>
                        </Interface>
                </Interfaces>
                <Accessibility>PUBLIC</Accessibility>
                <Static>YES</Static>
                <Fields>
                        <Field>
                                <id>1</id>
                                <Name>field1</Name>
                                <Type>INT</Type>
                                <Accessibility>PUBLIC</Accessibility>
                                <Static>NO</Static>
                                <Abstract>NO</Abstract>
                        </Field>
                        <Field>
                                <id>1234</id>
                                <Name>FIELD2</Name>
                                <Type>INT</Type>
                                <Accessibility>PPRIVATE</Accessibility>
                                <Static>NO</Static>
                                <Abstract>NO</Abstract>
                        </Field>
                </Fields>
                <Components>
                        <Component>
                                <id>1239456</id>
                                <Name>COMPONENT1</Name>
                                <ReturnType>INT</ReturnType>
                                <Accessibility>PUBLIC</Accessibility>
                                <PARAMETERS>
                                        <Type>INT</Type>
                                        <NAME>PARA1</NAME>
                                </PARAMETERS>
                                <PARAMETERS>
                                        <Type>INT</Type>
                                        <NAME>PARA2</NAME>
                                </PARAMETERS>
                        </Component>
                </Components>
        </Table>
</Sourcecode>
```

Figure 4.6-2: Example of a Dataset in XML view.

Datasets are passed between the web application and services to help those entities successfully complete their defined processes. Figure 4.6-2 displays a simple Dataset structure; this highlights the use of XML within Datasets and how easily this system could be integrated with other heterogeneous systems by using these open standards.

An important advantage of using Datasets within the ReSULT system is the ability to cache data. This caching enables the reduction of connections made to databases that significantly reduce the load a database server has to take when the system is scaled up (Section 4.3.1). This is of significant benefit to this system when considering the size of data that is searched through for each user.

The information stored in these datasets will hold data from a number of data tables that will be used during a user's session. To prepare the datasets for this data, an XML schema is imported into the datasets. This provides table structure information and appropriate relational constraints between those tables that will enable the dataset to be queried. XML schemas are defined using XML schema definition language (XSD).

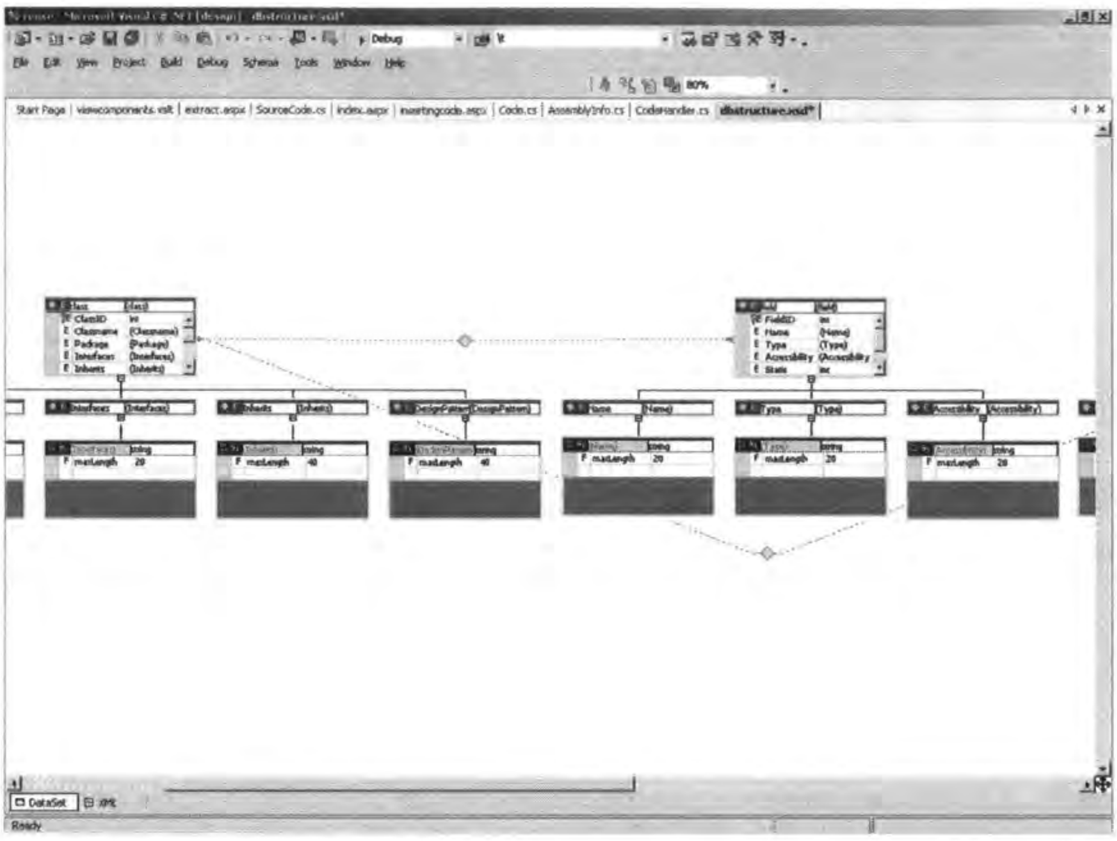


Figure 4.6-3: A screenshot of a small section of '*dbstructure.xsd*' in a graphical format.

Microsoft's Visual Studio 2003 provides graphical support for this within their development tool. Within this graphical support, addition elements such as types, complex types etc. can be added to the schema. An option to display the raw XML is given at the bottom of the screen. The full XSD transcription in XML is given when this option is selected. This is shown in the Appendix Section 6.

```
Session["SearchResults"] = dsResults;
Session["DBStructure"] = dsStructure;
```

Figure 4.6-4: ASP.NET code displaying the initialising of dataset session variables.

For each search performed by the ReSULT system, two datasets are created and populated with data from the database tables' class, fields, and components (Figure 4.6-4). With this choice of tables, the opportunity for the '*sourcecode*' object (Section 4.4) to be created is given. One dataset will represent the entire collection of components held within the system for the desired language. The other dataset will hold the results returned by the system for the search criteria entered. In addition to the data tables mentioned earlier on in this paragraph, this dataset holds information concerning rating. This is of vital importance when displaying the results to the user.

4.6.3 Displaying Transferred Data in ReSULT

As seen in Figure 4.6-2, raw XML data within datasets is not very pleasing to the eye. With the ReSULT system there are a number of pages that display database information. These pages are:

- '*viewsearchresults.aspx*'
- '*viewfields.aspx*'
- '*viewcomponents.aspx*'
- '*viewclass.aspx*'
- '*viewdesignpattern.aspx*'

Each page displays different data upon it; therefore, each one will have its own translation page. As discussed in Section 2.5.15, XSLT style sheets use XPATH expressions to locate and display data within the associated XML sources.

```
        <xsl:variable name ="ClassID" select="ClassID"/>
```

Figure 4.6-5: Sample of '*viewclass.aspx*'.

XSLT gives the opportunity for XML data to use within HTML components. This ability is given by given a variable with the style sheet (see Figure 4.6-5).

```
              <TD>
                     <form action="viewfields.aspx" method="post" name="ClassID"
target="_self">
                     <input type = "hidden" name="classID" value="{$ClassID}" />
                     <input type="submit" value="View" />
                     </form>
              </TD>
              <TD>
                     <form action="viewcomponents.aspx" method="post"
name="ClassID" target="_self">
                     <input type = "hidden" name="classID" value="{$ClassID}" />
                     <input type="submit" value="View" />
```

Figure 4.6-6: Sample of '*viewsearchresults.aspx*'.

These XSL variables are applied to a HTML component by encasing the declared variable within curly brackets. A number of examples showing how this is applied are displayed above in Figure 4.6-6. This is where xml data is being used as a value that is being posted in a form. Not only can this be applied to values within a form, but also it can be embedded into a string that will form a query string hyperlink.

```
<TBODY>

<xsl:for-each select="structure/class">
        <xsl:variable name ="ClassID" select="ClassID"/>
        <tr>
                <form action="extract.aspx" method="post" name="extract">
                        <TD>
                                <input type="hidden" name="ClassID" value="{$ClassID}" />
                                <input type="submit" name="extract" value="Extract" />
                        </TD>
                </form>
                <TD>
                        <xsl:value-of select="Rating"/>%
                </TD>
                <TD>
                        <xsl:value-of select="ClassID"/>
                </TD>
                <TD>
                <a href="viewclass.aspx?ClassID={ClassID}">
                        <xsl:value-of select="Classname"/>
                </a>
                </TD>
                <TD>
                        <xsl:if test="Package!=';'"><xsl:value-of select="Package"/></xsl:if>
                </TD>
```

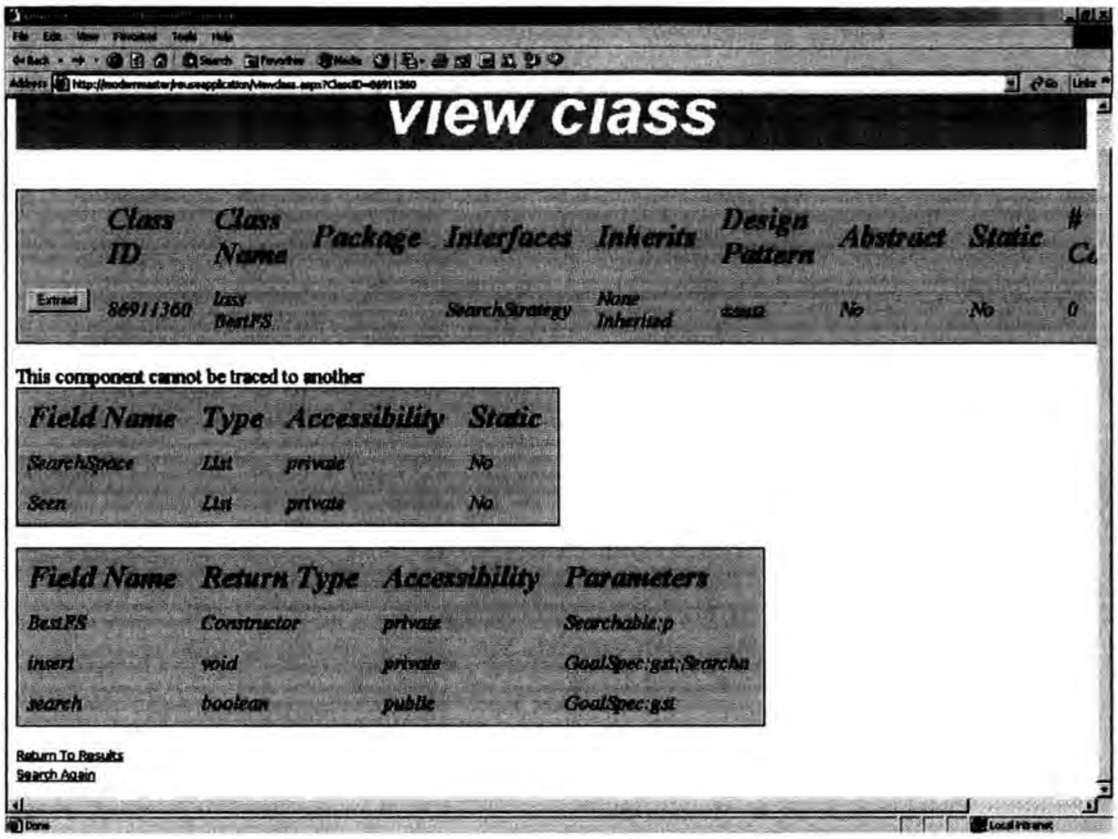
Figure 4.6-7: Sample of '*viewsearchresults.aspx*'.

```
                <TD>
                        <xsl:if test="DesignPattern!=';'">
                                <xsl:variable name ="DesignPattern" select="DesignPattern"/>
                                <a href="viewdesignpattern.aspx?designpattern={$DesignPattern}"><xsl:value-of select="DesignPattern"/></a>
                        </xsl:if>
                </TD>
                <TD>
                        <xsl:choose>
                                <xsl:when test="Abstract[. !='0']">
                                        Yes
                                </xsl:when>
                                <xsl:otherwise>No</xsl:otherwise>
                        </xsl:choose>
                </TD>
```

Figure 4.6-8: Sample of '*viewclass.aspx*'.

The "*xsl:for-each*" XPATH expression iterates through an XML source for every occurrence of the select value. In Figure 4.6-7, the select value is "*structure/class*". If this was applied to the XML in Figure 7.4-2 in the Appendix Section 4, at every iteration, the XSL processor would extract the group of children tags associated with this value. In the ReSULT system the main use of this expression was to iterate through an XML source so that individual items, such as fields, components or classes, are translated and placed as individual records in a HTML table.

The XPATH expressions *choose*, *when*, and *otherwise* perform a similar to an "if, if-else, else" conditional statement. This gives the programmer flexibility of what can be outputted for the user. The main use of this functionality in the ReSULT system is the translation of binary data held inside the system into a textual yes or no format. This is displayed in Figure 4.6-8, where the value for the abstract field depends upon whether the input is a '0' or '1'.



	Class ID	Class Name	Package	Interfaces	Inherits	Design Pattern	Abstract	Static	# C
Extract	86911360	lass BestFS		SearchStrategy	None Inherited	[illegible]	No	No	0

This component cannot be traced to another

Field Name	Type	Accessibility	Static
SearchSpace	List	private	No
Seen	List	private	No

Field Name	Return Type	Accessibility	Parameters
BestFS	Constructor	private	Searchable:p
insert	void	private	GoalSpec:gst;Searcha
search	boolean	public	GoalSpec:gst

Figure 4.6-9: A screenshot of the '*viewclass.aspx*'.

```
.BackgroundTitle {
        font-family: Arial, Helvetica, sans-serif;
        font-size: 70px;
        font-style: italic;
        line-height: normal;
        font-weight: bolder;
        font-variant: normal;
        text-transform: lowercase;
        color: #FFFFFF;
        background-color: #6BB7FF;
        letter-spacing: normal;
        text-align: center;
        word-spacing: normal;
}
```
Figure 4.6-10: A fragment of the '*normalstyle.css*' file.

The CSS file 'normalstyle.css' provides additional HTML formatting to web pages in the ReSULT system. An example of this is shown in Figure 4.6-9; this provides the developer the opportunity to reuse HTML styles in a number of different web pages by calling its class name. The example in Figure 4.6-10 has a class name '*BackgroundTitle*'. The result of applying this formatting is apparent in Figure 4.6-9.

The highlighting of tables within the system (seen in Figure 4.6-9) is performed by Jscript coding. This is located within an HTML component, and is linked to each web page that uses tables.

4.7 Summary

This chapter begins with an insight of the use-cases defined in chapter 3. Following on from this, the system architecture is proposed with a detail specification for the hardware and software used. A description of the objects used within in the system is further defined. This includes the structure of objects used within the ReSULT system and the tables used within the database.

The next section of the design elaborates the use-cases identified during the design stage. There are two areas for discussion within the design for insert code use-case; these are insert code framework, and keyword analysis. Within the design for the use-case for identifying and selecting components the keyword ranking algorithm is defined. The final use-case is expanded and developed for extracting components.

Within the final section of the design chapter, the focus is transferred to the design of web services architecture and application. An insight is given into how data is organised and displayed within the web application.

Chapter 5 Case Study

5.1 Introduction

The previous chapters of this thesis have displayed how the architecture and processes of the ReSULT reuse system came about. From early on the emphasis of this system was focused upon three different processes identified within current reuse practices (Section 3.3.2 and 3.3.3); these are inserting a component, identifying a reusable component, and extracting a component. This chapter will describe how the ReSULT system will be tested and evaluated against these three areas of the system.

For any system to be tested and evaluated, a framework must be chosen that reflects the goals of the system. The second section of this chapter highlights the experimental framework taken in this research. Using the experimental framework chosen, metrics are defined that corresponds to this. These are outlined in the Appendix Section 9.

Section 5.4 concentrates on the application of the ReSULT reuse system in a case study. This case study describes the process of how the system is to be used in a small fictional company, and has been chosen to accurately reflect the uses of this system in a software engineering company that is applying software reuse to their software development processes. From the metrics obtained from this case study, critical evaluation of the system can begin in the next chapter.

5.2 Modelling Software Quality

Improving software quality is one of the aims of dealing with the present software crisis [Paulk95]. Developers can aim towards improving software quality by being objective towards it, and measuring their performance.

The main goal of a software measurement process is to satisfy certain information needs by identifying entities, and the attributes of these entities. The attributes of software are classified as internal and external attributes. An example of an external attribute is 'reliability'. This also replies on the environment and users. From a user's point of view, the quality of the software is regarded as the external attributes of the product.

Over the years a number of approaches have been defined to analyse the quality of software. The external attributes that are regarded by the user are hard to objectively measure, and thus evaluation becomes difficult. To gain effective evaluation a relationship is established between the external and internal attributes so that software measurements can be taken.

The measurement principles are formulation, collection, analysis, interpretation, and feedback. There is often confusion with regards to terms metrics and measurement. Lorenz et al defines the terms as follows "Metrics is a standard of measurement used to judge the attributes of something being measured, such as quality or complexity, in an objective manner. On the other hand, Measurement is the determination of the value of a metric for a particular object. Therefore, considered with those definitions, the term, measurement should be used, when mentioned about the activity itself to measure something" [Loren94].

In order to evaluate software quality quantitatively and qualitatively, metrics are established to measure the software's quality from defined quality metric models. The majority of quality models are hierarchically based, such as McCall et al [McCal77]. A number of problems with these models were identified by Mei et al [Mei02]. They addressed a number of problems with the traditional hierarchical quality model:

- *Which quality criteria should be included into the metrics model?*
- *What relationship between these quality criteria?*
- *Which metrics should be associated with the quality criteria?*
- *How to combine the values of these metrics to derive the value of quality criteria?* [Mei02].

In practice these problems become more complex as certain metrics are associated with several quality criteria. This generates the possibility of a metric being positive in one criterion but negative towards another. There are also conflicts between quality criteria when analysing maintainability and efficiency.

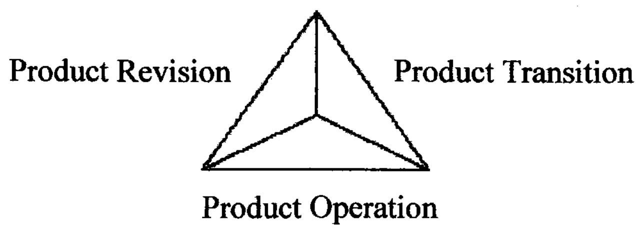


Figure 5.2-1: McCall's Triangle of Quality Factors.

For this research, the McCall software quality model [McCal77] is used in the undertaking of evaluation. This model is an extension to the hierarchical Goal-Question-Metric model that is generally adopted as a basis of software evaluation. The main principle of the McCall model is for a system to be split into three areas (Figure 5.2-1). Each area is then decomposed into a set of measurable properties, which themselves can be decomposed into a set of criteria for metric assessment.

Product Revision

- *Maintainability (M_a) (Can I fix it?)*
- *Flexibility (F_l) (Can I change it?)*
- *Testability (T_e) (Can I test it?)*

Product Transition

- *Portability (P_o) (Will I be able to use it on another machine?)*
- *Reusability (R_{eu}) (Will I be able to reuse some of the software?)*
- *Interoperability (I_O) (Will I be able to interface another system*

Product Operation

- *Correctness (C_o) (Does it do what I want?)*
- *Usability (U_s) (Can I run it?)*
- *Efficiency (E_f) (Will it turn on use hardware as well as it can?)*
- *Reliability (R_{el}) (Does it do it accurately all the time?)*
- *Integrity (I_n) (Is it secure from external attacks?)*

The criteria can be attained from using a set of software metrics.

5.3 Software Metrics

For this research, the goal of using software metrics is to help evaluate the quality of the ReSULT reuse system. Metrics are divided into two independent groups, direct and indirect. Immediate measurable attributes such as lines of code or execution speed are classed as direct. Other metrics that are not immediately quantifiable e.g. functionality, or reliability are categorised as indirect. These are highly subjective and difficult to measure. The measurement of software quality for the ReSULT reuse system includes both approaches.

As described in Section 5.2, metrics are classified into two groups direct, or indirect. Whichever group a metric is categorised into, it must hold true for the following properties for the metric to be a good objective evaluator.

- *Attributes of effective software metrics*

- *Simple and computable*
- *Empirically and intuitively persuasive*
- *Consistent and objective*
- *Consistent in units and dimensions*
- *Programming language independent*
- *Effective mechanism for quality feedback* [Berto04]

It is extremely important that these properties are held within metrics. Often quality factors scores are highly subjective and are open to questioning by the reader.

In the Appendix Section 9, the quality factors declared in Section 5.2 are decomposed into metric based criteria that will be used to evaluate the ReSULT system within a scenario based case study.

5.4 Case Study

5.4.1 Scenario Based Case Study

The objective of this case study is to evaluate the model defined in the ReSULT model, and to gain metric values that indicate successes and failures of this prototype system. In order to demonstrate the strengths of the ReSULT system, twenty problem statements have been defined to produce results that display the effectiveness of the searching and retrieving mechanism, and will be tested on five undergraduate and five postgraduate students to prove how successful it is.

The objectives of the case study are to:

- *Show how the ReSULT system developed in this research is applied in practice during a software lifecycle.*
- *Identify what search settings produce optimal search results, by repeating the application of the case studies.*
- *Provide performance data concerning the web services involved in the system (using the Web Application Stress Tool by Microsoft).*

5.4.2 Preparation for the Case Study

The goal of this case study is to identify the success of the searching mechanism, and of the inserting process. To prepare for the testing of the insertion process, one hundred Java scripts were developed from a mixture of correct and incorrect solutions to first year undergraduate programming practical work. With these scripts their OCL representations are also produced. These were then inserted into the ReSULT reuse system. With this populated system, a user has the ability to enter search criterion into system to obtain search results.

To reflect upon how this system would be used within a company, an approach is taken in this research that corresponds with normal practices used inside software engineering companies. These practices focus upon the software lifecycle waterfall model [Jacob99]. This includes stages such as requirements engineering, design, implementation, testing, and maintenance. The ReSULT reuse system is effective during the design and implementation stages of this lifecycle. To devise a testing strategy for the searching mechanism of this system, an analysis into what outputs are received from the previous stage of the lifecycle is needed.

A triangle is made up of three sides. Create options that enable you to initialise a triangle by entering the length of its three sides. Test whether the lengths of only three sides have been entered, get each side and identify the area of the triangle, test whether the sides of the triangle are equal to each other, and from these result identify whether the triangle is scalene, isosceles or equilateral.

Figure 5.4-1: Example problem statement.

During requirements engineering, a description of the problem is composed that defines 'what' the problem is, and not 'how' it can be addressed. From this statement, functional and non-functional requirements are gathered. For the ReSULT system, there is no concern for functional or non-functional requirements, only the problem statement is used. This statement is analysed, and broken down into design modules using a concept called 'Class, Responsibility, and Collaboration cards' (CRC) [Beck89]. CRC cards follow on from the previous stage of the software lifecycle by describing 'how' a problem is to be solved.

Class name: Triangle	
Super class:	
Subclasses:	
5.4.2.1.1 Responsibilities	**5.4.2.1.2 Collaborations**
Initialise(side1,side2,side3)	Integers
All side lengths entered?	Boolean
Get area of triangle	Real
Are three sides equal to each other?	Boolean
Are two sides equal to each other?	Boolean
Are all sides unique in length?	Boolean

Table 5.4-1: Example CRC card.

Table 5.4-1 displays the resulting CRC from the problem definition in Figure 5.4-1. There are three sections of CRCs; these are Class name, responsibilities, and collaborators. CRCs are conceived from their problem definitions by identifying the nouns and verbs within them. The responsibility field of a CRC is populated by verbs found; these are implemented as public functions during the implementation. The collaborators field identifies nouns that represent objects within the system.

The data held in the CRC displayed in Table 5.4-1 is used as search criteria. However, there is a need to be selective upon what information can be used as search criteria in the ReSULT system. For this, it has been decided to use the class name, super class, subclasses, and one adjective from each responsibility listed. The words selected from the CRC example are double underlined in Table 5.4-1.

To provide a quantitative analysis of the search mechanism, the process is repeated on nineteen other problem statements that are based upon nineteen questions from the first year programming practical work, so that a mean average can be obtained.

5.4.3 Problem Statement for the Case Study

The following problem statement is a fictional account that displays how the ReSULT system could be introduced into an organisation.

A fictional software engineering company (Amberwood Engineering) has researched into the advantages and disadvantages of integrating a reuse strategy into its current processes. The findings from this research highlighted the potential economic gains and the overall improvement in software quality.

Amberwood specialise in producing diary systems for government departments and blue-chip companies. These systems often have many similarities, and in the past programmers have copied and pasted code from their own code to develop new applications. The introduction of the ReSULT reuse system brought about the calculation for the total number of scripts (excluding different versions, and design documents) that had been developed through the lifespan of the company; this amounted to one hundred scripts.

By introducing ReSULT reuse system, it is hoped that it would help influence the spread of knowledge through the company, and increase the efficiency of developing solutions for clients. This relies on the searching mechanism of the ReSULT system to produce accurate results. The criteria used for searching is obtained from the problem definition defined at the beginning of any contract.

5.5 Quality Factor Scoring

From the metrics defined in the Appendix Section 9, the application of the quality factors that are defined in McCall's software quality model [McCal77] is performed. The equations defined in Table 7.4-4 of the Appendix Section 9 show how the individual metrics for each quality factor are calculated. A key for the acronyms used in these equations can be found in Table 7.4-3 of the Appendix Section 9.

5.6 Summary

In this chapter there is a description of how the ReSULT system will be evaluated. A software quality model will be used to define specific quantitative and qualitative criteria to measure from the system. It was decided to use McCall's quality model to do this.

The second section of this chapter details the case study that will be used to evaluate the ReSULT system using McCall's software quality model. The applied fictional scenario consists of a software engineering company that produces reusable components, and also reuses these components in current projects.

The following chapter evaluates the results produced from this chapter.

Chapter 6 Results and Evaluation

6.1 Introduction

This chapter evaluates the results of the work described in this thesis. The results of the research which has been conducted are evaluated in two main sections that correlate with McCall's software quality factors [McCal77].

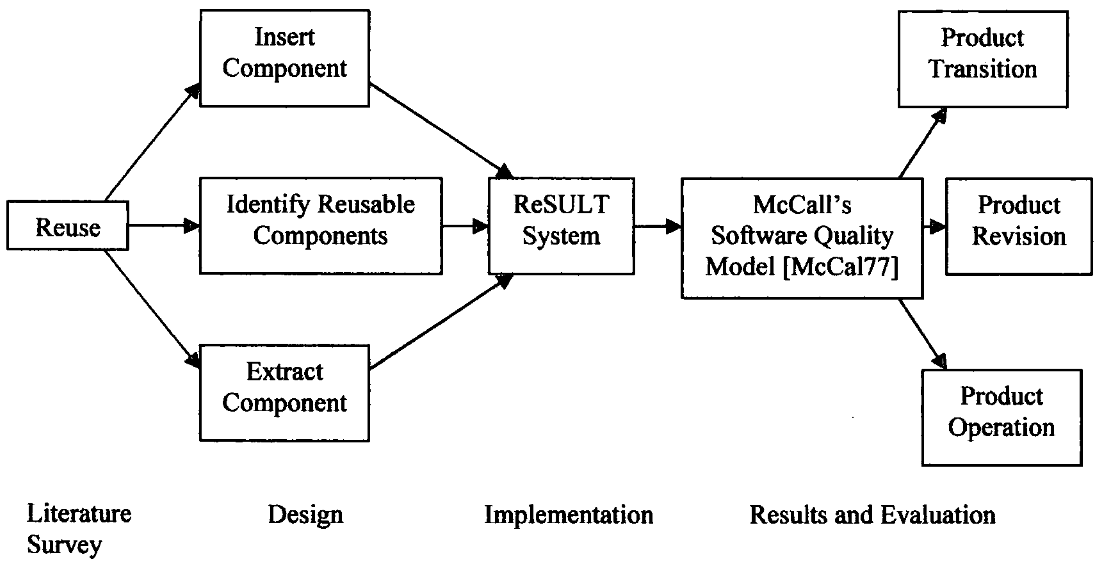


Figure 6.1-1: An abstract representation of the layout contained within this work.

The first section evaluates the issues involved with introducing the ReSULT system into a fictional scenario. This evaluates how the toolset supports reuse, changes that may occur to an organisation, and what benefits are brought from the introduction of the ReSULT system.

The second section looks more in depth into the ReSULT system and evaluates the operations within it. This takes into account the usability, performance, component integration, and error tolerance of the ReSULT system.

6.2 Transition Issues when Introducing ReSULT into an Organisation

6.2.1 Using the Toolset to Support Reuse

For reuse to be successful, a defined strategy must be put into place for reuse to occur. The integration of reuse into an organisation must take into account the components that are already present within an organisation, while defining standards for the developing of future components.

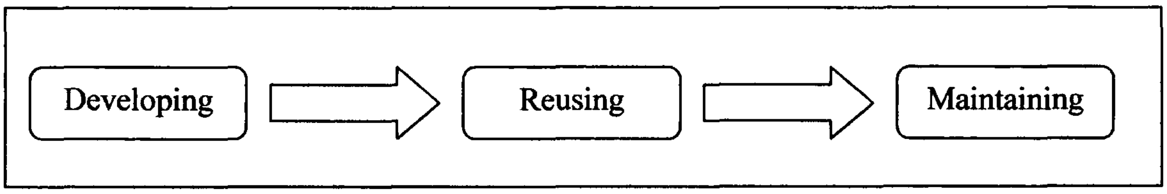


Figure 6.2-1: The lifecycle of reusable components.

Figure 6.2-1 displays the lifecycle of reusable components. The following sections will describe how the ReSULT tool was applied to each stage of the lifecycle.

6.2.1.1 Developing Components

When developing components for use within the ReSULT system consideration must be made towards the number of comments added to the component. This aids the system in two ways; firstly, it helps a reuser understand important beacons within a component (Section 2.3.1), and secondly, it is the extraction of these beacons from the code that provides a mechanism for classifying components.

Components that are designed for reuse must be self contained units of code with descriptive entities, such as fields and methods, which complement the comments made to them. Using a naming convention that totally avoids the context of which the component is a part of will not aid the reuser, or the ReSULT system into identifying a satisfactory component.

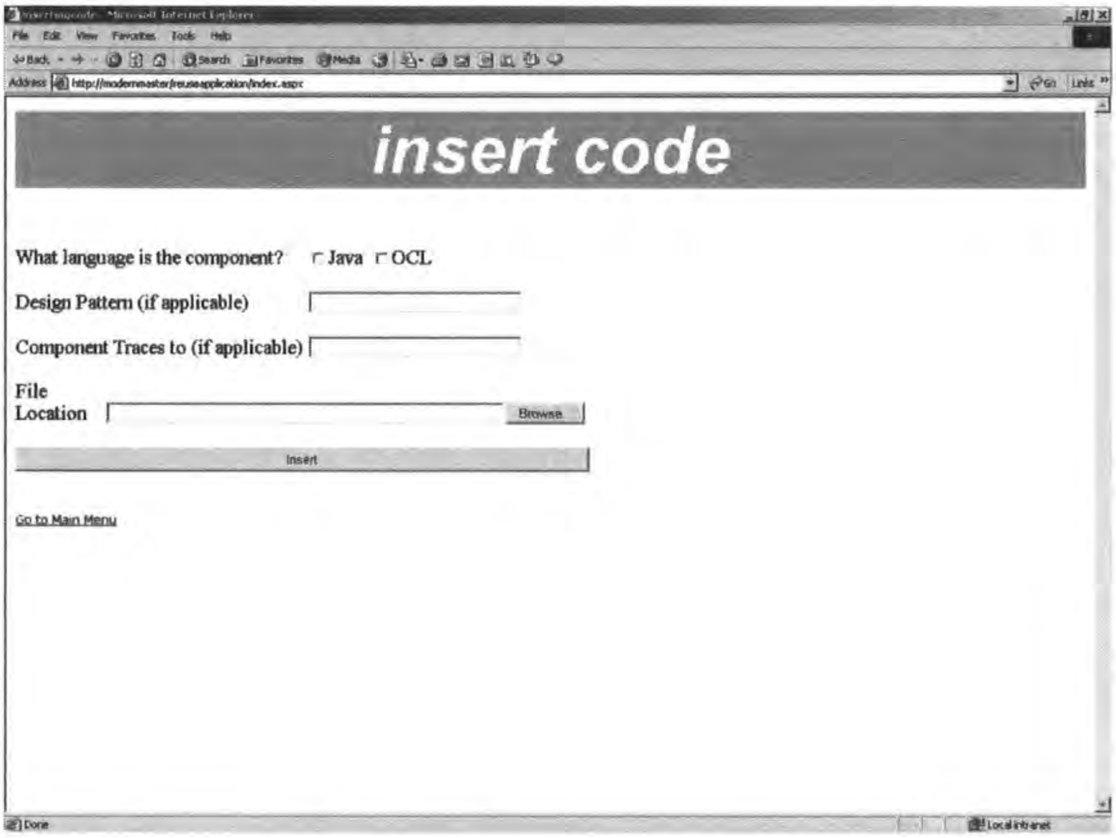
The ReSULT is not a compiler of any sort, nor is it an environment where code is spawned. The ReSULT system is purely a tool where text-based files are processed by identifying patterns within files and extracting information. The downfall of this is that the system cannot identify incorrect syntax from the correct format. It is down to the developer to create scripts in their desired environments. From these environments, the script can be compiled to identify errors. Once a script is identified as error free, it is ready to insert into ReSULT.

One key feature of the ReSULT system is that it provides the ability of identifying Java components that have been produced from an OCL component, or displays the trace from the Java component to an OCL component. When inserting a component into the system, the developer is given the option of inserting a code identification number that this component traces from. This approach may lead to errors with inserting the correct code identification number. Components that trace each other may not be added to the system at the same time or by the same developer. As time passes, the probability of the correct class identification number being added into the system falls. The ReSULT system does not offer a 'lookup' facility for the developer and in doing so, the developer changes his/her role into a 'reuser' within the ReSULT system to identify which component the current work has originated from.

6.2.1.2 Reusing Components

The second mode of operation is when a developer is searching for a component that they want to maintain, or include in their current system. In Section 2.2.2, there are a number of preconditions that must be met in order for a developer to be able to incorporate components into their software system. These preconditions are listed below, along with the support which ReSULT provides for each level.

1. The component must exist.


insertingcode - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://modernmaster/reuseapplication/index.aspx
Go Links
Browse...
Insert
Go to Main Menu
Done
Local intranet

insert code

What language is the component? Java OCL

Design Pattern (if applicable)

Component Traces to (if applicable)

File Location

Figure 6.2-2: A screenshot of '*insertcode.aspx*'.

The ReSULT system gives the opportunities for developed Java and OCL components the opportunity to be inserted into the system. Figure 6.2-2 displays the form that gives the ability for users to insert components into the system.

2. The component must be available to the developer.

ReSULT system enables developers to store components in a reuse repository.

3. The developer must be able to find the component.

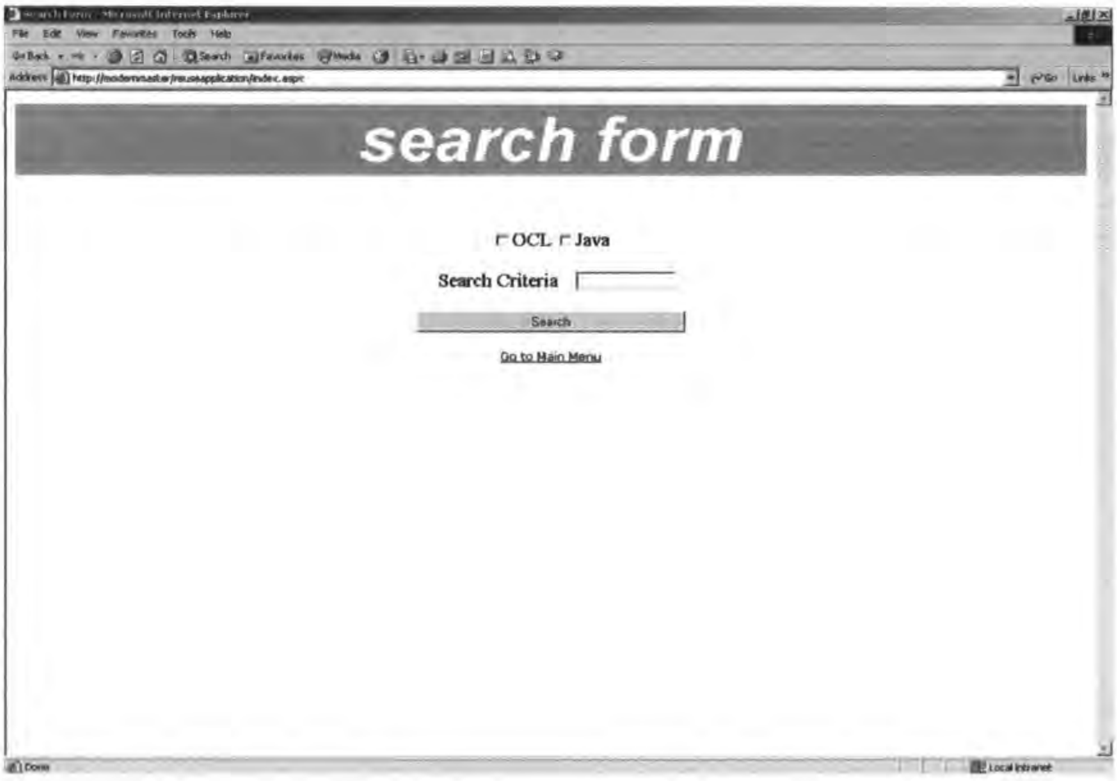
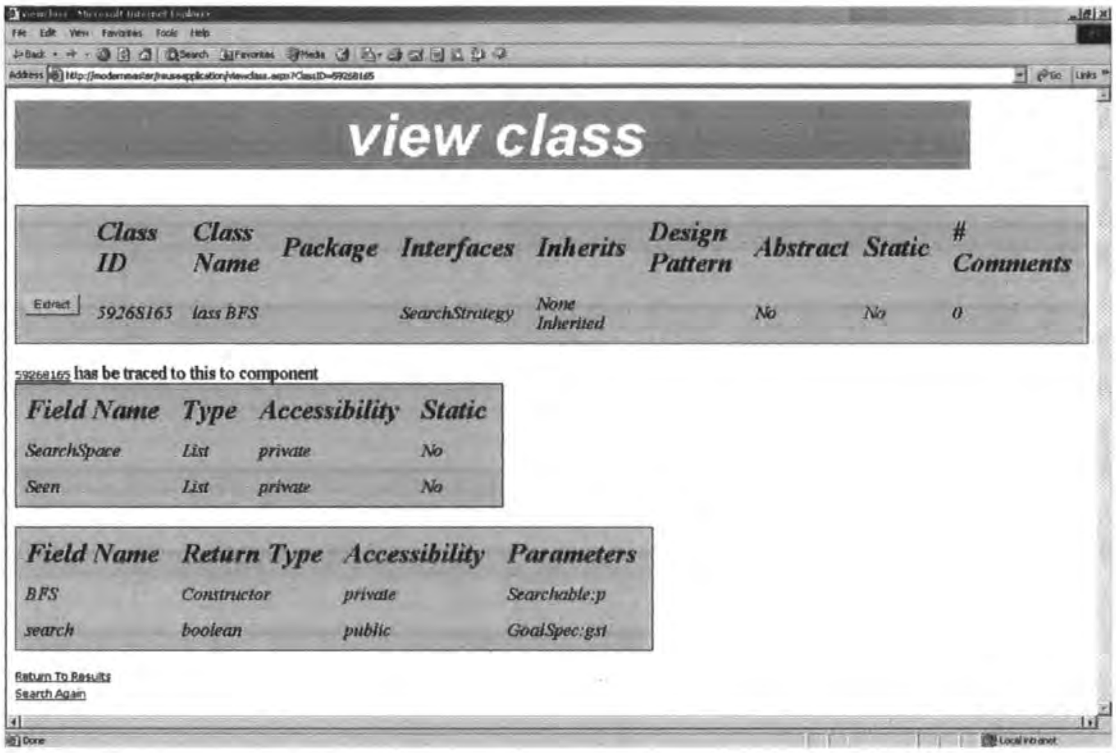


Figure 6.2-3: A screenshot of '*searchform.aspx*'.

The ReSULT system stores as well as the component, an abstract representation of it. This abstract representation involves data categorised by the means described in Section 4.4. This provides a reuser with a multi-faceted searching mechanism for examining a reuse repository by keyword search criteria. This provides a multi-faceted searching mechanism that is used by a reuser by entering keywords as searching criteria. Figure 6.2-3 displays the keyword search form that allows the user to select either a search for Java or OCL components.

4. Once found, the developer must be able to understand the component.



view class

	Class ID	Class Name	Package	Interfaces	Inherits	Design Pattern	Abstract	Static	# Comments
Extract	59268165	lass BFS		SearchStrategy	None Inherited		No	No	0

59268165 has be traced to this to component

Field Name	Type	Accessibility	Static
SearchSpace	List	private	No
Seen	List	private	No

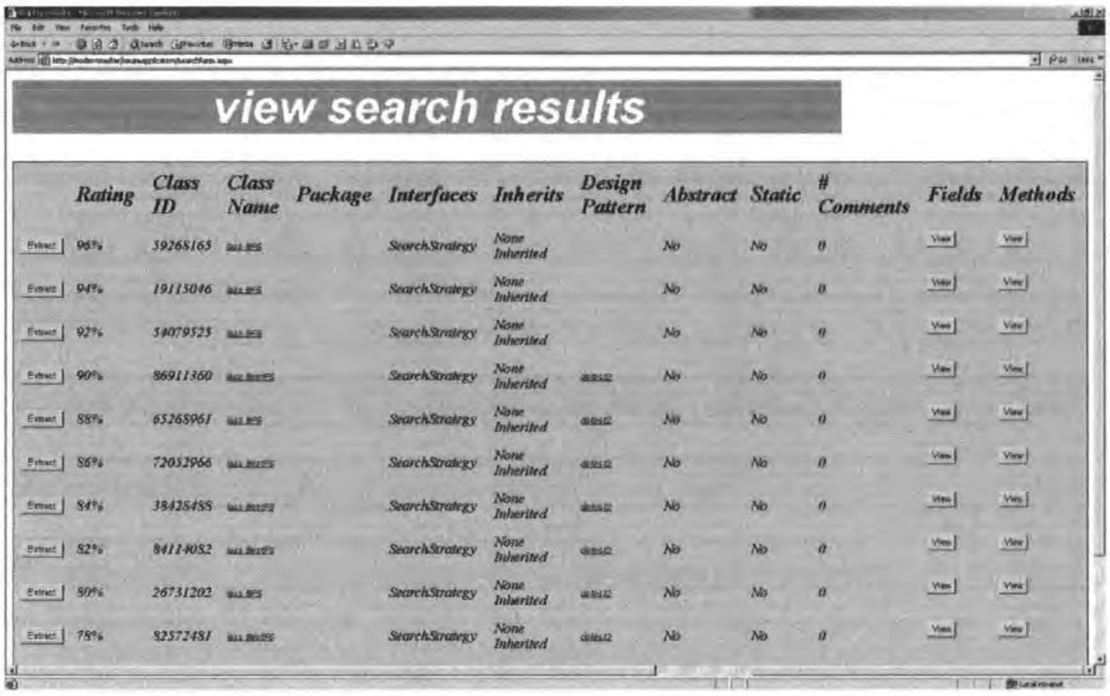
Field Name	Return Type	Accessibility	Parameters
BFS	Constructor	private	Searchable:p
search	boolean	public	GoalSpec:gst

Return To Results
Search Again

Figure 6.2-4: A screenshot of '*viewclass.aspx*' (note: picture has had to be merged because of the page being to large for the screen).

The ReSULT system displays the selected component in the same format as which it was inserted into the system. The system does not remove any indents or line spaces from the component, thus keeping to a uniform structure that is readable and easily interpreted by a reuser (Figure 6.2-4).

5. **Based on an understanding of the component, the developer must identify the component as being valid for the current system.**



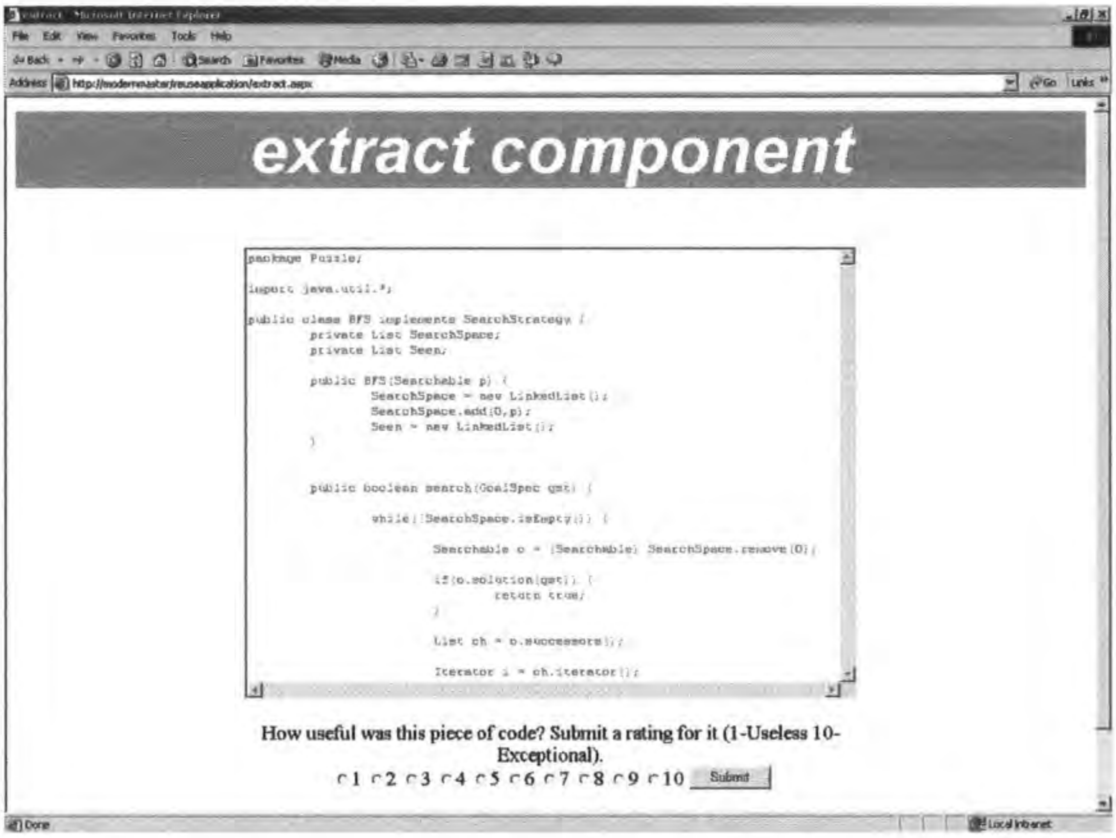
view search results

	Rating	Class ID	Class Name	Package	Interfaces	Inherits	Design Pattern	Abstract	Static	# Comments	Fields	Methods
Extract	96%	59268165	[illegible]		SearchStrategy	None Inherited		No	No	0	View	View
Extract	94%	19115046	[illegible]		SearchStrategy	None Inherited		No	No	0	View	View
Extract	92%	54079525	[illegible]		SearchStrategy	None Inherited		No	No	0	View	View
Extract	90%	86911360	[illegible]		SearchStrategy	None Inherited	[illegible]	No	No	0	View	View
Extract	88%	65268961	[illegible]		SearchStrategy	None Inherited	[illegible]	No	No	0	View	View
Extract	86%	72052966	[illegible]		SearchStrategy	None Inherited	[illegible]	No	No	0	View	View
Extract	84%	38428488	[illegible]		SearchStrategy	None Inherited	[illegible]	No	No	0	View	View
Extract	82%	84114052	[illegible]		SearchStrategy	None Inherited	[illegible]	No	No	0	View	View
Extract	80%	26731202	[illegible]		SearchStrategy	None Inherited	[illegible]	No	No	0	View	View
Extract	78%	82572481	[illegible]		SearchStrategy	None Inherited	[illegible]	No	No	0	View	View

Figure 6.2-5: A screenshot of '*viewresults.aspx*' (note: picture has had to be merged because of the page being to large for the screen).

The ReSULT system provides a list of candidates for a reuser to choose from (Figure 6.2-5). This list contains information that is associated with the component within the system (detailed in Section 4.4); this helps the reuser define a mental model of what the component does. From this model, a comparison is made with their image of that the ideal component should be, and a decision on the desirability of that component is then produced.

6. **The developer must be able to successfully integrate the component into the current system.**


extract - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://modernmaster/reuseapplication/extract.aspx
Go Links
extract component

```
package Puzzle;

import java.util.*;

public class BFS implements SearchStrategy {
        private List SearchSpace;
        private List Seen;

        public BFS(Searchable p) {
                SearchSpace = new LinkedList();
                SearchSpace.add(0,p);
                Seen = new LinkedList();
        }


        public boolean search(GoalSpec gst) {

                while(!SearchSpace.isEmpty()) {

                        Searchable o = (Searchable) SearchSpace.remove(0);

                        if(o.solution(gst)) {
                                return true;
                        }

                        List ch = o.successors();

                        Iterator i = ch.iterator();
```

How useful was this piece of code? Submit a rating for it (1-Useless 10-Exceptional).

○1 ○2 ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10 Submit

Figure 6.2-6: A screenshot of '*extractcomponent.aspx*'.

The ReSULT system offers the reuser the opportunity to copy and paste the selected component into their existing system (Figure 6.2-6).

6.2.1.3 Maintaining Components

Maintenance of the software products within the ReSULT system takes two forms, an existing component that is converted to a reusable component, or a component that is already present within the system that needs to be updated due to performance, functionality, or error reasons (Section 2.2.2).

When converting existing components for use within the ReSULT system, a uniform approach must be developed. This approach consists of the examination of components for specific properties, and updating the component where this property is not met. These properties are listed below.

- *A significant number of comments present.*
- *The inclusion of both inline and prologue comments.*
- *High cohesion.*
- *Low coupling with other components.*
- *A definite naming scheme for fields, and methods.*

This approach ensures that an accurate representation is taken for each component inserted; therefore, improving the accuracy of the searching mechanism within the ReSULT system.

In current practices, the majority of time that is taken up within the software lifecycle is during the process of maintenance [Timen89]. As discussed in Section 2.3, an estimate of 50% of all maintenance effort is placed within the process of understanding the code that is being maintained. Current practices involve maintainers analysing documentation to determine an understanding. The ReSULT system is not designed to replace the need for documentation (good documentation is always a sign of a good system), but if the documentation is not up to date, the

ReSULT bridges the gap in knowledge by displaying the structural features present within a component and the component it traces from.

6.2.2 The Problems Faced by the Company in Implementing the ReSULT system into its Existing Process

When an organisation already has introduced a reuse policy into its business processes, it is harder to integrate the ReSULT system without needing to refine the system's design to match the current reuse process. Lets take for example, an organisation that depends upon a reuse repository with a change configuration management (CCM) system; currently the ReSULT system does not have CCM facilities. Versions of the same component are stored within the system, but the ReSULT system does not give any indication of which is the current version. One benefit of reusing code is that as more reusers develop using the same component, the likelihood is that any underlying errors will come forth and cause new versions of the component being produced without these errors. This is examined further in Section 7.4 'Suggestions for Future Research'.

6.3 An Evaluation of the Operation of the Toolset

6.3.1 Usability

Usability is an important concept that has to be investigated within the introduction of any system. Seamless integration of a new system into an organisation is desired by any software developers, but is often never achieved [Kwon98]. From a user's aspect, seamless integration of a new system is warranted by the measurement of a number of factors. These factors are measured by metrics M9.1 through to M11.2 in Appendix Section 9 inclusive, and cover the usability factors screen design, error tolerance, users' expectation, suitability, documentation, and training.

Users Expectation

One of the most underestimated aspects of what is misjudging the users' expectations, and misinterpreting what the system will do. A system's level of acceptance is decided by its users. By analysing precisely what features the ReSULT system performs for reusers to the needs determined in Section 3.2 and 3.3, a value for the metric M10.2 is obtained for evaluation.

Training Strategies/Intensive Training

Users are often unsure about the introduction of new systems to their work practices. Management have to consider appropriate training strategies that details how new users should use the system while stimulating confidence with the new system. With the introduction of the ReSULT system, the content of a training strategy has to consider two factors, training with reuse, and training for reuse.

Training for reuse considers the introduction of the ReSULT system into an organisation where reuse is not present. To aid the integration of the ReSULT system into an organisation, the teaching of reuse is performed using the ReSULT system.

The fundamentals of reuse are listed in Section 2.2.1; these must be thoroughly incorporated into the training scheme. The principles, methods and skills required to develop reusable software cannot be learned effectively by generalities and platitudes. Instead, developers must learn concrete technical skills and gain hands-on experience during training [Schmi99].

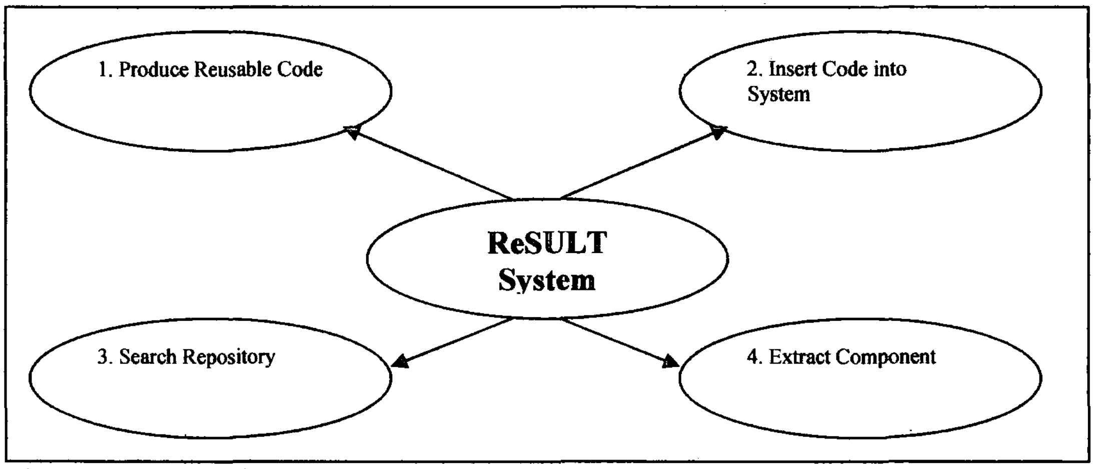


Figure 6.3-1: The key processes within the ReSULT system.

When an organisation already possesses some form of reuse within its corporate boundaries, training strategies are designed to aid the transfer of knowledge from the current system to the new system that is being introduced. Training strategies are looked upon as the formation of links between key processes between the two systems. Within the ReSULT system, there are four key processes (shown in Figure 6.3-1).

Without components to reuse, there would be no reuse system. In an unorganised system of reuse, there would be many standards of code production. Engineers would selectively remember useful segments of their code, and reuse them within their own projects. Reusable code would not be produced through the performing of problem domain analysis, but instead code would be designed for a specific role within a project. The chance of this role being generic through many different projects is low.

It was discussed in Section 2.2.2 that effective reuse is brought about by successful problem domain analysis, and by producing components that are loosely coupled and highly cohesive. In addition to these high level properties, low level properties such as an appropriate naming schema for variables, methods and classes, which refers clearly to the problem domain, is applied to components.

The ReSULT system brings to an organisation a central storage for reusable components. Old practices of only storing their work on either restricted access network spaces, or individual machines should be removed from the workplace to gain the most out of software reuse that is aided by the ReSULT system. Training for this process must focus upon diminishing the culture of the 'not invented here' syndrome within an organisation by focusing on uniform standards of code production.

Software engineers follow a different approach towards selecting components for reuse. In Section 2.2.2, an analysis of the different approaches used in the selection of components was taken to understand the details of this process. From the findings in this section, an approach was taken that took into an account how a system is developed from an initial problem statement. This approach was taken during the gathering of experimental data, and is described in Section 5.4.3.

The extraction of a component from the ReSULT system relies upon the user copying the component from the web browser, and pasting it into their project. This formed a simple method of transferring components from the centralised repository to individual projects. Within primitive approaches to reuse, approaches such as copy and pasting are familiar practices (Section 3.3.1.1). The approach used in ReSULT expands this practice by enabling developers to copy and paste components from other developers instead of not just from their own reusable code; therefore, the need for additional training is limited.

6.3.2 Performance

When evaluating the performance of the searching mechanism, a number of different factors are observed that may lead to a downgrading of performance from the search mechanism. These factors are:

- *The number of users connected to the web application server.*
- *The number of concurrent searches made.*
- *The amount of data held within the database.*

6.3.2.1 Web Application Efficiency

The symptoms of an under-performing web application server are highly noticeable to the user, and affect their confidence towards the system. Users will wait about 10 seconds for a page to download, sometimes 15 seconds before they lose interest

[Web07]. For the ReSULT system, the Internet Information Service web service was used to handle the requests for the C# .NET web application.

Connections	1	2	3	4	5	6	7	8	9	10	μ
Average Response Time (mS)	5714	11428	17142	22857	28571	34285	39999	45714	51428	57142	31428

Table 6.3-1: The results of the average time taken for the browser to response after a request.

Within the Windows 2000 operating system, this software is hard coded to accept no more than ten connections. Table 6.3-1 shows the time taken for the '*index.aspx*' of the ReSULT application to be returned to the requesting browser when ten simultaneous requests are made for that page. '*index.aspx*' is a static web page that greets the user when they initially reach the web application. The goal of this experiment was to measure the performance of the ASP. NET worker process within IIS (Section 4.3.1).

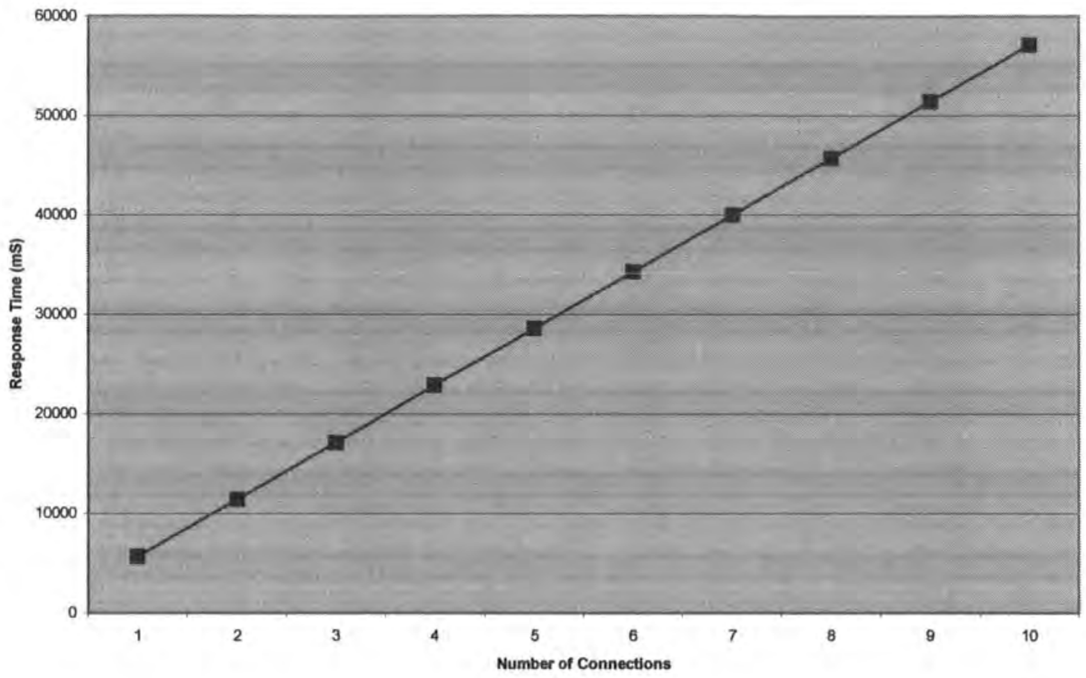


Figure 6.3-2: The relationship between the response time and number connections to IIS.

Microsoft's Web Application Stress (WAS) tool simulates multiple clients attempting to connect to web applications and services. The results shown in Figure 6.3-2 display a distinct linear relation between the response time of the ReSULT web application and the number of connections. This ensures that as the numbers of connections grow, the response time does not increase exponentially. It has to be remembered that a limit is placed upon the number of connections accepted by IIS at any one time within Windows 2000. This is set is to ten, and because of the small number of connections the chance of a downgraded performance from the test system is highly unlikely. Ten concurrent connections are appropriate within any small organisation, but it would not be acceptable to any larger distributed organisation with which this product is aimed towards.

6.3.2.2 Web Service Performance

The underlying architecture of the system consists of three web services that perform the following processes, inserting, searching, and extracting components. Each of these web services are based upon XML. This ensures interoperability within heterogeneous networks of a distributed organisation. However, when compared to traditional web interactions, the requests and replies are much larger for XML transaction because of the need to parse XML code; this adds additional server overhead [Tian02].

To test the performance of these three web services, performance monitors were placed at each web service. These monitors measured the time taken for a request to receive a response from the web service. To remove the possibility of losing packets during transmission testing is performed on the same machine as the web application, and web services. A mean value is taken from fifty requests made from the ReSULT web application.

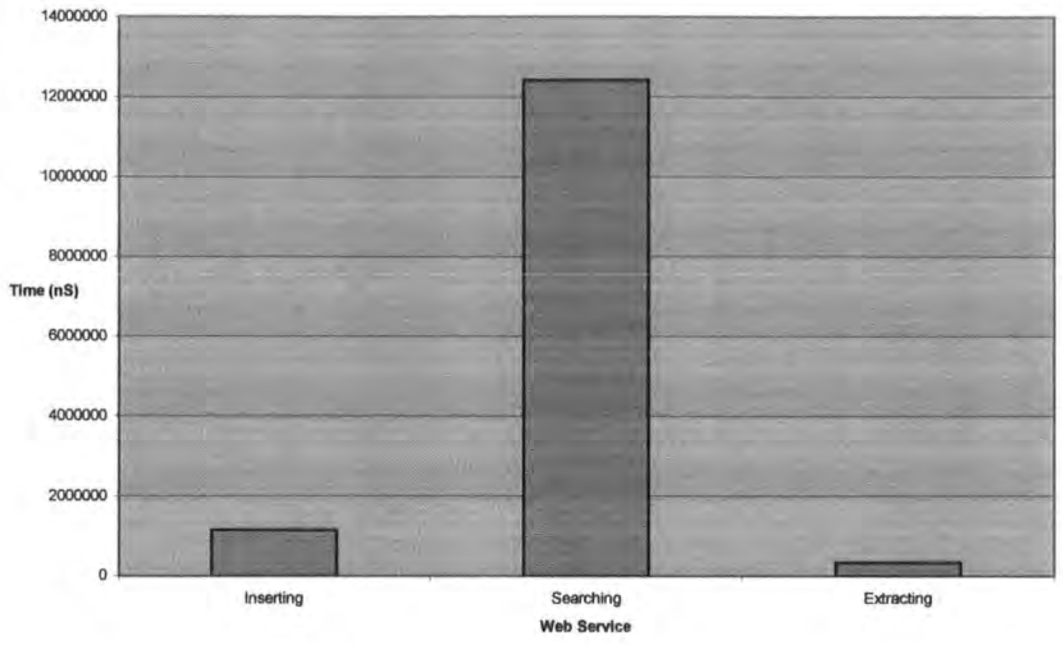


Figure 6.3-3: The mean times taken for the three web services to give a response to a request made from the ReSULT web application.

There are a number of factors that may cause web services to decrease in response time, such factors include the number of connections, bytes transferred, or the amount of computation needed; these factors must be considered when evaluating the web services that are present within ReSULT (these are listed in Section 4.5). To control the affects of network latency on the results, all web services and the web application were hosted on the same machine.

For the web service '*extract component*', the web service performance depends upon how much data is transferred between the MySQL server and itself. The performance of the web service '*insert component*' again depends upon the size of the component, but additional processing and data transfer is needed to gain an external representation of the component and for it to be inserted into the database along with the component itself. It is this additional processing and data transfer that produces the average increases observed in the performance between the web services. Gathering external representations is vital to the ReSULT system. They aid the searching mechanism by providing different façades that can be searched by the web service 'identify reusable components'.

Interpreting Figure 6.3-3, the differences between the response time for the web services that are involved with searching for and inserting components is approximately by a multiple of ten. This outcome occurs from the design architecture for the system that is detailed in Section 4.3.1. The design of this web service took advantage of a number of features within .NET that are designed to reduce the number of connections made to data sources. .NET provides a caching mechanism using 'datasets' that stores data tables from a database locally at the web service, where they are queried, and updated. At the end of the transaction any updates made to the dataset are updated made to the dataset are replicated to the data source.

The use of datasets within this project is not justified because of the small number of connections made to the service. In the current testing environment, the cost taken to deliver the functionality of datasets is greater than the cost for the number of connections. The implementation of datasets can only be justified when the cost for the number of connections exceeds this. Even then, web services could be held on different machines to improve performance (even with network latency), to avoid implementing datasets. It also has to be considered whether it is worthwhile implementing this feature if large data tables are involved, this may mean downloading a large portion of data that may never be used, or produce lapses in the security of the system.

6.3.2.3 Database Efficiency

Database efficiency is not just based upon the software used, but upon how the data is organised within a database. The design of the database is discussed within Section 4.4. The concept of this design was to produce two representations of a component, the actual component (stored inside the Code Repository table), and an external representation. This external representation consisted of five different façades; these are Classes, Fields, Methods, Context Relation, Class Type, and Traceability. Grouping these properties into individual tables removed the need for individual components to be analysed every time a search is performed.

Number of Files	**0**	**1**	**2**	**4**	**8**	**16**	**32**	**64**
Cumulative File Size	0	0.458	1.458	3.458	10.251	25.159	62.245	125.214
Class	1	2.1	2.3	2.4	2.8	3.2	3.4	4.8
Methods	1	2.1	2.2	2.4	2.9	3.6	4.8	5.9
Fields	1	2.2	2.5	2.7	3.3	5.2	9.6	10.5
Code Repository	1	2.5	3	5.4	18.5	45.3	112.2	225
Context Relation	1	2.1	2.1	2.2	2.4	2.8	3.6	5.3
Keyword Relation	1	4	8.1	8.1	10.2	16.4	23.1	35.1
Class Type	1	2.1	2.2	2.2	2.4	2.9	2.8	4.4
Traceability	1	2.1	2.1	2.1	2.3	2.5	2.8	3.5
Total Size	**8**	**19.2**	**24.5**	**27.5**	**44.8**	**81.9**	**162.3**	**294.5**

Table 6.3-2: The size of each database table in kilobytes at binary intervals.

Table 6.3-2 displays the size of the database for each component entered into it. A notable result from this table is when the number of files entered into the system is zero. This is because the database management system allocates a minimum space allocation for each table to hold its structure within it.

Storing two different representations of the same component will of course have an additional cost towards database size. Table 6.3-2 shows that when a small number of components are held in the database the ratio between the total size of the database and the actual file size is roughly forty times larger.

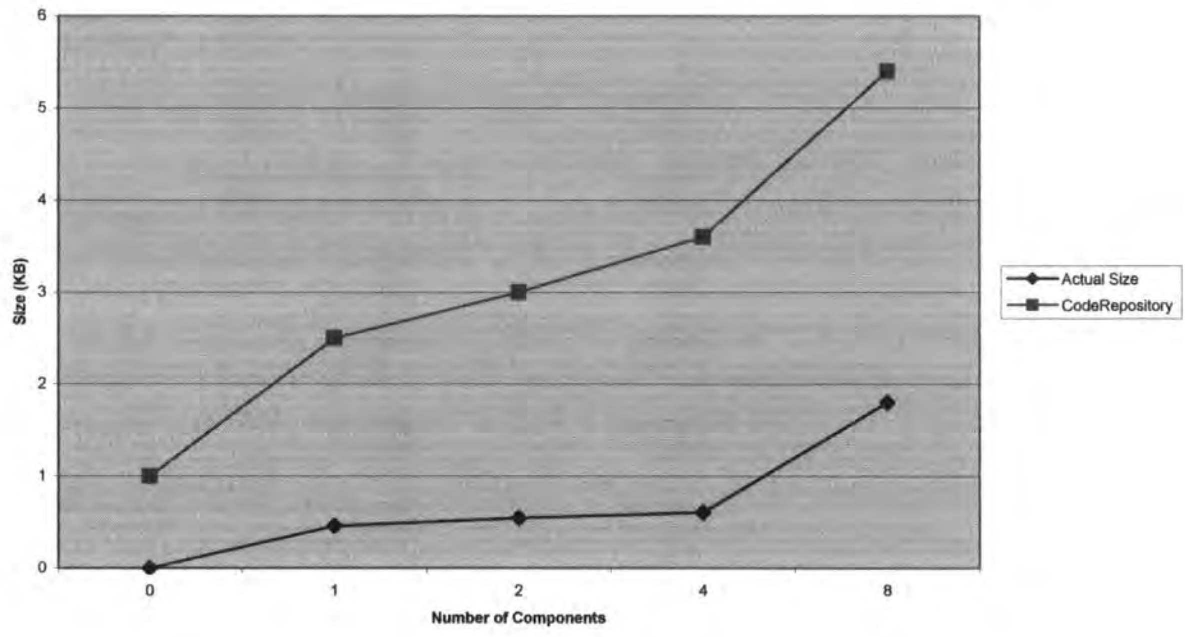


Figure 6.3-4: The relationship between the size of the component and the amount of data stored within the database.

As discussed in the previous paragraph, the overhead for storing a small number of components within the MySQL database table is significantly larger than the actual size. The largest increase is inserting the database with the data for the first component. From an empty state of 8KB, the table increases to 19.25KB by the addition of just a 0.458KB file.

Overheads are also seen when comparing the actual size of a component and the size of it within MySQL. Files are stored within MySQL in a BLOB format (described in Section 4.5.3). Examining the Figure 6.3-4, the BLOB format at which MySQL stores the component files is not the most efficient. The relationship between these factors is approximately that the database representation is approximately three times as big as its actual size.

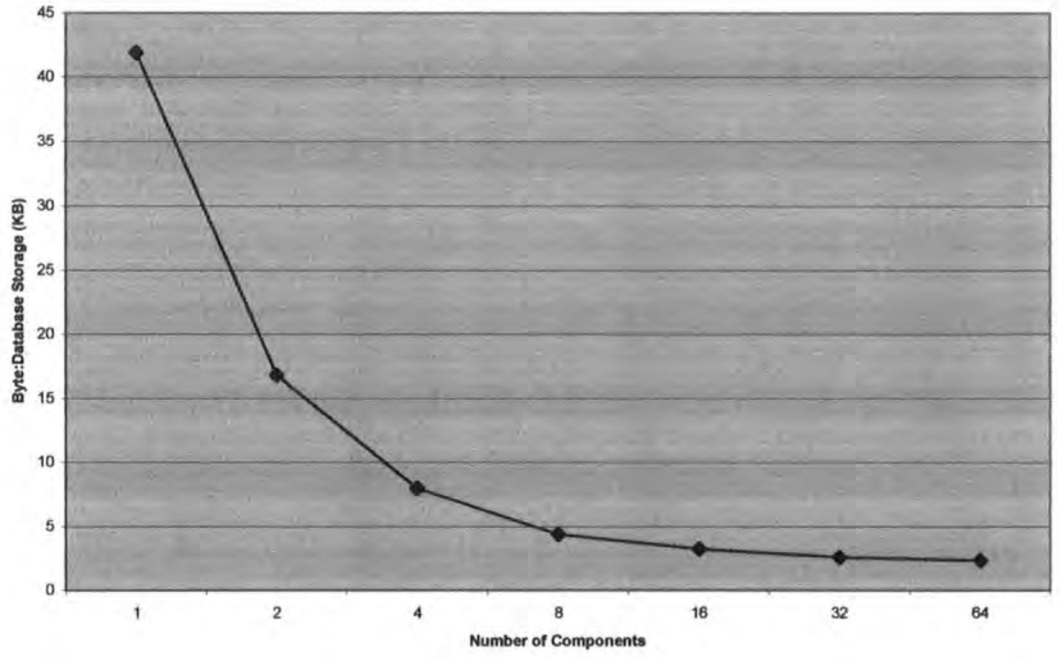


Figure 6.3-5: The relationship between 'Total Size' and 'Actual Size' from the Table 6.2

Figure 6.3-5 displays that the large ratio is just an initial overhead placed upon the data by MySQL, and that the relationship forms a logarithmic curve. The increase storage capacity needed to store BLOB objects does not affect this curve in any significant manner. This curve displays that this approach of storing two representations is ideal when considering large quantities of components that are needed within a successful reuse programme.

6.3.3 Scalability

When considering the scalability of a system, the analysis of why a system must change is undertaken. Firstly, a system could become a victim of its own success. As software engineers realise the benefits of using the ReSULT system, the demands placed upon the system may lead to a downgrade in the quality of service or a complete failure unless expanded. Secondly, systems must adapt with changing business conditions to maintain a cutting edge. This involves adding additional functionality to the system. Within this section, an evaluation of how these factors are involved within the system's architecture and search algorithm is performed.

6.3.3.1 System Architecture

The current hardware architecture is limited, and is developed purely for testing. The architecture of this system is shown in Figure 4.3-1. The version of IIS used in this system is hard coded at only accepting ten concurrent connections when installed on the Windows 2000 service pack 4 test machine. This limits the number of users that can interact with the system at any one time, and limits possible growth in usage of the system.

During Section 4.3.1, it was discussed how there were different web servers available on the market. The performance of Microsoft's IIS compared to other web servers, such as Zeus, iPlanet, and Apache, IIS is far superior when delivering static pages and performs very well when dealing with dynamic page requests [Tian02]. This is apparent when taking into consideration the results from Figure 6.3-2 because IIS delivers a linear relationship when measuring the response time and the number of requesting connections; however, increased testing is needed to identify whether this is true when larger numbers of requests are received; a change of operating system is therefore needed.

Consideration of the movement away from the Windows 2000 architecture is of paramount importance when planning for the future development of the ReSULT system. When choosing an operation system that is suitable for the task of hosting the ReSULT system, it must have the ability to host .NET services, IIS, and has the added feature of being secure. Windows based operating systems, such as Server 2003 or Windows XP Professional, are examples of operating systems that can host .NET services and IIS, but the level of security within these systems is questionable, with many security breaches being identified since their releases. In the future, it is worthwhile for an investigation towards the compatibility of Linux based servers within the ReSULT architecture to improve security of the system.

The adaptation of the ReSULT system from a test system to a live system, involves the separation of the web and database services on to individual servers. The distribution of these services on to individual machines will increase the capacity of the system, but will also increase network latency, and bring in concurrency problems with database locking during transactions.

As users identify the benefits of reuse, new applications for reuse are identified within the organisation. To allow this growth, the software architecture of the ReSULT system is designed to a framework that allows developers to expand the system. Full details of this framework are found in Section 4.5.1. This framework contains interface definitions that must be implemented when developers are expanding this system. To allow for the structural differences between design components and object orientated source code, an additional interface is implemented for the development of additional source code languages.

6.3.3.2 Algorithms

The efficiency of the algorithms involved with the ReSULT is an important factor when considering the scalability of the system. Within the ReSULT system, two algorithms are present that involve the insertion of data into the system, and the searching of information.

Search Algorithm

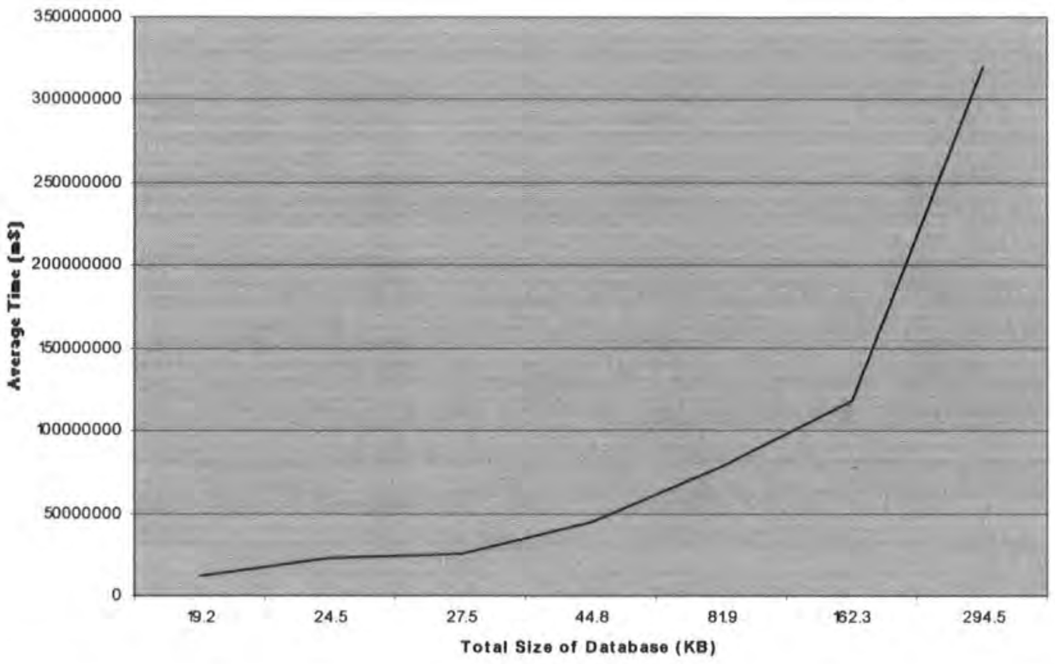


Figure 6.3-6: The relationship between the database size and the average time for the ReSULT to produce search results when using one keyword as search criteria keyword.

The details of this algorithm are detailed in Section 4.5.2.2. A major factor of this algorithm is the amount of database caching involved. Described earlier in Section 6.3.3.2, it was concluded that the use of datasets was the key reason for the efficiency differences between the three web services within the ReSULT system. When considering the scaling up of this algorithm, Figure 6.3-6 displays the results when different amounts of data were in the ReSULT system. This graph displays that the algorithm operates at approximately to O log (n^2). This is not a desirable result for the system, and will cause the performance downgrade during operation.

To improve the performance of this algorithm, a solution that involves a reduction of data passing from the database server to the web service is needed. By defining a subset within the original data, the amount of data can be reduced, but this gives the opportunity for the web service of needing to reopen connections with the database to obtain data. The use of datasets was designed to reduce the need for this, and to improve overall efficiency of a database server by lowering the number of connection requests. For this work, the use of datasets was not seen as being advantageous. This is because the time cost gained from the number of concurrent connections to the web service does not produce sufficient service downgrading for the use of caching of data within datasets to be profitable. From Figure 6.3-6 and Figure 6.3-3, when taking into consideration that ten connections to a service provides a response of 57142 ms, and that for 162.3KB of data held in the MySQL database, a time cost of 118014223 ms is achieved. Using these values, the conclusion is made that approximately 2000 connections are needed to provide a sufficient time cost for datasets to be implemented within the ReSULT system. This translates into a ratio of twelve concurrent connections for every kilobyte of data stored in the database. This figure reflects that implementing the current format of ReSULT is only beneficial to large organisations where a reuse mentality is present throughout it, and distributed systems are essential.

Insert Algorithm

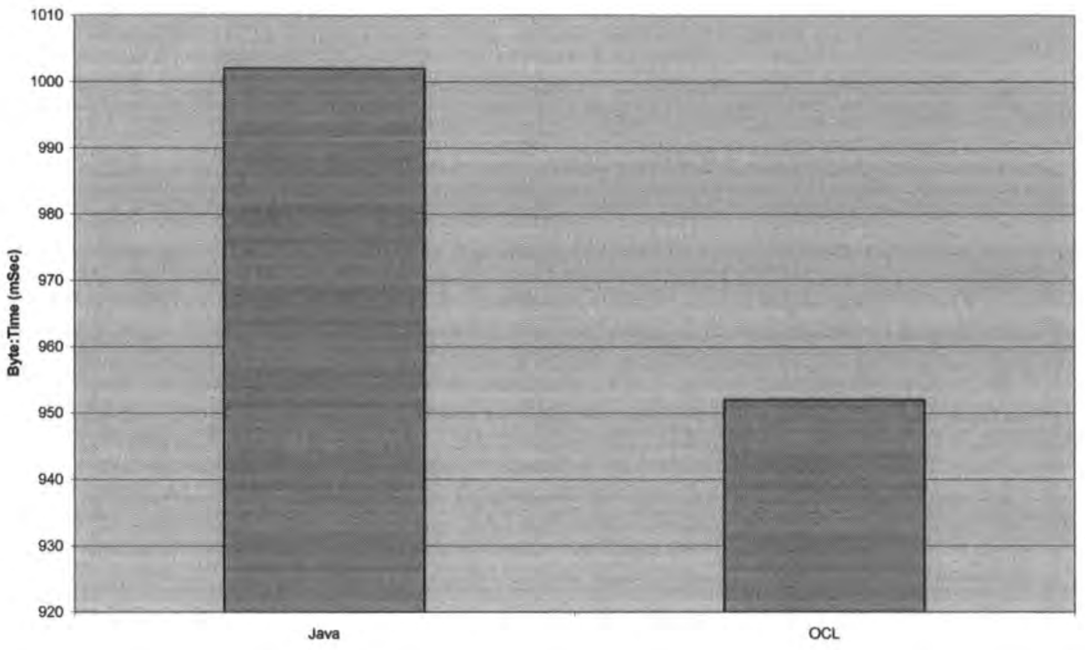


Figure 6.3-7: The efficiency between the algorithms used to analyse code within the ReSULT system.

From Figure 6.3-7, the time taken for the insert algorithm to process a byte of Java code on average takes fifty milliseconds more than OCL script. To identify where this difference in time is achieved, the analysis of the different approaches were taken towards parsing each format of component. For each format, there are two stages of parsing; structure analysis (Section 4.5.1) and keyword analysis (Section 4.5.1.2).

For the processing of Java scripts during structure analysis, an approach is taken where blocks of code are identified and then the lines inside them are analysed. This ensures that any features within the code are associated with the correct structure. OCL does not take this approach; it processes each line within a script independently because these scripts do not contain any code blocks. The benefit of which is that these scripts are processed quicker by ReSULT, but a reduction in the readability of a script is seen.

Keyword analysis is the function where comments are identified within components. Each language is processed differently within the ReSULT system. Figure 4.5-7 displays the approaches taken. The actions taken by the both comment parsers can be categorised into two groups; comment identification and comment processing. While actions involved within these groups are different, simple procedures such as identifying and removing characters from a component are only being performed.

6.3.4 Error Tolerance

There are two factors to investigate when considering error tolerance within the evaluation of the ReSULT system. Firstly, how often will functional errors appear to the user? Secondly, what is the accuracy of the components being displayed as search results?

6.3.4.1 Functional Errors

When discussing functional errors, the consideration of screens that are either displayed incorrectly or are reported with server errors is undertaken. With the current output given by the WAS tool and the limitations imposed by Windows 2000 operating system, no server failures were obtained. This was mainly due to the limited pressure placed on IIS; therefore, the request rate did not exceed the service rate. A small number of exceptions were thrown and caught by the application when interacting with the '*insertcode*' web service. When these occurred, the ASP .NET driver process failed, and the system needed to be rebooted.

6.3.4.2 Inserting Code Errors

An error within the web service of '*Insertingcode*' can lead to incorrect representation of a component. An error is classified as a misrepresentation of a field, method, or class. For example, "*classnam*'" is missing the last character and is interpreted as an error.

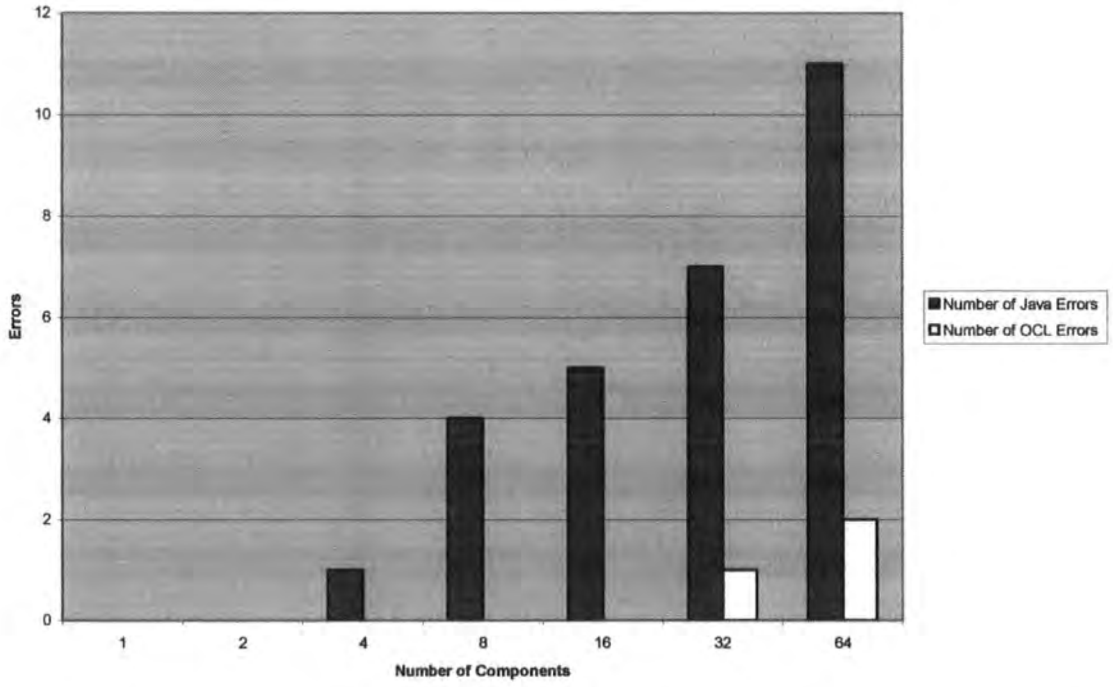


Figure 6.3-8: The number of errors produced during the data capture mechanism with the 'Insert code' web service

Figure 6.3-8 identifies that the current version of '*InsertSourceCode*' contains more errors than '*InsertOCL*'. This has a direct result on the search results produced by

ReSULT, and reduces the accuracy of searches. For future versions, errors produced by both classes need to be removed.

6.3.4.3 Search Errors

There is no one description of what classifies as a search error. The only characteristic displayed by search errors is that they are undesired by the reuser. To examine why components are undesirable, consideration of the processes within the ReSULT system is performed. These processes are:

- *Entry of search criteria by the reuser*
- *Development of ranking scores*
- *Identify components' ratings*

In Section 5.4.3, a description of how the search criteria were identified for inserting into the ReSULT system was given. This took into consideration the outputs from earlier stages of the software lifecycle such as requirement engineering. The output produced from this stage is a problem statement; from this the identification of nouns and verbs can proceed. Using the CRC approach [Beck89], the identification of these nouns and verbs indicate the desired objects wanted by the reuser and the actions that are wanted performed by these objects. However, the likelihood of a component that identically matches the search criteria entered into ReSULT is highly unlikely. To give the reuser indication of how closely components relate to their search criteria, a rating score of between 0-100 was assigned to identify components. The algorithm used to produce these ratings is described in Section 4.5.2.3. What this algorithm achieved is to identify components that match the search criteria entered in more than one façade of the ReSULT. These façades are listed below.

- *Class name*
- *Method name*
- *Field name*
- *Keyword*
- *Traceability*
- *Code Hits*
- *Code Ratings*

The searching mechanism queried the tables listed above (these represented façades of a component) and obtained a rank from each of them. The ranks obtained from 'Code Hits', 'Code Ratings', and 'Keyword' all had weights placed on the rank. These ranks were then processed to obtain a percentage score; with 100% identifying a component as perfect, but as explained in Section 4.5.2.3 the likelihood of this occurring is low.

The approach defined above has a number of possible flaws. Firstly, there was no preference towards matching a class over a field. It may be of more relevance for a reuser to have a preference towards a keyword matching a class than towards a field. One approach to solve this is by placing weights upon the structured searches, with higher values used on class searches than on fields. This approach places greater focus towards users to search for objects rather than operations within their choice of keywords. This is helpful because different objects may contain the same titled operation and may produce inaccurate results, where as when objects of the same name are identified; it is highly likely that it is a different version of the same component. With either approach, a reuser must take their own understanding and initiative to identify exactly what they want from the search results.

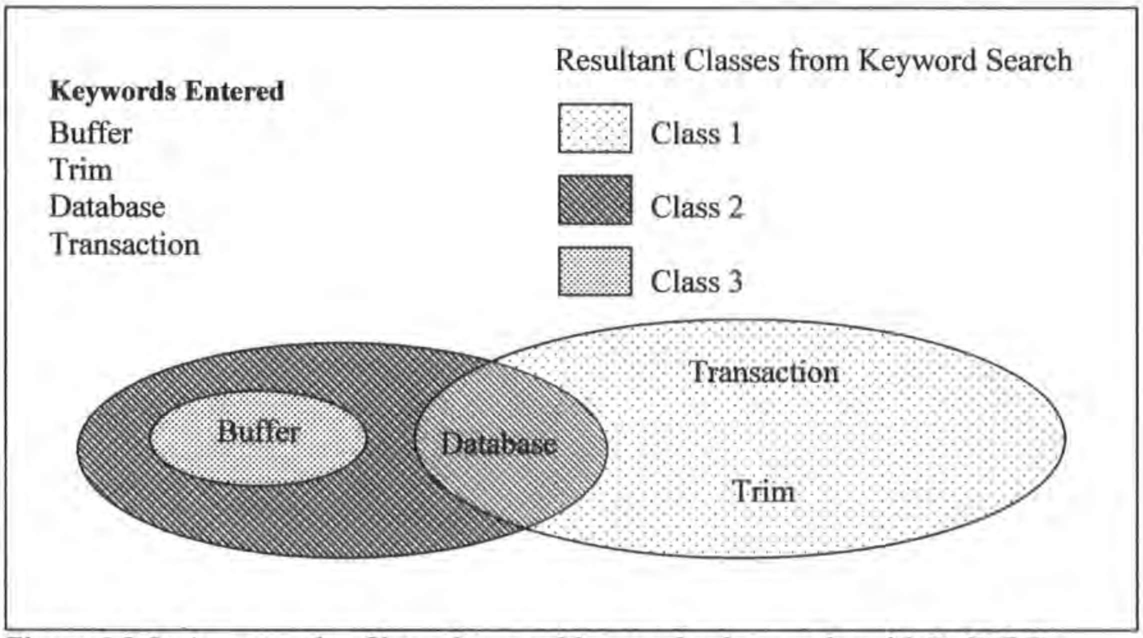


Figure 6.3-9: An example of how the searching mechanism works with ReSULT.

Secondly, the algorithm used does not rely upon the order of keywords entered. When a reuser enters keywords into the ReSULT system using the selection process described in Section 4.5.2.3, the system purely relies upon the presence of the keywords words being present within the different database tables. From this, it is increasingly likely that when using the example displayed in Figure 6.3-9, 'Class 1' would be ranked higher than 'Class 2'. However, if the searching algorithm used an ordering approach, 'Class 2' would rate higher than 'Class 1'. To identify the correct approach, consideration must be made into what form of reuse this system is being aimed towards.

Entering more search criteria into ReSULT increases the likelihood that the component achieving the highest rating will be the ideal component for reuse. This is however not a goal for the ReSULT system. The concept for this system within the process of reuse is to act as a first level filter; therefore, components of a lower rating must be distinguished clearly from each other. To achieve this clarity, a limit to the number of words a reuser can enter into a system was placed within ReSULT. Placing this limit reduces the number of components identified by the system, and improved ReSULT's performance, but this does not increase the mean value for the ratings produced. To increase the performance of ReSULT even further and to achieve a larger mean value, a limit to the number of results provided by the system should be implemented into the system in future versions.

By enforcing this restriction upon reusers, they are forced to prioritise features that they desire. The current design of ReSULT applies weights to ranks achieved from querying the façades of 'Code Hits', 'Code Ratings', and 'Keyword'. Weighting of ranks are not applied for structural features such as class hierarchies, interfaces implemented, methods and field names. The decision to not apply this weighting was based on the limit of search criteria, and for reuser's feedback to distinguish component quality. One attribute of component quality is documentation. If a component contains many comments that detail the processes within the component, it will firstly give a reuser an advantage in gaining an in-depth understanding of a component; but secondly, a larger number of comments will be identified by the ReSULT system. This increases the likelihood that this collection will contain replications of important terms that a reuser has entered as search criteria. Using the formula designed in Section 4.5.2.3, the ratings of a component that contain replicated comments and are equivalent to the reuser's search criteria are increased, and are displayed higher up in the list of search results. The ability of placing weights on 'Code Hits' and 'Code Ratings' to indicate quality is a problem during the early stages of deployment because there will be no data to make these distinctions. This will increase the amount of work (in the form of examining components) that a reuser will perform, and may damage their confidence of using this system.

6.3.4.4 Integrating Components into a Project

In Section 2.2.2, discussion of the differences between white box and black box reuse [Kwon98] was performed. From this, it was identified that it is more effective for the reuser to identify a component that they do not have to build upon, and can simply insert into their project. If this component is identified during the design phase, the reuser can then take advantage of the traceability function with the system that can select the source code script for the OCL component, or identify design patterns that classes may be members of. It is however; an unlikely circumstance that all the keywords entered will be associated with one component. The result of which will lead to the reuser either altering the desired component, or producing new objects that will interact with the selected class to provide the ideal functionality. By using the approach 'close but not perfect' when considering the selection algorithm, less effort is needed into performing adaptive maintenance to the component. It is therefore, more appropriate to consider the number of terms associated with a component than to consider the ordering of these terms.

6.4 Overview of Work

Product Transition	78%
Product Operation	86%
Product Revision	52%

Table 6.4-1: Summary of McCall's Software Evaluation Criteria.

Table 6.4-1 displays the percentage scoring for McCall's software quality factors [McCal77]. These values have been gathered by grouping together the measurable properties defined in Section 5.2. These properties have been calculated using the equations for McCall's software quality model in Table 7.4-4, and using the metrics scores in Table 7.4-2 for values for these equations.

From the scores achieved in Table 6.4-1, the ReSULT system displays weakness when considering the areas of maintenance, flexibility and testability, but shows significant quality when considering the portability, reusability, interoperability, correctness, efficiency, integrity and reliability.

6.5 Summary

In this chapter, the ReSULT reuse system was evaluated. The criteria for the success of this research appeared in Chapter 1 were discussed.

In the next section, its strong points and weak points were identified.

Through the above evaluation, it can be said that the soundness and usefulness of ReSULT towards reuse within an organisation has been demonstrated.

Chapter 7 Conclusions

7.1 The Main Achievements of the Research

The achievements and results of this research are as follows:

- The development of a distributed reuse system (ReSULT).
- ReSULT allowed two different types of code to be inserted into a reuse repository.
- The reuse repository was searchable using keywords. The results of these searches adjusted due to the popularity of a component's extraction and user ratings of it; therefore increasing the accuracy of the search.
- Components could be extracted from the repository.
- Components were related by design pattern and package; therefore, identifying components that were related to the component from the search results was quick and simply.
- An approach was also defined that identified the how the ReSULT system was to be introduced into the software lifecycle of an organisation.

7.2 General Conclusions of the Research

The outcome of this thesis was the production of a distributed system called Reuse Sourcecode Units Library Tool (ReSULT). The major results of this research as described in the criteria for success in Chapter 1 are as follows.

Criteria 1: Suggest guidelines for an approach to code reuse.

There are four key processes within ReSULT, these correspond to key activities within reuse, and are focused upon when considering an approach to code reuse.

Process 1: Produce reusable code

For the ReSULT system's search mechanism to work efficiently, inserted components must conform to a set of standards for components (Section 2.3.2.1) such as prologue commentary at the beginning of a component, traceability to a component's design document, in-line comments, and correct indentation. These standards ensure that components' external representations are identified precisely before being inserted into the system. Applying these standards, also encourages the diminishing of the 'not invented here' syndrome within an organisation that users must adhere to. The guidelines for producing reusable code within the ReSULT system are:

- *Uniform approach to converting existing components*
- *A significant number of comments present within code*

- *The inclusion of both inline and prologue comments*
- *High cohesion*
- *Low coupling with other components*
- *A definitive naming scheme for fields, and methods*

Process 2: Insert code into system

Insert code into the ReSULT system relied upon the presence of a reusable asset. This asset may be part of a design pattern, or has been produced by design documents that can be referenced. It is at this stage where the entering of this information into the system is performed.

Process 3: Search repository

ReSULT introduced a three-part approach to searching for reusable assets. The first defines the problem within a statement. The second uses this problem statement to identify attributes within it and organise them onto Class Responsibility and Collaboration (CRC) cards. From CRC cards, OCL transcriptions are produced or identified using the ReSULT search mechanism. Search criteria were selected using the values from CRC. Source code can then be identified using traces from OCL components, or if no reuse options are available, code is written. This approach increases the time spent on design, but decreases implementation costs by increasing software quality, and decreasing time costs with integrating reusable components into projects.

Process 4: Extract component

The extraction of a component from the ReSULT system relied upon the user copying the component from the web browser and pasting it into their projects. The centralised storage architecture for reusable components within the ReSULT system, and the accessibility to the system via HTTP enables reusers to access many peoples work across physical boundaries.

Criteria 2: Identifying criteria that are used to select a component for reuse within a repository.

The properties defined for selection criteria can be identified as either structural elements of components or user feedback. When considering structural elements, consideration is made for properties of components such as class name, method name, field names, comments, design pattern information, and traceability links that have been declared by the author. These properties are declared as different facets a component is viewed upon. To include feedback from previous reusers experience with selected components feedback data such as how many times a component has been extracted from the system, and feedback scores provided from the user is also included as search criteria.

Criteria 3: Provide a distributed tool that enables many employees within an organisation to insert and search for reusable components.

The ReSULT is a means for employees to effectively reuse code within an organisation. The system used web service technologies to provide communication database servers, web application servers, and the reuser.

Within this system, there are four processes (Criteria 1). These provide functionality for reusers to insert, search and extract components from a reuse repository. To define an accurate search algorithm, research was performed into how components could be represented within the system, and how this representation could be identified and placed into the system during the inserting process. The criteria chosen for this is defined in Criteria 2.

The performance of the searching was not desirable. The operating time of O log (n^2) was identified as being caused by the use of datasets when transporting data between web services, and the need for additional processing to identify the external representation of the component. Analysis into this identified that roughly two thousand concurrent connections are needed to provide a sufficient time cost for datasets to be implemented within ReSULT; therefore, the conclusion was made that ReSULT is beneficial to large organisations where a reuse mentality is present throughout.

The process of parsing components and inserting both the internal and external representations of a component produced a time cost that was quicker than the process of searching. There was a significant difference in the time taken to parse the different languages. Java code on average took fifty milliseconds to insert per byte compared to OCL, but a reduction in the detail of the component's external was seen.

To provide the ability for the system to grow and adjust to changes in working practices, a framework was developed that eased the integration of new languages. ReSULT provided interfaces for new implementation modules to aid this maintenance.

The system architecture provided a method of delivering the system to many users. The findings found that when analysing the performance of the web application, the limit placed upon the number of connections to the web service did not justify the implementation design. The design of the system was for a large number of connections and to improve efficiency for connections from the many users by applying datasets to the system design; this was not justified. The test system only allowed for ten concurrent connections, and with the large overhead the datasets had on the system. The advantages of using datasets were never seen, and at the low level of connections impaired the systems efficiency.

MySQL database server is an effective means of storing data. The design of ReSULT ensured that both the internal representation and external representations of a component were stored in this database server. The internal representation i.e. the component itself was stored in a Binary Large Object (BLOB) format. BLOB format is not the most efficient for storing small component because the database representation is approximately three times larger than its actual size. This is just an initial overhead placed upon the data by MySQL, and as component sizes increases the ratio decreases. This forms a relationship that corresponded to a logarithmic curve.

Criteria 4: Within the distributed reuse system; design a search mechanism that will provide accurate search results that reflect upon the many facets a component can be viewed from.

As concluded earlier in Criteria 2, the ReSULT system identifies a number of different structural elements and user feedback that are interpreted as individual façades of a component. Weights were applied to the rankings of these façades, so that the final ratings placed upon components increased if objects and operations were identified with the component. Smaller weights were applied to user feedback scores. The concept for this was to acknowledge that for a component to be identified as being good, it must be deemed this by many reusers and not by a select few. As time goes by, the number of components extracted and the number of feedback scores increase; therefore, improving the accuracy and reliability of the search results. Accuracy also increased if there was a significant number of comments were present within components. It has to be remembered that the ReSULT system was designed to act as a first level filter because the system uses the 'close but not perfect' approach was considering the production of search results. A reuser therefore must take their own understandings and initiative to identify exactly what they want from the search results, and possibly expand upon it.

During this work an investigation was performed that considered what search criteria was entered into the system by a reuser. It was identified that it was more appropriate to consider the number of terms associated with a component then to consider the ordering of search criteria terms, and that by entering more increases the likelihood that the component achieving the highest rating will be ideal for reuse within that scenario. To give clarity to the outputted results and increase performance, a limit was placed on the possible number of words entered as search criteria. This ensured that a reuser identifies strong terms such as objects and operations that they are considering to be reused.

Criteria 5: Validating the usefulness and usability of the distributed reuse system.

The forecasted benefits of introducing the ReSULT system into an organisations fall into three different categories, increased speed of production, financial benefits, and improve the quality of the software. The financial benefits brought by introducing the ReSULT all concern the reduction in time spent in producing code. An increased speed is forecasted to be observed by the reduction in time spent during the implementation phase of projects. This is aided through providing an efficient accurate search mechanism that searches a reuse repository for keywords and identifying features that have been defined at the design stage.

A large investment is initially needed at the introduction of a reuse system, the main reason for this being the production of reusable assets to populate a repository. To counter these costs, the use of web service architecture within the ReSULT lowers the time costs of the initial implementation, and enables the possibility of the selling of reusable assets to third parties using EDI.

An increase in software quality is not directly brought to an organisation by introducing a reuse tool; it is brought by the application of two concepts, introducing a disciplined approach to reuse, and continual code reviewing. Applying these concepts increases the percentage of time spent on planning and reduces time within implementation. If errors are identified later on in a components lifecycle, perfective maintenance is performed upon the component, therefore giving an improvement in the component's quality. Providing additional design documentation eases maintenance duties and gave the ReSULT system more opportunities to gain a greater detailed external representation of a component, therefore improving the accuracy of component searching.

ReSULT provided a web-based application that is accessible over a heterogeneous network. This enables the distribution of knowledge through an organisation of how reusable assets are designed. To gain the most out of the ReSULT, it is predicted that a stringent training is put into place that not only teaches the processes of ReSULT, but of how to produce reusable components.

7.3 The Limitations of the Approach

Firstly, in this research, the ReSULT system was not applied and tested within a real company; therefore, the true benefits or problems caused by implementing the ReSULT system into an existing reuse process, or initiate a reuse approach within a company could not be identified.

Secondly, all application servers were located on one machine. This affected the performances of the services they provided, and an accurate measurement of the systems performance could not be obtained.

Finally, the implemented system did not have a suitable architecture that supported the desired application of the system. The web server that supported the system only allowed for a maximum of ten connections at any one time, and therefore could not be tested to the appropriate levels.

7.4 Suggestions for Future Research

From the research performed within this work, a foundation has been built that can extended in a number of areas. These areas are distributed systems, and software engineering.

Within the field of distributed systems, further research into the areas of how to reduce the time costs incurred by the using the ReSULT. A commercial area of research within the distributed arena is the application of e-commerce within the ReSULT. The area of e-commerce in question concerns Business-to-Business (B2B) commerce. This research would examine the interactions between web services from different organisations and the possible security issues that may arise.

When considering possible extensions of research within Software Engineering, the main areas of research involve change configuration management, and reverse engineering. The advantages of applying change configuration management to the ReSULT system are the application of versioning of components, and providing the ability of locking components to one editing sessions. These are highly advantageous features when considering a large distributed system with many users accessing and editing components on it. Within the area of reverse engineering, research into the identifying of external representations for other languages is also a possibility. This would allow the possibility for the system to inserting other languages and expanding the scope of the system.

References

[Aoyam95]Aoyama, M.; "New age of software development: How component based software engineering changes the way of software development"; International Conference on Software Engineering; 1998; pp179-185

[Agres99] Agresti, W. W., McGarry, F.; "The Minnowbrook Workshop on Software Reuse: a summary report". In Tracz. W.; "Software Reuse: Emerging Technology", Ed., IEEE; 1987

[Bator92] Batory, D., O'Malley., S.; "The Design and Implementation of Hierarchical Software Systems with Reusable Components"; ACM Transactions on Software Engineering and Methodology Vol. 1(4); October 1992; : pp355-398

[Beck89] Beck, K.; "A Laboratory for Teaching Object-Orientated Thinking"; From the OOPSLA'89 Conference Proceedings October 1-6, 1989, New Orleans, Louisiana; pp1-6

[Benne00] Bennett, K. H.; Rajlich, V. T; "Software Maintenance and Evolution: a Roadmap". In Finkelstein, A. (ed.); "The Future of Software Engineering"; ACM Press; 2000; pp75-87

[Berto04] Bertoa M.F., Vallecillo A.; "Usability metrics for software components"; Dpto. Lenguajes y Ciencias de la Computacion, Universidad de Malaga.

[Boehm99]Boehm, B.; "Managing Software Productivity and Reuse"; IEEE Computer Vol. 32; Sept. 1999; pp111-113

[Brook95]Brooks, J.R.; "No Silver Bullet Refined". In The Mythical Man-Month, Anniversary ed. Reading, MA.; Addison-Wesley Pub. Company, Inc.; pp205-226

[Brook83]Brooks, R.; "Towards a Theory of the Comprehension of Computer Programs"; International Journal of Man-Machine Studies; Vol. 18 (6); 1983; pp543–554

[Brown98] Brown, A.W., Wallnau, K.C.; "The Current State of CBSE"; IEEE Software Vol. 15 (5); Sept. 1998; pp37-46

[Caput98] Caputo, K.; "CMM Implementation Guide: Choreographing Software Process Improvement"; Book; 1st Edition; Addison-Wesley Pub. Company, Inc.; 1998

[Cauld01] Cauldwell, P., Chopra, V., Zoran, Z., Damschen, G., Dix, C., Chawla, R., Saunders, K., Olander, G., Norton, F., Hong, T., Ogbuji, U., Richman, M.A.; "Professional XML Web Services"; Book; Wrox Press Ltd 2001

[Cerem02] Ceremi, E.; "Web Services Essentials"; Book; O'Reilly & Assoc; Feb 2002

[Chaud02] Chaudhary, A.S., Saleen, M.A., Bukhari, H.Z.; "Web Services in Distributed Applications: Advantages and Problems"; Ghulam Ishaq Khan Institute of Engineering Science and Technology; 2002

[Czarn00] Czarnecki, K., Eisenecker, V.W.; "Generative Programming methods, Tools and Application"; Book; Addison-Wesley Pub. Company, Inc. 2000

[Dogac02] Dogac, A., Tambag, Y., Pembecioglu P., Pektas S., Laleci G., Kurt Gokhan., Toprak S., Kabak Y.; "An ebXML infrastructure Implementation through UDDI Registries and RosettaNet PIP's"; ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, USA; June 2002; pp512-523

[Dykma99]Dykman, N.; "Sage: Generating applications with UML and Components"; MSc Thesis; University of Utah; 1999.

[Fishe87] Fischer, G.; "Cognitive view of reuse and redesign"; IEEE Software, 4 (4); 1987; pp60-72

[Flurr01] Flurry, G.; "Applying web services to the Application service provider environment: An example of web services applied to e-business"; IBM DeveloperWorks; January 2001

[Fraze92] Frazer, A.; "Reverse Engineering – hype, hope or here?" In Software Reuse and Reverse Engineering in Practice; P. A. V Hall (ed.) pp209-243

[Gamma95]Gamma, E., Helm, R., Johnson, R., Vlissides, J.; "Design Patterns: Elements of Reusable Software"; Book; Addison-Wesley; 1995

[Glass00] Glass, G., "The Web Services Revolution: Applying Web Services to Applications", IBM Developer Works, November 2000

[Gotts00] Gottschalk, K.; "Architecture Overview: The next stage of evolution for e-business"; IBM Developer Works; September 2000

[Gotts02] Gottschalk, K., Graham, S., Kreger, H., Snell, J.; "Introduction to Web Services Architecture"; IBM Systems Journal; Vol. 41 (2); 2002

[Henni97] Henninger, S.; "Case-Based Knowledge Management Tools for Software Development"; Journal of Automated Software Engineering, Vol. 4 (3); July 1997

[Hoyda91] Hoydalsvik, G. M., Karlsson, E. A., Sorumgard, S., Thunem, S., Tryggeseth, E.; "Object-Orientated Development With and For Reuse"; Division of Computer Systems and Telematics, Norwegian Institute of Technology; 1991

[Irani01] Irani, R.; "Introduction to ebXml". In Fletcher, P.; Waterhouse M.; "Web Services and Business Strategies and Architectures Book"; Book; Expert Press; 2002; pp221-236

[Jacob99] Jacobson, I., Booch, G., Rumbaugh, S.; "The Unified Software Development Process"; Book; Addison-Wesley Pub. Company, Inc. 1999

[Jepso01] Jepson, T.; "SOAP Cleans up Interoperability Problems on the Web"; IT Professional, Vol. 3 (1); January-February 2001; pp52-55

[Karls95] Karlsson, E.A., "Software Reuse: A Holistic Approach"; Book; John Wiley & Sons Ltd; 1995

[Krueg92] Krueger, C.W.; "Software Reuse"; ACM Computing Surveys; Vol.24 (2); June 1992

[Kunda00] Kunda, D., Brooks, L.; "Identifying and Classifying Processes (traditional and soft factors) that Support COTS Component Selection: A Case Study"; European Journal of Information Systems, Vol. 9(4); 2000; pp 226-234

[Kwon98] Kwon, O.C.; "A Process Model of Maintenance with Reuse: An Investigation and an Implementation"; Ph.D. Thesis, University of Durham; 1998

[Larma05] Larman, C.; "Applying UML and Patterns: An Introduction to Object-Orientated Analysis and Design and Iterative Development"; Book; Third Edition; Prentice Hall; 2005

[Lehma80] Lehman, M.; "Programs, life cycles and laws of software evolution", Proceedings of IEEE Special Issue on Software Engineering, Vol. 68 (9), Sept. 1980; pp1060-1076

[Lien80] Lientz, B. P., Swanson, E. B.; "Software Maintenance Management"; Book; Addison Wesley; 1980

[Lim98] Lim, W.C.; "Managing software reuse: a comprehensive guide to strategically reengineering the organisation for reusable components"; Book; Prentice Hall PTR; 1998

[Loren94] Lorenz, M., Kidd J.; "Object-Oriented Software Metrics: A Practical Guide"; Book; Prentice-Hall, Inc.; 1994

[McCal77] McCall, J.A, Richards, P.K., Walters, G.F.; "Factors in Software Quality, vol. 1, 2, 3"; Springfield, VA: National Technical Information Service; 1977

[McIlr68] McIlroy, M.D., "Mass-produced Software Components". In Naur, P., Randell, P(ed).; Proceedings of NATO Software Engineering Conference Vol. 1; NATO Science committee 1968; pp138-150

[McNam84]McNamara, B.; "Japanese Software Factories". In Standish T.A.; "An Essay on Software Reuse"; IEEE Transactions on Software Engineering; Vol. 10, No. 5; September 1984

[Mei02] Mei, B., Xie T., Yang F.; "A Model-Based Approach to Object-Oriented Software Metrics"; Journal of Computer Science and Technology Vol. 17 (6) Nov. 2002; pp757-769

[Mitte01] Mittermeir, R. T., Bollin, A., Pozewaunig, H., Rauner-Reithmayer, D.; "Goal-Driven Combination of Software Comprehension Approaches for Component Based Development"; Institute of Informatics-Systems, Klagenfurt University, Austria

[Meyer98] Meyer, B.; "Object Orientated Software Construction"; Book; Prentice-Hall; 1998

[Morri03] Morris, J., Peng Lam, C., Bundell, G.A., Lee, G., Parker, K.; "Setting a Framework for Trusted Component Trading". In Component-Based Software Quality: Methods and Techniques, Cechich, A., Piattini, M., Vallecillo, A.; (Eds.) Lecture Notes in Computer Science, Vol. 2693, Springer; June 2003; pp 101-131

[Norma83]Norman, D.A.; "Some observations on mental models". In Gentner, D., Stevens, A.L; Eds., Mental Models, Erlbaum, Hillsdale, NJ, 1983; pp7-14

[ORior02] O'Riordan, D.; "Business Process Standards for Web Services". In Fletcher, P.; Waterhouse M.; "Web Services and Business Strategies and Architectures Book"; Book; Expert Press; 2002; pp157-174

[Paulk95] Paulk, M.C., Weber, C.V., Curtis, B., Chriss, M.B.; "The Capability Maturity Model: Guidelines for Improving the Software Process"; Book; Addison-Wesley Pub. Company, Inc.; 1995

[Priet87] Prieto-Diaz, R., Freeman, P.; "Classifying Software for Reusability"; IEEE Software; January 1987; Vol.4 (1) pp6-16

[Rajli02] Rajlich, V., Wilde, N.; "The Role of Concepts in Program Comprehension"; Proc. Of IWPC 2002, IEEE Computer Society Press, Los Alamitos, CA; pp271-278

[Ramal04] Ramalingam, V., LaBelle, D., Widenbeck, S.; "Efficacy and Mental Models in Learning to Program", Proc. Of the9th annual SIGCSE Conference on innovations and technology in Computer Science Eduction; 2004; pp171-175

[Ranan00] Rananjan, A., Roy, J., "XML: Data's Universal Language" IT Professional Vol. 2 (3); May/June 2000; pp32-36

[Retko97] Retkowsky, F.; "Software reuse from an external memory: The cognitive issues of support tools". In PPIG Workshop '98 Proceedings (Open University, Milton Keynes, UK, Jan. 1998); Psychology of Programming Interest Group 1998

[Rosen95] Rosenbaum, S., DuCastel, B.; "Managing software reuse – an experience report". In Proceeding of 17th International Conference in Software Engineering", Seattle, WA; 1995; pp105-111

[Samta02] Samtani, G., Sadhwani, D.; "Enterprise application integration (EAI) and Web Sevices". In Fletcher, P., Waterhouse, M. (Eds),Web services Business Strategies and Architectures, Expert Press Ltd, Birmingham, AL, pp.39-54.

[Schmi99] Schmidt, D.C.; "Why Software Reuse has Failed and How to Make It Work for You", C++ Report SIGS Vol. 11; January 1999

[Selec03a] "Software Reuse: Measuring Return on Investment"; Select Business Solutions, Inc; 2003

[Selec03b] "Select on Web Services"; Select Business Solutions, Inc; 2003

[Snell02] Snell J., Tidwell, D., Kulchenko, P.; "Programming Web Services with SOAP"; Book; O'Reilly & Assoc inc.; Jan 2002

[Solow84] Soloway, E., Ehrlich, K.; "Empirical Studies of Programming Knowledge"; IEEE Transactions on Software Engineering; Vol. SE-10, No. 5, pp. 595 – 609; September 1984

[Stand84] Standish, T.A.; "An Essay on Software Reuse"; IEEE Transactions on Software Engineering; vol. 10. (5); Sept 1984. In Burd, E.L.; "Reuse with Risk Management: A Decision Support Approach"; D.Phil. Thesis, University of York; 1999

[Stand97] Standish, T.A.; "An Essay on Software Reuse: principles and practices and economic models"; Book; Addison-Wesley Pub. Company, Inc; 1997

[Sycar03] Sycara, K.P.; "From the 'eyeball' web to the Transaction web"; On the Move Federated Conferences (OTM '03): "On the Move to Meaningful Internet Systems and Ubiquitous Computing"; Catania, Sicily, Italy; 2003

[Tian02] Tian, M., Voigt, T., Naumowicz, H., Schiller, J.; "Performance Impact of Web Services on Internet Servers"; Computer Systems & Telematics; Freie Universität Berlin; 2002

[Timen89] Timens, T.; "Cognitive Models of Program Comprehension"; Software Engineering Research Center, 1989

[Tracz95] Tracz, W., "Confessions of a used Program Salesman: Institutionalizing Software Reuse"; Book; Addison-Wesley Pub Company, Inc.; 1995.

[vande03] van der Aalst, W.M.P., "Don't go with the flow: Web Services Composition Standards Exposed". In "Trends & Controversies"; IEEE Intelligent Systems; Vol 18 (1); 2003; pp72-76

[vonMa94]von Mayrhauser, A., Vans, A. M.; "Program Understanding – A Survey" Computer, Vol. 28 (5); August 1995; pp44-55

[Web01] "Namespaces in XML", WWW Consortium, January; online at URL: http://www.w3.org/TR/REC-xml-names/; Viewed on 31 November 29th November 2003

[Web02] "UDDI Technical White Paper"; Publication; Aribia Inc.; International Business Machines Corporation 5; 2000 online at http://www.uddi.com/pubs/lru_UDDI_Technical White Paper; Viewed on 24th January 2004

[Web03] Frey, J.E., Rosvall, A.; "Web Services Simplifies iAppliance Software Reuse"; online at URL: http://www.iapplianceweb.com/story/OEG20020331S0003; Viewed on 20th November 2003

[Web04] Baxter, J., Humpries, A.; ".NET ARCHITECTURE"; online at URL: http://ecommerce.ncsu.edu/csc513/student-work/DOT-NET/DOT-NET.htm; Viewed on 30th November 2004

[Web05] Wikipedia Encyclopedia..; "Reverse Engineering Definition"; online at URL: http://en.wikipedia.org/wiki/Reverse_engineering; Viewed on 12th December 2003.

[Web06] "Introduction to XML Schemas"; online at URL: msdn.microsoft.com/library/en-us/ vbcon/html/vbconIntroductionToXMLSchemas.asp; Viewed on 24th June 2004

[Web07] "Research-Based Web Guidelines: Software/Hardware"; online at URL: http://usability.gov/guidelines/softhard.html; Viewed on the 3rd March 2005

[Web08] "ebXML – Enabling A Global Electronic Market"; Viewed on the 11th May 2005

[Weyuk98]Weyuker, E.J.; "Testing Component Based Software: A Cautionary Tale"; IEEE Software 15; 1998; pp32-37

[Yunwe02]Yunwen, Y.; "An Empirical User Study of an Active Reuse Repository System." In Proceeding of 7th International Conference on Software Reuse (ICSR-7), Austin, TX, pp281-292, Apr 15-19, 2002.

Appendix 1 UML Modelling of ReSULT System

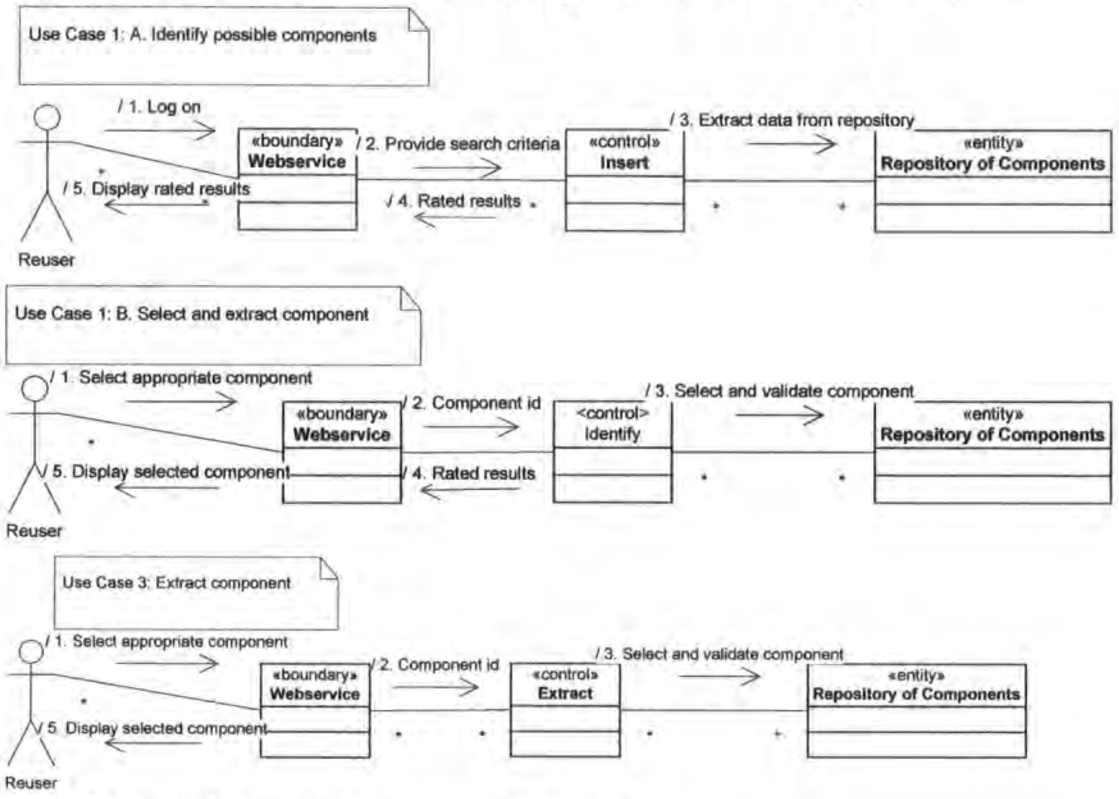

Use Case 1: A. Identify possible components
/ 1. Log on
«boundary» Webservice
/ 2. Provide search criteria
«control» Insert
/ 3. Extract data from repository
«entity» Repository of Components
/ 5. Display rated results
/ 4. Rated results
Reuser
Use Case 1: B. Select and extract component
/ 1. Select appropriate component
«boundary» Webservice
/ 2. Component id
<control> Identify
/ 3. Select and validate component
«entity» Repository of Components
/ 5. Display selected component
/ 4. Rated results
Reuser
Use Case 3: Extract component
/ 1. Select appropriate component
«boundary» Webservice
/ 2. Component id
«control» Extract
/ 3. Select and validate component
«entity» Repository of Components
/ 5. Display selected component
Reuser

Figure 7.4-1: Collaboration between analysis classes

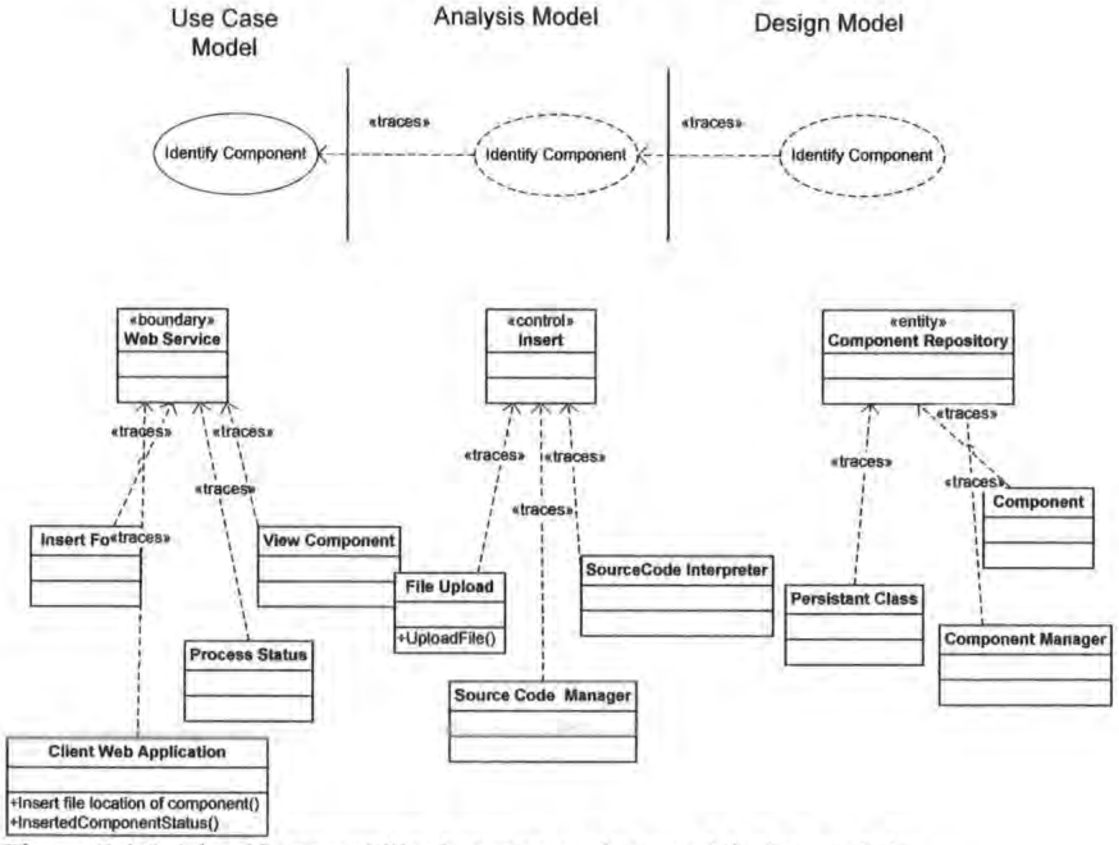


Figure 7.4-2: Identify traceability between analysis and design models

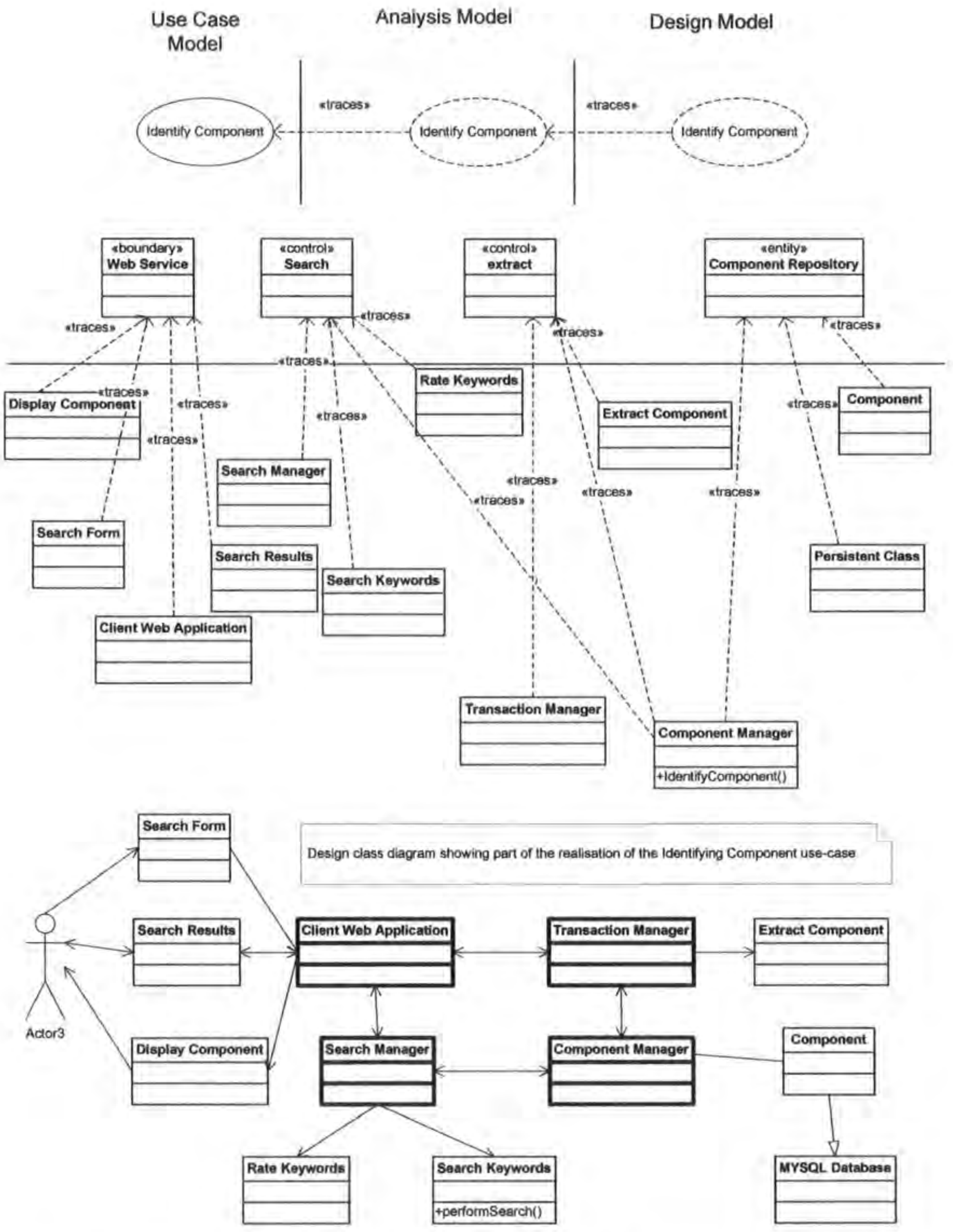


Figure 7.4-3: The realisation of the 'Identifying Component' use-case from analysis classes to design classes.

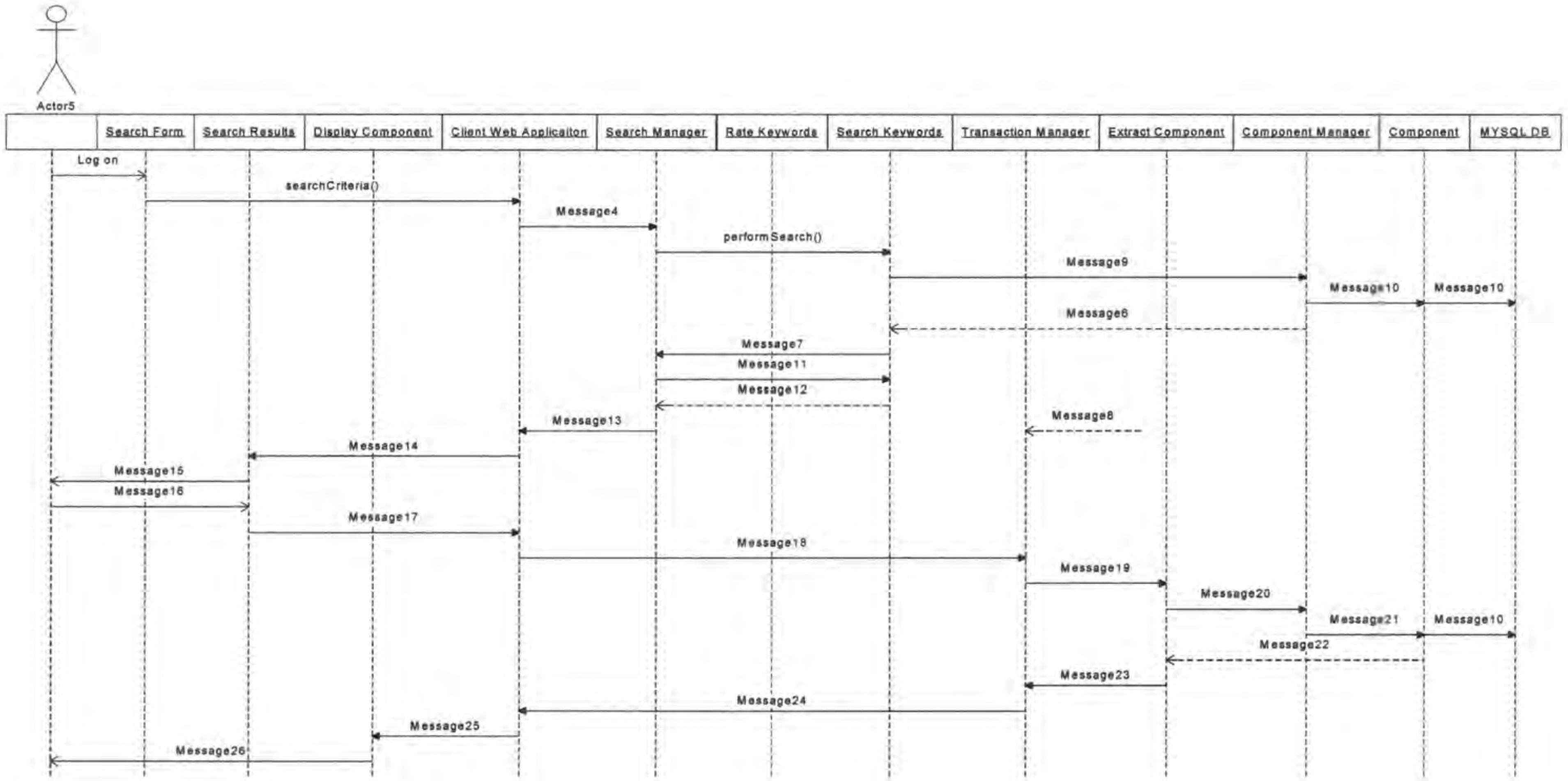


Figure 7.4-4: A sequence diagram showing the relationship between the different design classes in the 'Identifying Components' use-case.

Appendix 2 Web Service Descriptions (WSDL)

```
  <?xml version="1.0" encoding="utf-8" ?>
- <definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:s0="http://129.234.201.28/reuse/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    targetNamespace="http://129.234.201.28/reuse/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
  - <types>
    - <s:schema elementFormDefault="qualified"
        targetNamespace="http://129.234.201.28/reuse/">
      - <s:element name="getRankedSearch">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="0" maxOccurs="1"
                name="sSearchMethod" type="s:string"
                />
              <s:element minOccurs="0" maxOccurs="1"
                name="sKeyWords" type="s:string" />
            </s:sequence>
          </s:complexType>
        </s:element>
      - <s:element name="getRankedSearchResponse">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="0" maxOccurs="1"
                name="getRankedSearchResult"
                type="s:string" />
            </s:sequence>
          </s:complexType>
        </s:element>
      </s:schema>
    </types>
  - <message name="getRankedSearchSoapIn">
      <part name="parameters" element="s0:getRankedSearch"
        />
    </message>
  - <message name="getRankedSearchSoapOut">
      <part name="parameters"
        element="s0:getRankedSearchResponse" />
    </message>
  - <portType name="searchrepositorySoap">
    - <operation name="getRankedSearch">
        <input message="s0:getRankedSearchSoapIn" />
        <output message="s0:getRankedSearchSoapOut" />
      </operation>
    </portType>
  - <binding name="searchrepositorySoap"
      type="s0:searchrepositorySoap">
      <soap:binding
         transport="http://schemas.xmlsoap.org/soap/http"
         style="document" />
    - <operation name="getRankedSearch">
        <soap:operation
          soapAction="http://129.234.201.28/reuse/getRank
          edSearch" style="document" />
      - <input>
          <soap:body use="literal" />
        </input>
      - <output>
          <soap:body use="literal" />
        </output>
      </operation>
    </binding>
  - <service name="searchrepository">
    - <port name="searchrepositorySoap"
         binding="s0:searchrepositorySoap">
        <soap:address
          location="http://localhost/reuse/searchrepository.
          asmx" />
      </port>
    </service>
  </definitions>
```

Figure 7.4-5: WSDL Description for the web service '*Identifying Reusable Component*'

```
  <?xml version="1.0" encoding="utf-8" ?>
- <definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:s0="http://129.234.201.28/reuse/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    targetNamespace="http://129.234.201.28/reuse/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
  - <types>
    - <s:schema elementFormDefault="qualified"
         targetNamespace="http://129.234.201.28/reuse/">
      - <s:element name="insertCode">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="0" maxOccurs="1"
                 name="iaSearchMethod"
                 type="s0:ArrayOfInt" />
              <s:element minOccurs="0" maxOccurs="1"
                 name="baUpload"
                 type="s:base64Binary" />
            </s:sequence>
          </s:complexType>
        </s:element>
      - <s:complexType name="ArrayOfInt">
        - <s:sequence>
            <s:element minOccurs="0"
               maxOccurs="unbounded" name="int"
               type="s:int" />
          </s:sequence>
        </s:complexType>
      - <s:element name="insertCodeResponse">
          <s:complexType />
        </s:element>
      </s:schema>
    </types>
  - <message name="insertCodeSoapIn">
      <part name="parameters" element="s0:insertCode" />
    </message>
  - <message name="insertCodeSoapOut">
      <part name="parameters"
         element="s0:insertCodeResponse" />
    </message>
  - <portType name="insertingcodeSoap">
    - <operation name="insertCode">
        <input message="s0:insertCodeSoapIn" />
        <output message="s0:insertCodeSoapOut" />
      </operation>
    </portType>
  - <binding name="insertingcodeSoap"
      type="s0:insertingcodeSoap">
      <soap:binding
         transport="http://schemas.xmlsoap.org/soap/http"
         style="document" />
    - <operation name="insertCode">
        <soap:operation
           soapAction="http://129.234.201.28/reuse/insertC
           ode" style="document" />
      - <input>
          <soap:body use="literal" />
        </input>
      - <output>
          <soap:body use="literal" />
        </output>
      </operation>
    </binding>
  - <service name="insertingcode">
    - <port name="insertingcodeSoap"
         binding="s0:insertingcodeSoap">
        <soap:address
           location="http://localhost/reuse/insertingcode.as
           mx" />
      </port>
    </service>
  </definitions>
```
Figure 7.4-6: WSDL description for the web service '*InsertingCode.asmx*'

```
   <?xml version="1.0" encoding="utf-8" ?>
- <wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tns="http://tempuri.org/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    targetNamespace="http://tempuri.org/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
- <wsdl:types>
- <s:schema elementFormDefault="qualified"
    targetNamespace="http://tempuri.org/">
- <s:element name="insertComponent">
  <s:complexType />
   </s:element>
- <s:element name="insertComponentResponse">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="insertComponentResult"
    type="s:boolean" />
   </s:sequence>
   </s:complexType>
   </s:element>
- <s:element name="submitRating">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="iRating" type="s:int" />
   </s:sequence>
   </s:complexType>
   </s:element>
- <s:element name="submitRatingResponse">
  <s:complexType />
   </s:element>
- <s:element name="extractComponent">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="uniqueid" type="s:int" />
   </s:sequence>
   </s:complexType>
   </s:element>
- <s:element name="extractComponentResponse">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="0" maxOccurs="1" name="extractComponentResult"
    type="s:string" />
   </s:sequence>
   </s:complexType>
   </s:element>
   </s:schema>
   </wsdl:types>
- <wsdl:message name="insertComponentSoapIn">
  <wsdl:part name="parameters" element="tns:insertComponent" />
   </wsdl:message>
```

```
- <wsdl:message name="insertComponentSoapOut">
  <wsdl:part name="parameters" element="tns:insertComponentResponse" />
    </wsdl:message>
- <wsdl:message name="submitRatingSoapIn">
  <wsdl:part name="parameters" element="tns:submitRating" />
    </wsdl:message>
- <wsdl:message name="submitRatingSoapOut">
  <wsdl:part name="parameters" element="tns:submitRatingResponse" />
    </wsdl:message>
- <wsdl:message name="extractComponentSoapIn">
  <wsdl:part name="parameters" element="tns:extractComponent" />
    </wsdl:message>
- <wsdl:message name="extractComponentSoapOut">
  <wsdl:part name="parameters" element="tns:extractComponentResponse" />
    </wsdl:message>
- <wsdl:portType name="transactionsSoap">
- <wsdl:operation name="insertComponent">
  <wsdl:input message="tns:insertComponentSoapIn" />
  <wsdl:output message="tns:insertComponentSoapOut" />
    </wsdl:operation>
- <wsdl:operation name="submitRating">
  <wsdl:input message="tns:submitRatingSoapIn" />
  <wsdl:output message="tns:submitRatingSoapOut" />
    </wsdl:operation>
- <wsdl:operation name="extractComponent">
  <wsdl:input message="tns:extractComponentSoapIn" />
  <wsdl:output message="tns:extractComponentSoapOut" />
    </wsdl:operation>
    </wsdl:portType>
- <wsdl:binding name="transactionsSoap" type="tns:transactionsSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
     style="document" />
- <wsdl:operation name="insertComponent">
  <soap:operation soapAction="http://tempuri.org/insertComponent"
     style="document" />
- <wsdl:input>
  <soap:body use="literal" />
    </wsdl:input>
- <wsdl:output>
  <soap:body use="literal" />
    </wsdl:output>
    </wsdl:operation>
- <wsdl:operation name="submitRating">
  <soap:operation soapAction="http://tempuri.org/submitRating"
     style="document" />
- <wsdl:input>
  <soap:body use="literal" />
    </wsdl:input>
- <wsdl:output>
  <soap:body use="literal" />
    </wsdl:output>
    </wsdl:operation>
- <wsdl:operation name="extractComponent">
```

```
  <soap:operation soapAction="http://tempuri.org/extractComponent"
     style="document" />
- <wsdl:input>
  <soap:body use="literal" />
    </wsdl:input>
- <wsdl:output>
  <soap:body use="literal" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
- <wsdl:service name="transactions">
  <documentation xmlns="http://schemas.xmlsoap.org/wsdl/" />
- <wsdl:port name="transactionsSoap" binding="tns:transactionsSoap">
  <soap:address location="http://cs201-028/reuse/transactions.asmx" />
    </wsdl:port>
    </wsdl:service>
    </wsdl:definitions>
```

Figure 7.4-7: WSDL for the web service '*extractcomponent.asmx*'

Appendix 3 Architecture

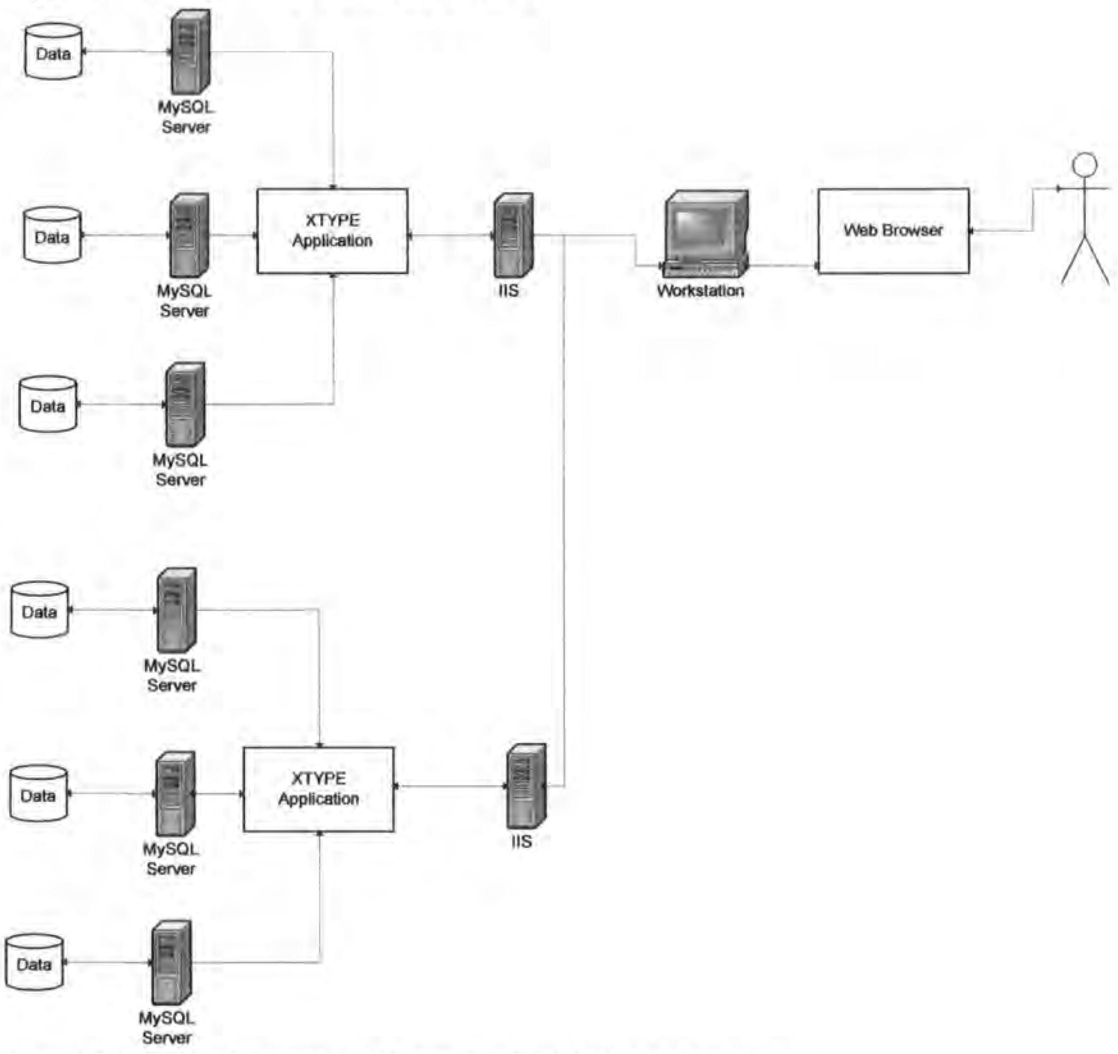


Figure 7.4-8: An Example of Extended XTYPE Architecture

Appendix 4 XSLT Descriptions

```
  <?xml version="1.0" ?>
- <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
- <xsl:template match="/">
- <HTML xmlns="http://www.w3.org/1999/xhtml">
- <HEAD>
  <link href="normalstyle.css" rel="stylesheet" type="text/css" />
    </HEAD>
- <BODY>
- <TABLE cellspacing="3" cellpadding="8"
    style="behavior:url(tablefunctions.htc);" slcolor="#FFFFCC"
    hlcolor="#BEC5DE">
- <THEAD>
- <TD class="heading">
  <B>Field Name</B>
    </TD>
- <TD class="heading">
  <B>Return Type</B>
    </TD>
- <TD class="heading">
  <B>Accessibility</B>
    </TD>
    </THEAD>
- <TBODY>
- <xsl:for-each select="Sourcecode/Table">
- <xsl:if test="ClassID[.=12]">
- <xsl:for-each select="Components/Component">
- <tr>
- <TD>
  <xsl:value-of select="Name" />
    </TD>
- <TD>
  <xsl:value-of select="ReturnType" />
    </TD>
- <TD>
  <xsl:value-of select="Accessibility" />
    </TD>
    </tr>
    </xsl:for-each>
    </xsl:if>
    </xsl:for-each>
    </TBODY>
    </TABLE>
    </BODY>
    </HTML>
    </xsl:template>
    </xsl:stylesheet>
```

Figure 7.4-1: XSLT description for '*viewcomponents.xslt*'

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:template match="/">
<HTML xmlns="http://www.w3.org/1999/xhtml">
<HEAD>
<link href="normalstyle.css" rel="stylesheet" type="text/css"/>
</HEAD>
<BODY>

<TABLE cellspacing="3" cellpadding="8"
style="behavior:url(tablefunctions.htc);" slcolor="#FFFFCC"
hlcolor="#BEC5DE">
<THEAD>
<TD></TD>
<TD class="heading"><B>Class ID</B></TD>
<TD class="heading"><B>Class Name</B></TD>
<TD class="heading"><B>Package</B></TD>
<TD class="heading"><B>Interfaces</B></TD>
<TD class="heading"><B>Inherits</B></TD>
<TD class="heading"><B>Design Pattern</B></TD>
<TD class="heading"><B>Abstract</B></TD>
<TD class="heading"><B>Static</B></TD>
<TD class="heading"><B># Comments</B></TD>
</THEAD>
<TBODY>
	<xsl:variable name ="ClassID"
select="structure/class/ClassID"/>
	<tr>
		<TD>
			<form action="extract.aspx" method="post"
name="extract" target="_self">
				<input type="hidden" name="ClassID"
value="{$ClassID}" />
				<input type="submit" name="extract"
value="Extract" />
			</form>
		</TD>
		<TD>
			<xsl:value-of select="structure/class/ClassID"/>
		</TD>
		<TD>
			<xsl:value-of select="structure/class/Classname"/>
		</TD>
		<TD>
			<xsl:if test="Package!=';'"><xsl:value-of
select="Package"/></xsl:if>
		</TD>
		<TD>
			<xsl:choose>
				<xsl:when test="structure/class/Interfaces[.
!=';']">
					<xsl:value-of
select="structure/class/Interfaces"/>
				</xsl:when>
				<xsl:otherwise>None
Implemented</xsl:otherwise>
			</xsl:choose>
		</TD>
		<TD>
			<xsl:choose>
```

```
                        <xsl:when test="structure/class/Inherits[.
!=';']">
                              <xsl:value-of
select="structure/class/Inherits"/>
                        </xsl:when>
                        <xsl:otherwise>None Inherited</xsl:otherwise>
                  </xsl:choose>
            </TD>
            <TD>
                  <xsl:if test="structure/class/DesignPattern!=';'">
                        <xsl:variable name ="DesignPattern"
select="structure/class/DesignPattern"/>
                        <a
href="viewdesignpattern.aspx?designpattern={$DesignPattern}"><xsl:val
ue-of select="structure/class/DesignPattern"/></a>

                  </xsl:if>
            </TD>
            <TD>
                  <xsl:choose>
                        <xsl:when test="structure/class/Abstract[.
!='0']">
                              Yes
                        </xsl:when>
                        <xsl:otherwise>No</xsl:otherwise>
                  </xsl:choose>
            </TD>
            <TD>
                  <xsl:choose>
                        <xsl:when test="structure/class/Static[.
!='0']">
                              Yes
                        </xsl:when>
                        <xsl:otherwise>No</xsl:otherwise>
                  </xsl:choose>
            </TD>
            <TD>
                  <xsl:value-of
select="structure/class/TotalNumberOfComments"/>
            </TD>
      </tr>
</TBODY>
</TABLE>
<br/>
<xsl:choose>
   <xsl:when test="structure/traceability/ClassID[. !='0']">
      <xsl:for-each select="structure/traceability">
   <xsl:variable name ="TraceOCL" select="OCLID"/>
   <xsl:variable name ="TraceClass" select="ClassID"/>
            <xsl:choose>
                  <xsl:when test="structure/traceability/ClassID[.
!='{$ClassID}']">
                        This component traces back to <a
href="viewclass.aspx?ClassID={$TraceOCL}"><xsl:value-of
select="TraceOCL"/></a>
                        </xsl:when>
                  <xsl:otherwise>
                        <a
href="viewclass.aspx?ClassID={$TraceClass}"><xsl:value-of
select="ClassID"/></a> has be traced to this to component
                  </xsl:otherwise>
            </xsl:choose>
      </xsl:for-each>
      </xsl:when>
      <xsl:otherwise>This component cannot be traced to anoth0u114
?</xsl:otherwise>
</xsl:choose>
<br/>
<xsl:choose>
   <xsl:when test="structure/field/FieldID[. !='']">
      <TABLE cellspacing="3" cellpadding="8"
      style="behavior:url(tablefunctions.htc);" slcolor="#FFFFCC"
hlcolor="#BEC5DE">
      <THEAD class="heading">
      <TD ><B>Field Name</B></TD>
      <TD><B>Type</B></TD>
      <TD><B>Accessibility</B></TD>
      <TD><B>Static</B></TD>
      </THEAD>
      <TBODY>
      <xsl:for-each select="structure/field">
      <TR>
            <TD>
                  <xsl:value-of select="Name"/>
            </TD>
            <TD>
                  <xsl:value-of select="Type"/>
            </TD>
            <TD>
                  <xsl:value-of select="Accessibility"/>
            </TD>
            <TD>
                  <xsl:choose>
                        <xsl:when test="Static[. !='0']">
                              Yes
                        </xsl:when>
                        <xsl:otherwise>No</xsl:otherwise>
                  </xsl:choose>
            </TD>
      </TR>
      </xsl:for-each>
      </TBODY>
      </TABLE>
      </xsl:when>
      <xsl:otherwise>No Methods Found</xsl:otherwise>
</xsl:choose>
<br/>
<xsl:choose>
   <xsl:when test="structure/method/Name[. !='']">
            <TABLE cellspacing="3" cellpadding="8"
style="behavior:url(tablefunctions.htc);" slcolor="#FFFFCC"
hlcolor="#BEC5DE">
            <THEAD>
            <TD class="heading"><B>Field Name</B></TD>
            <TD class="heading"><B>Return Type</B></TD>
            <TD class="heading"><B>Accessibility</B></TD>
            <TD class="heading"><B>Parameters</B></TD>
            </THEAD>
            <TBODY>
            <xsl:for-each select="structure/method">
                  <tr>
                        <TD>
```

```
                                    <xsl:value-of select="Name"/>
                              </TD>
                              <TD>
                                    <xsl:value-of select="ReturnType"/>
                              </TD>
                              <TD>
                                    <xsl:value-of select="Accessibility"/>
                              </TD>
                              <TD>
                                    <xsl:value-of select="Parameters"/>
                              </TD>
                        </tr>
                  </xsl:for-each>
                  </TBODY>
            </TABLE>
</xsl:when>
<xsl:otherwise>No Methods Found</xsl:otherwise>
</xsl:choose>

<br/>
<a href="viewsearchresults.aspx">Return To Results</a>
<br/>
<a href="searchform.aspx">Search Again</a>

</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

Figure 7.4-2: XSLT description for '*viewclass.xslt*'

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">

<xsl:template match="/">
<HTML xmlns="http://www.w3.org/1999/xhtml">
<HEAD>
<link href="normalstyle.css" rel="stylesheet" type="text/css"/>
</HEAD>
<BODY>
<TABLE cellspacing="3" cellpadding="8"
style="behavior:url(tablefunctions.htc);" slcolor="#FFFFCC"
hlcolor="#BEC5DE">
<THEAD>
<TD></TD>
<TD class="heading"><B>Class ID</B></TD>
<TD class="heading"><B>Class Name</B></TD>
<TD class="heading"><B>Package</B></TD>
<TD class="heading"><B>Interfaces</B></TD>
<TD class="heading"><B>Inherits</B></TD>
<TD class="heading"><B>Design Pattern</B></TD>
<TD class="heading"><B>Abstract</B></TD>
<TD class="heading"><B>Static</B></TD>
<TD class="heading"><B># Comments</B></TD>
<TD class="heading"><B>Fields</B></TD>
<TD class="heading"><B>Methods</B></TD>

</THEAD>
<TBODY>
```

```
<xsl:for-each select="structure/class">
      <xsl:variable name ="ClassID" select="ClassID"/>

      <tr>
            <TD>
                  <form action="extract.aspx" method="post"
name="extract" target="_self">
                        <input type="hidden" name="ClassID"
value="{$ClassID}" />
                        <input type="submit" name="extract"
value="Extract" />
                  </form>
            </TD>
            <TD>
                  <xsl:value-of select="ClassID"/>
            </TD>
            <TD>
            <a href="viewclass.aspx?ClassID={ClassID}">
                  <xsl:value-of select="Classname"/>
            </a>
            </TD>
            <TD>
                  <xsl:if test="Package!=';'"><xsl:value-of
select="Package"/></xsl:if>
            </TD>
            <TD>
                  <xsl:choose>
                        <xsl:when test="structure/class/Interfaces[.
!=';']">
                              <xsl:value-of
select="structure/class/Interfaces"/>
                        </xsl:when>
                        <xsl:otherwise>None
Implemented</xsl:otherwise>
                  </xsl:choose>
            </TD>
            <TD>
                  <xsl:choose>
                        <xsl:when test="structure/class/Inherits[.
!=';']">
                              <xsl:value-of
select="structure/class/Inherits"/>
                        </xsl:when>
                        <xsl:otherwise>None Inherited</xsl:otherwise>
                  </xsl:choose>
            </TD>
            <TD>
                  <xsl:value-of select="DesignPattern"/>
            </TD>
            <TD>
                  <xsl:choose>
                        <xsl:when test="Abstract[. !='0']">
                              Yes
                        </xsl:when>
                        <xsl:otherwise>No</xsl:otherwise>
                  </xsl:choose>
            </TD>
            <TD>
                  <xsl:choose>
                        <xsl:when test="Static[. !='0']">
                              Yes
                        </xsl:when>
                        <xsl:otherwise>No</xsl:otherwise>
                  </xsl:choose>
            </TD>
            <TD>
                  <xsl:value-of select="TotalNumberOfComments"/>
            </TD>
            <TD>
                  <form action="viewfields.aspx" method="post"
name="ClassID" target="_self">
                  <input type = "hidden" name="classID"
value="{$ClassID}" />
                  <input type="submit" value="View" />
                  </form>
            </TD>
            <TD>
                  <form action="viewcomponents.aspx" method="post"
name="ClassID" target="_self">
                  <input type = "hidden" name="classID"
value="{$ClassID}" />
                  <input type="submit" value="View" />
                  </form>
            </TD>
      </tr>
</xsl:for-each>

</TBODY>
</TABLE>
<br/>
<a href="viewsearchresults.aspx">Return To Results</a>
<br/>
<a href="searchform.aspx">Search Again</a>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

Figure 1.1-3: XSLT description for '*viewdesignpattern.xslt*'

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">

<xsl:template match="/">
<HTML xmlns="http://www.w3.org/1999/xhtml">
<HEAD>
<link href="normalstyle.css" rel="stylesheet" type="text/css"/>
</HEAD>
<BODY>
<TABLE cellspacing="3" cellpadding="8"
style="behavior:url(tablefunctions.htc);" slcolor="#FFFFCC"
hlcolor="#BEC5DE">
<THEAD>
<TD></TD>
<TD class="heading"><B>Rating</B></TD>
<TD class="heading"><B>Class ID</B></TD>
<TD class="heading"><B>Class Name</B></TD>
<TD class="heading"><B>Package</B></TD>
<TD class="heading"><B>Interfaces</B></TD>
<TD class="heading"><B>Inherits</B></TD>
```

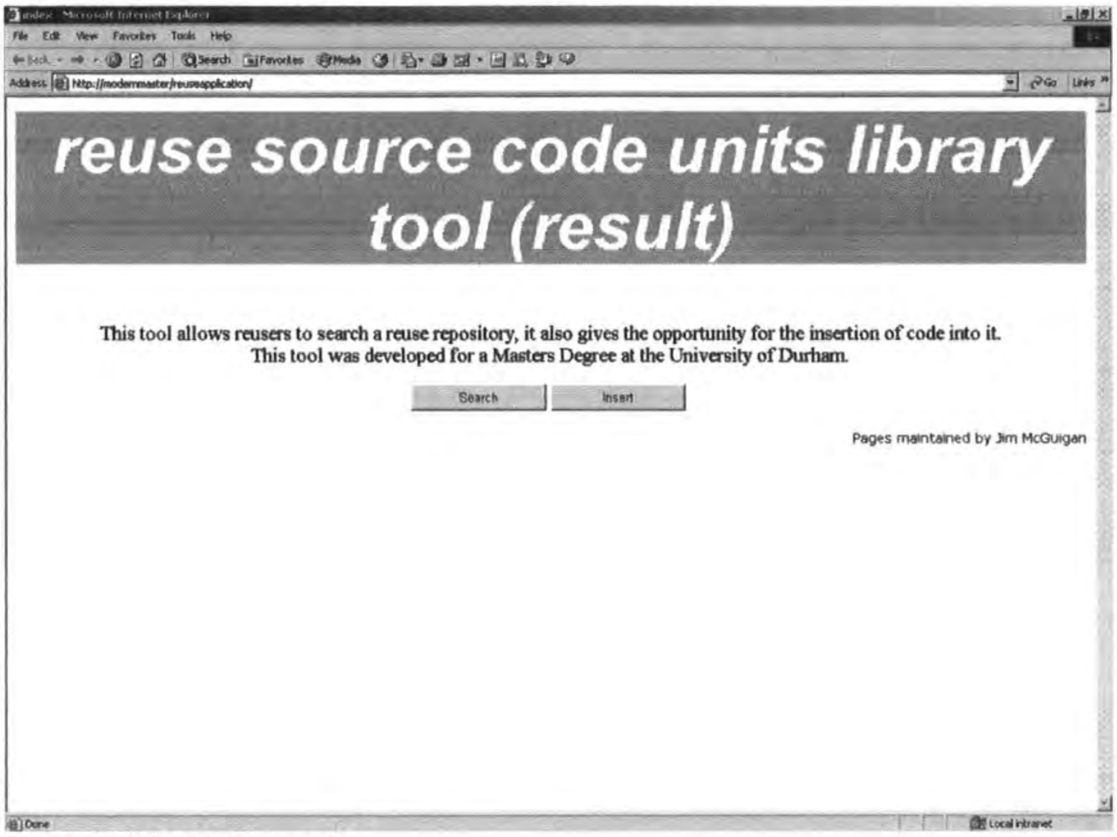
```
<TD class="heading"><B>Design Pattern</B></TD>
<TD class="heading"><B>Abstract</B></TD>
<TD class="heading"><B>Static</B></TD>
<TD class="heading"><B># Comments</B></TD>
<TD class="heading"><B>Fields</B></TD>
<TD class="heading"><B>Methods</B></TD>

</THEAD>
<TBODY>

<xsl:for-each select="structure/class">
      <xsl:variable name ="ClassID" select="ClassID"/>
      <tr>
            <form action="extract.aspx" method="post" name="extract">

                  <TD>
                        <input type="hidden" name="ClassID"
value="{$ClassID}" />
                        <input type="submit" name="extract"
value="Extract" />
                  </TD>
            </form>
            <TD>
                  <xsl:value-of select="Rating"/>%
            </TD>
            <TD>
                  <xsl:value-of select="ClassID"/>
            </TD>
            <TD>
            <a href="viewclass.aspx?ClassID={ClassID}">
                  <xsl:value-of select="Classname"/>
            </a>
            </TD>
            <TD>
                  <xsl:if test="Package!=';'"><xsl:value-of
select="Package"/></xsl:if>
            </TD>
            <TD>
                  <xsl:value-of select="Interfaces"/>
            </TD>
            <TD>
                  <xsl:value-of select="Inherits"/>
            </TD>
            <TD>
                  <xsl:if test="DesignPattern!=';'">
                        <xsl:variable name ="DesignPattern"
select="DesignPattern"/>
                        <a
href="viewdesignpattern.aspx?designpattern={$DesignPattern}"><xsl:val
ue-of select="DesignPattern"/></a>
                  </xsl:if>


            </TD>
            <TD>
                  <xsl:choose>
                        <xsl:when test="Abstract[. !='0']">
                              Yes
                        </xsl:when>
                        <xsl:otherwise>No</xsl:otherwise>
                  </xsl:choose>
            </TD>
            <TD>
                  <xsl:choose>
                        <xsl:when test="Static[. !='0']">
                              Yes
                        </xsl:when>
                        <xsl:otherwise>No</xsl:otherwise>
                  </xsl:choose>
            </TD>
            <TD>
                  <xsl:value-of select="TotalNumberOfComments"/>
            </TD>
            <TD>
                  <form action="viewfields.aspx" method="post"
name="ClassID" target="_self">
                  <input type = "hidden" name="classID"
value="{$ClassID}" />
                  <input type="submit" value="View" />
                  </form>
            </TD>
            <TD>
                  <form action="viewcomponents.aspx" method="post"
name="ClassID" target="_self">
                  <input type = "hidden" name="classID"
value="{$ClassID}" />
                  <input type="submit" value="View" />
                  </form>
            </TD>
      </tr>
</xsl:for-each>

</TBODY>
</TABLE>

<br/>
<a href="searchform.aspx">Search Again</a>

</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

Figure 1.1-4: XSLT description for '*viewsearchresults.xslt*'

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:template match="/">
<HTML xmlns="http://www.w3.org/1999/xhtml">
<HEAD>
<link href="normalstyle.css" rel="stylesheet" type="text/css"/>
</HEAD>
<BODY>

<xsl:choose>
   <xsl:when test="structure/field/FieldID[. !='']">
      <TABLE cellspacing="3" cellpadding="8"
      style="behavior:url(tablefunctions.htc);" slcolor="#FFFFCC"
hlcolor="#BEC5DE">
      <THEAD class="heading">
      <TD ><B>Field Name</B></TD>
      <TD><B>Type</B></TD>
      <TD><B>Accessibility</B></TD>
      <TD><B>Static</B></TD>
      </THEAD>
      <TBODY>
      <xsl:for-each select="structure/field">
      <TR>
            <TD>
                  <xsl:value-of select="Name"/>
            </TD>
            <TD>
                  <xsl:value-of select="Type"/>
            </TD>
            <TD>
                  <xsl:value-of select="Accessibility"/>
            </TD>
            <TD>
                  <xsl:choose>
                        <xsl:when test="Static[. !='0']">
                              Yes
                        </xsl:when>
                        <xsl:otherwise>No</xsl:otherwise>
                  </xsl:choose>
            </TD>
      </TR>
      </xsl:for-each>
      </TBODY>
      </TABLE>
      </xsl:when>
      <xsl:otherwise>No Methods Found</xsl:otherwise>
</xsl:choose>

<br/>
<a href="viewsearchresults.aspx">Return To Results</a>
<br/>
<a href="searchform.aspx">Search Again</a>

</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

Figure 1.1-5: XSLT description for '*viewFields.xslt*'

Appendix 5 ReSULT Screenshots


index - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://modernmaster/reuseapplication/
reuse source code units library tool (result)
Search
Insert
Done
Local intranet

This tool allows reusers to search a reuse repository, it also gives the opportunity for the insertion of code into it.
This tool was developed for a Masters Degree at the University of Durham.

Pages maintained by Jim McGuigan

Figure 1.1-6: '*Index.aspx*'

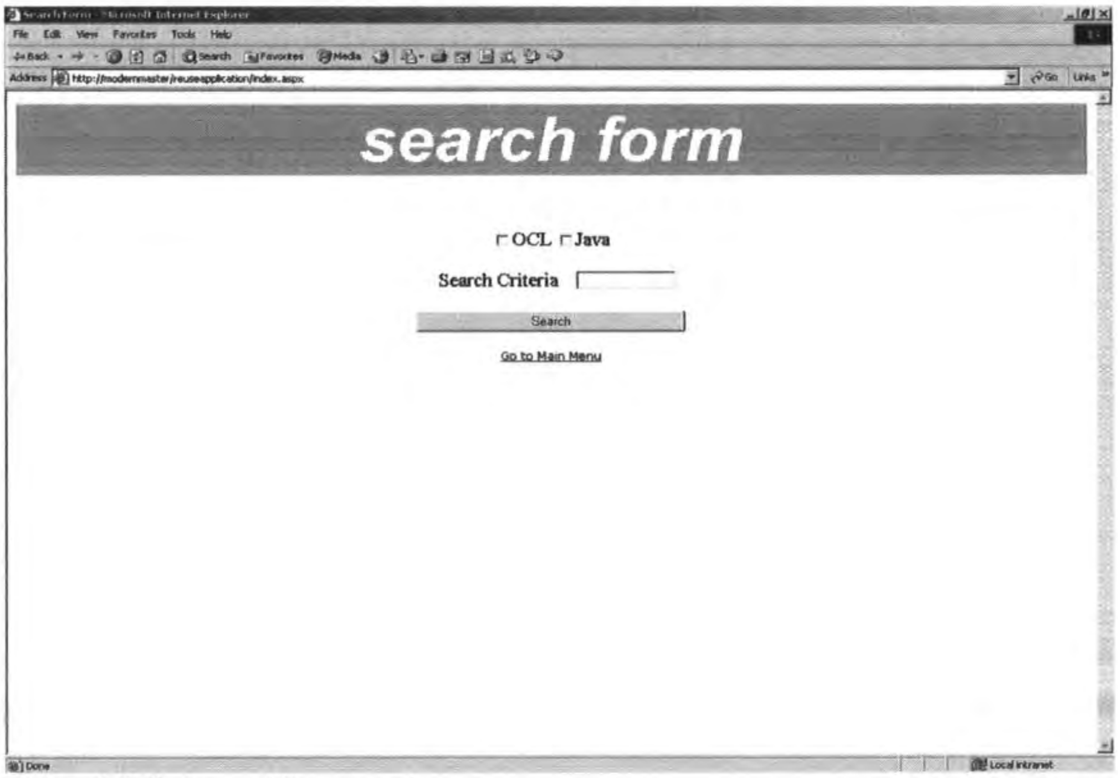
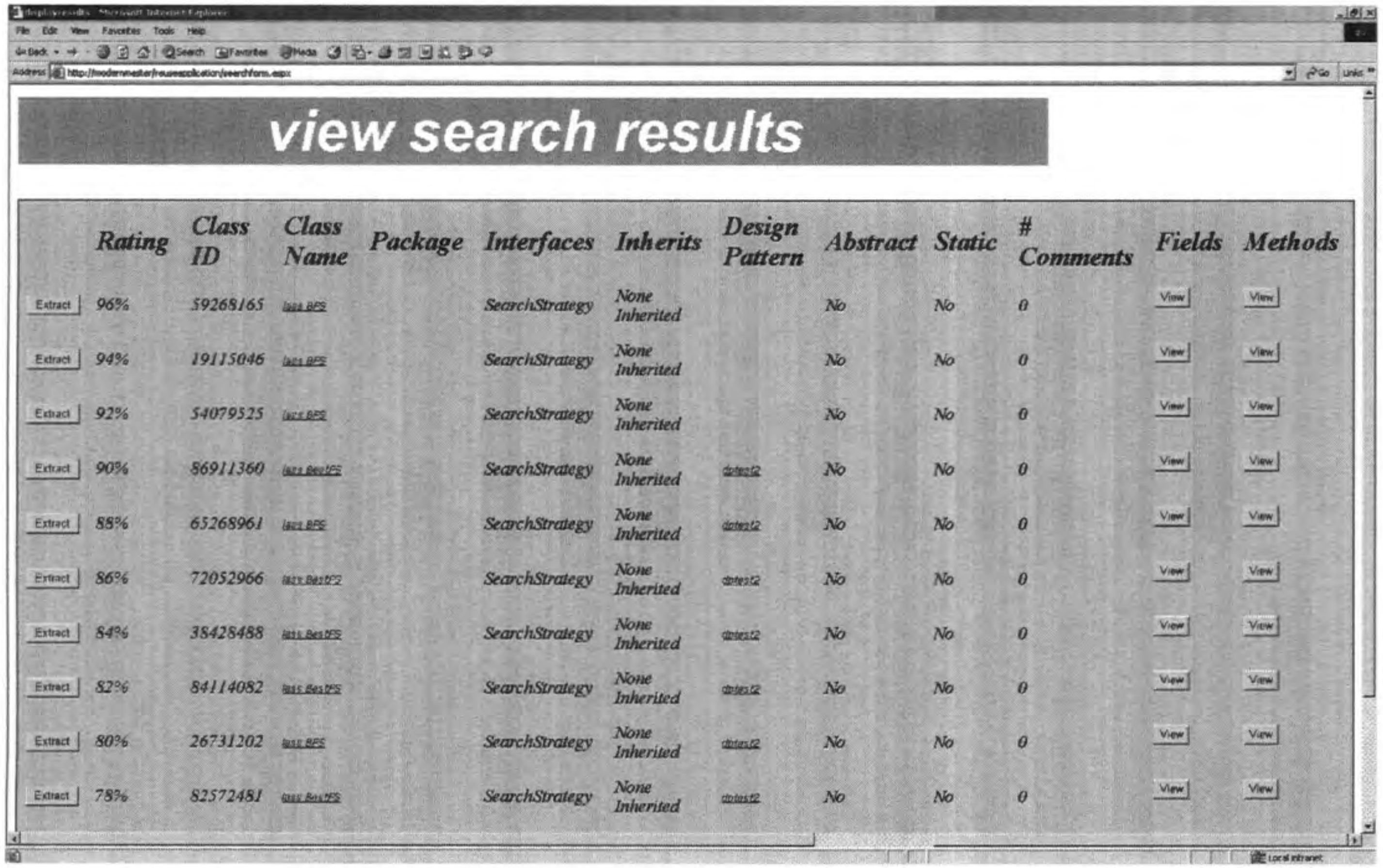
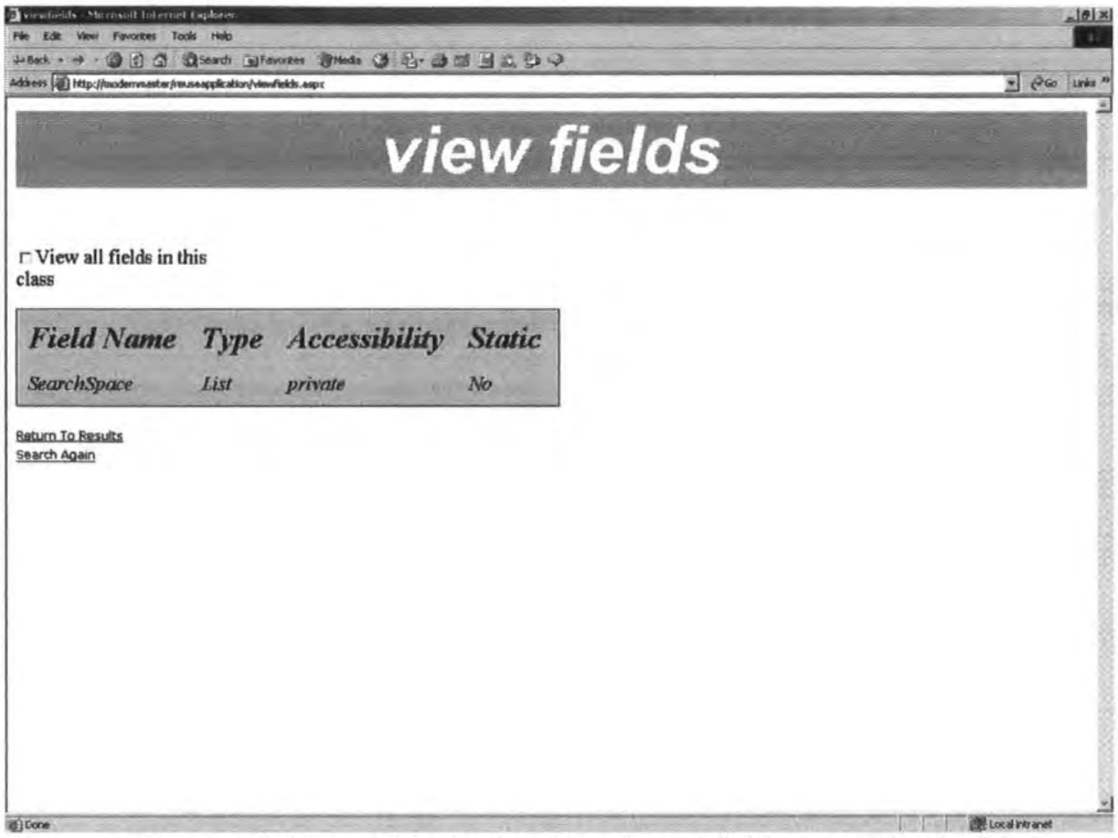

SearchForm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://modernmaster/reuseapplication/index.aspx
search form
OCL
Java
Search Criteria
Search
Go to Main Menu
Done
Local intranet

Figure 1.1-7: '*searchform.aspx*'



	Rating	Class ID	Class Name	Package	Interfaces	Inherits	Design Pattern	Abstract	Static	# Comments	Fields	Methods
Extract	96%	59268165	lass BFS		SearchStrategy	None Inherited		No	No	0	View	View
Extract	94%	19115046	lass BFS		SearchStrategy	None Inherited		No	No	0	View	View
Extract	92%	54079525	lass BFS		SearchStrategy	None Inherited		No	No	0	View	View
Extract	90%	86911360	lass BestFS		SearchStrategy	None Inherited	dptest2	No	No	0	View	View
Extract	88%	65268961	lass BFS		SearchStrategy	None Inherited	dptest2	No	No	0	View	View
Extract	86%	72052966	lass BestFS		SearchStrategy	None Inherited	dptest2	No	No	0	View	View
Extract	84%	38428488	lass BestFS		SearchStrategy	None Inherited	dptest2	No	No	0	View	View
Extract	82%	84114082	lass BestFS		SearchStrategy	None Inherited	dptest2	No	No	0	View	View
Extract	80%	26731202	lass BFS		SearchStrategy	None Inherited	dptest2	No	No	0	View	View
Extract	78%	82572481	lass BestFS		SearchStrategy	None Inherited	dptest2	No	No	0	View	View

Figure 1.1-8: viewresults.aspx (note: picture has had to be merged because of the page being to large for the screen)


viewfields - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://modernmaster/reuseapplication/viewfields.aspx

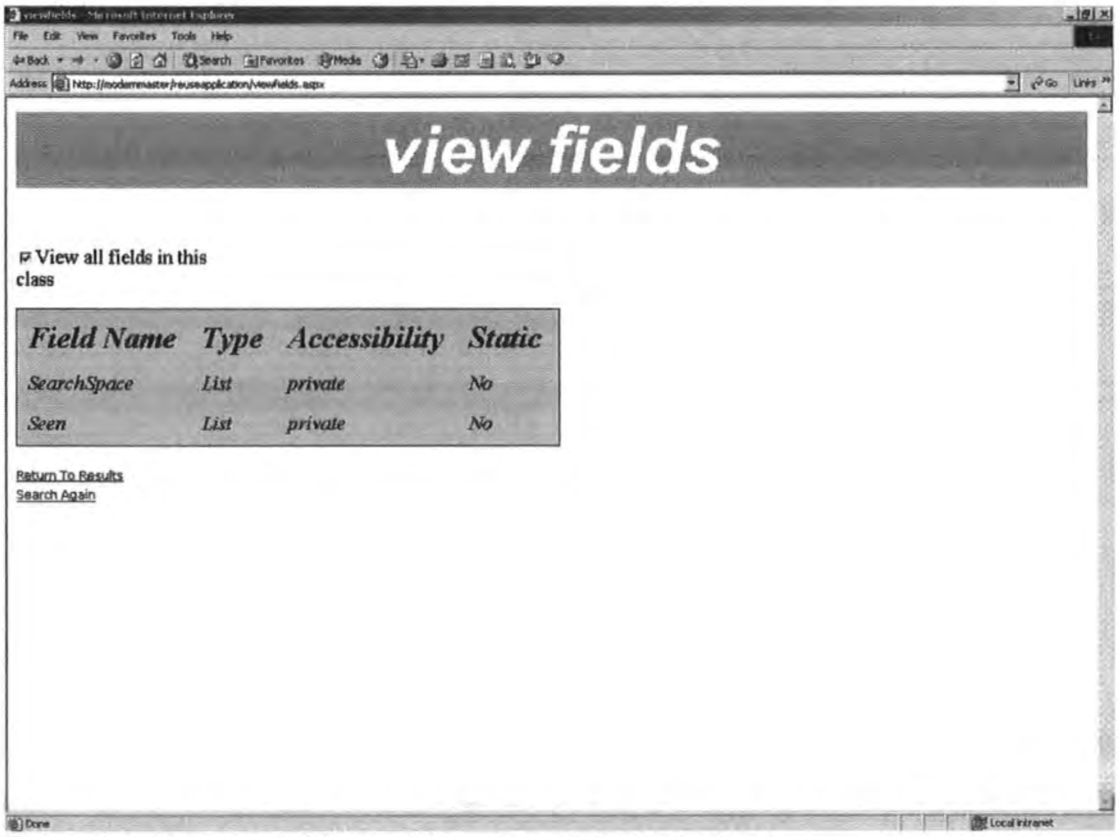
view fields

☐ View all fields in this class

Field Name	Type	Accessibility	Static
SearchSpace	List	private	No

Return To Results
Search Again

Figure 1.1-9: '*viewfields.aspx*' (only showing relevant fields to search criteria)


viewfields - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://modernmaster/reuseapplication/viewfields.aspx

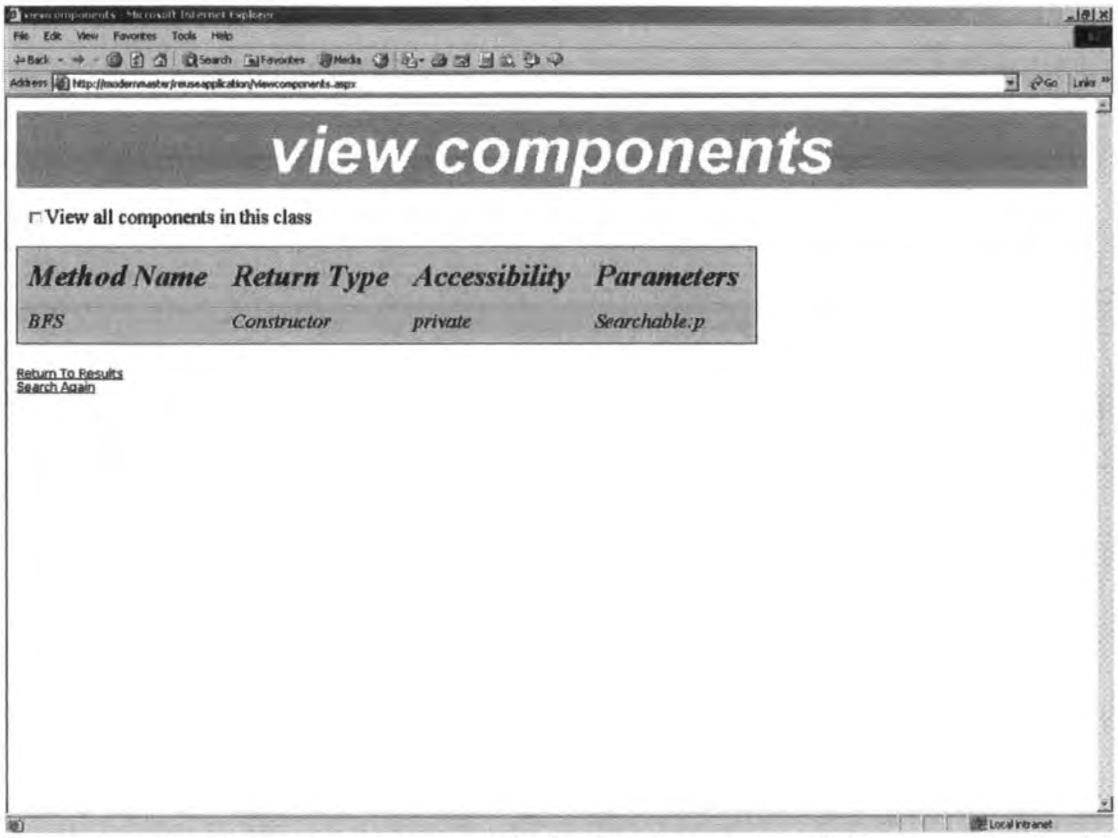
view fields

☑ View all fields in this class

Field Name	Type	Accessibility	Static
SearchSpace	List	private	No
Seen	List	private	No

Return To Results
Search Again

Figure 1.1-10: '*viewfields.aspx*' (showing all fields in class)



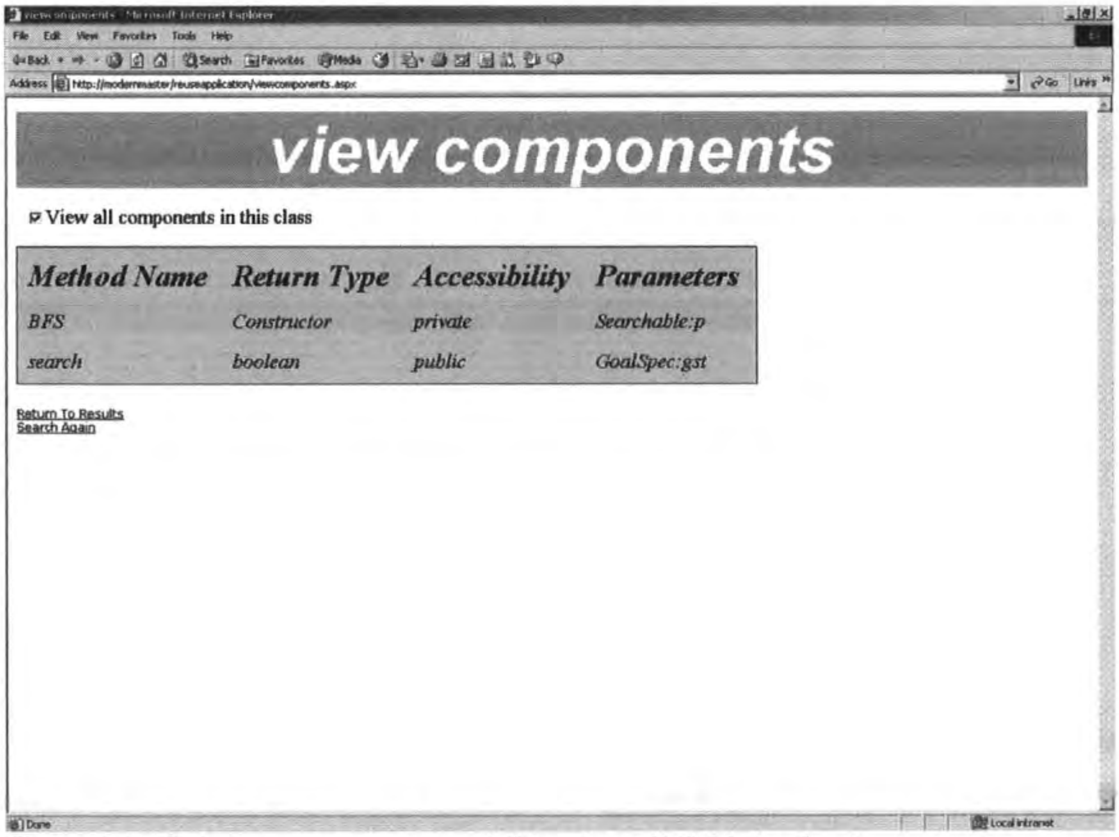
view components

☐ View all components in this class

Method Name	Return Type	Accessibility	Parameters
BFS	Constructor	private	Searchable:p

Return To Results
Search Again

Figure 1.1-11: '*viewcomponents.aspx*' (only showing relevant fields to search criteria)



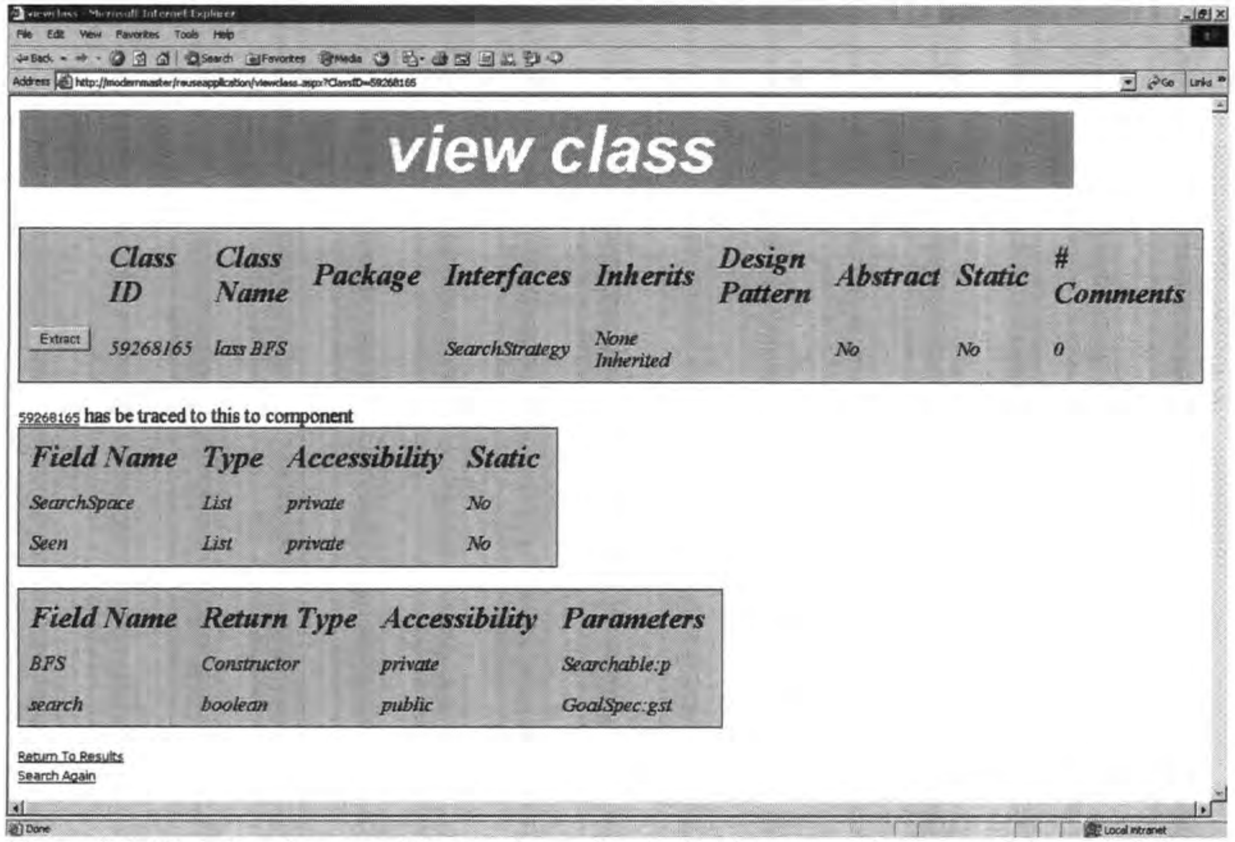
view components

☑ View all components in this class

Method Name	Return Type	Accessibility	Parameters
BFS	Constructor	private	Searchable:p
search	boolean	public	GoalSpec:gst

Return To Results
Search Again

Figure 1.1-12: '*viewcomponents.aspx*' (showing all fields in class)


viewclass - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://modernmaster/reuseapplication/viewclass.aspx?ClassID=59268165
Go Links
Done
Local intranet

view class

	Class ID	Class Name	Package	Interfaces	Inherits	Design Pattern	Abstract	Static	# Comments
Extract	*59268165*	*lass BFS*		*SearchStrategy*	*None Inherited*		*No*	*No*	*0*

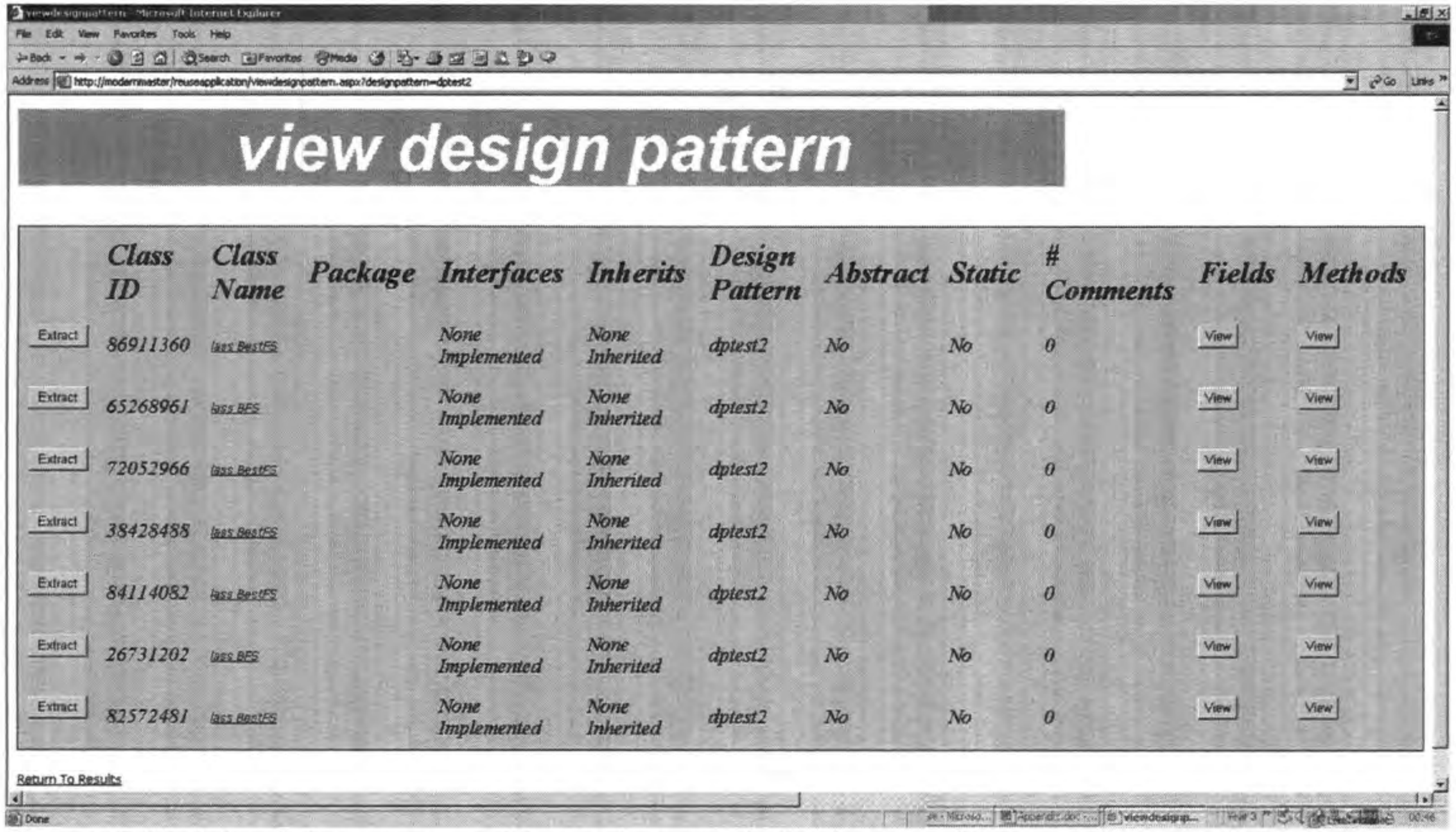
59268165 has be traced to this to component

Field Name	Type	Accessibility	Static
SearchSpace	*List*	*private*	*No*
Seen	*List*	*private*	*No*

Field Name	Return Type	Accessibility	Parameters
BFS	*Constructor*	*private*	*Searchable:p*
search	*boolean*	*public*	*GoalSpec:gst*

Return To Results
Search Again

Figure 1.1-13: '*viewclass.aspx*' (note: picture has had to be merged because of the page being to large for the screen)

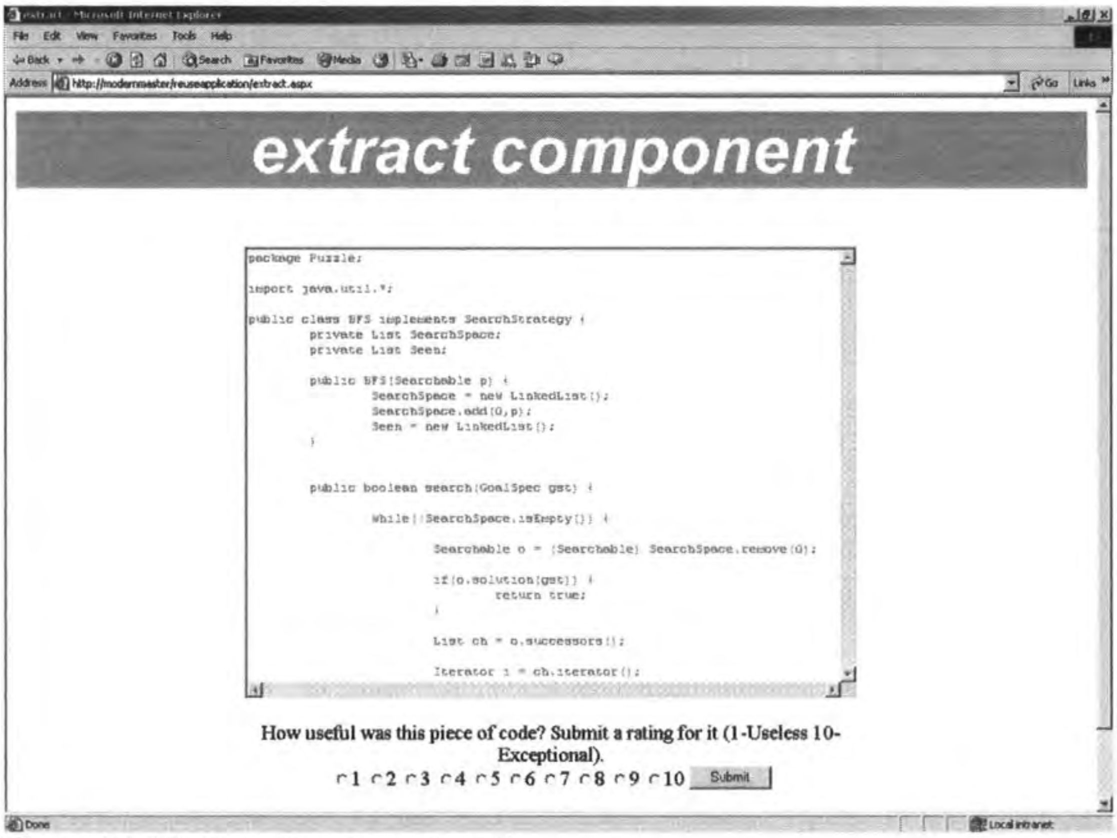

viewdesignpattern - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://modernmaster/reuseapplication/viewdesignpattern.aspx?designpattern=dptest2
Go Links
Done

view design pattern

	Class ID	Class Name	Package	Interfaces	Inherits	Design Pattern	Abstract	Static	# Comments	Fields	Methods
Extract	86911360	lass BestFS		None Implemented	None Inherited	dptest2	No	No	0	View	View
Extract	65268961	lass BFS		None Implemented	None Inherited	dptest2	No	No	0	View	View
Extract	72052966	lass BestFS		None Implemented	None Inherited	dptest2	No	No	0	View	View
Extract	38428488	lass BestFS		None Implemented	None Inherited	dptest2	No	No	0	View	View
Extract	84114082	lass BestFS		None Implemented	None Inherited	dptest2	No	No	0	View	View
Extract	26731202	lass BFS		None Implemented	None Inherited	dptest2	No	No	0	View	View
Extract	82572481	lass BestFS		None Implemented	None Inherited	dptest2	No	No	0	View	View

Return To Results

Figure 1.1-14: '*viewdesignpattern.aspx*' (note: picture has had to be merged because of the page being to large for the screen)



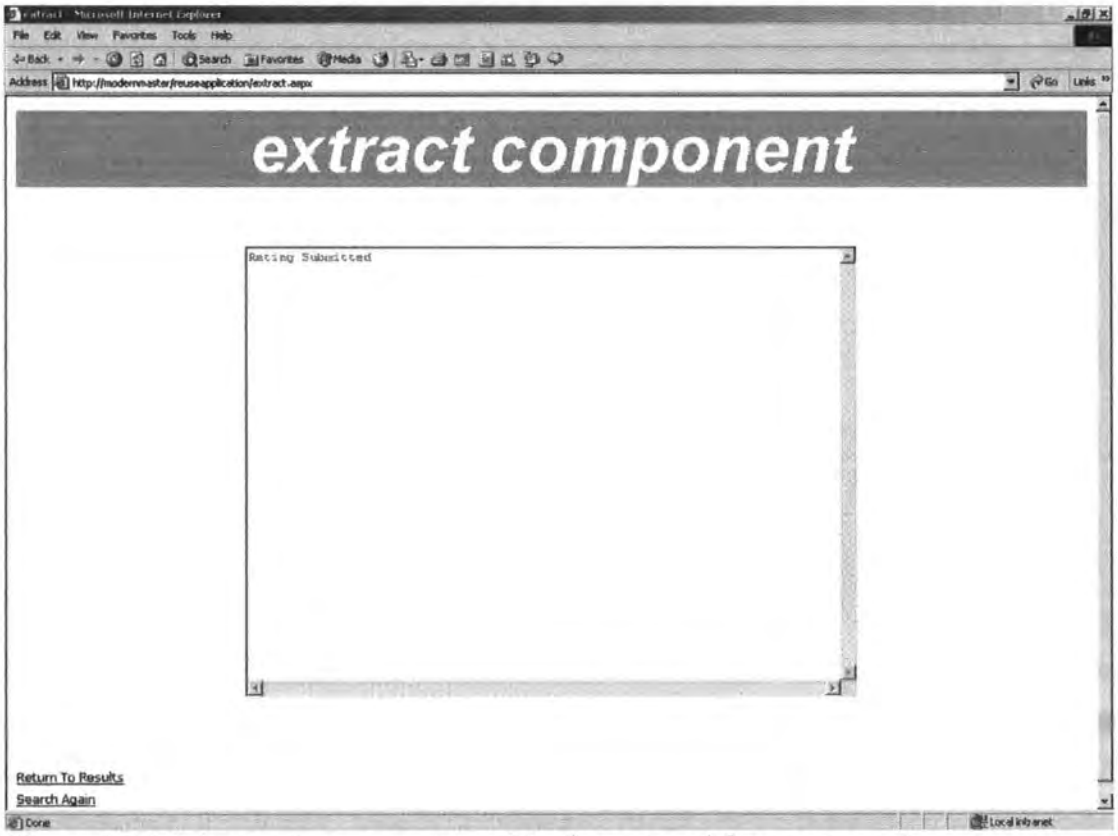
extract component

```
package Puzzle;

import java.util.*;

public class BFS implements SearchStrategy {
	private List SearchSpace;
	private List Seen;

	public BFS(Searchable p) {
		SearchSpace = new LinkedList();
		SearchSpace.add(0,p);
		Seen = new LinkedList();
	}


	public boolean search(GoalSpec gst) {

		While(!SearchSpace.isEmpty()) {

			Searchable o = (Searchable) SearchSpace.remove(0);

			if(o.solution(gst)) {
				return true;
			}

			List ch = o.successors();

			Iterator i = ch.iterator();
```

How useful was this piece of code? Submit a rating for it (1-Useless 10-Exceptional).

○1 ○2 ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10 Submit

Figure 1.1-15: '*extractcomponent.aspx*'

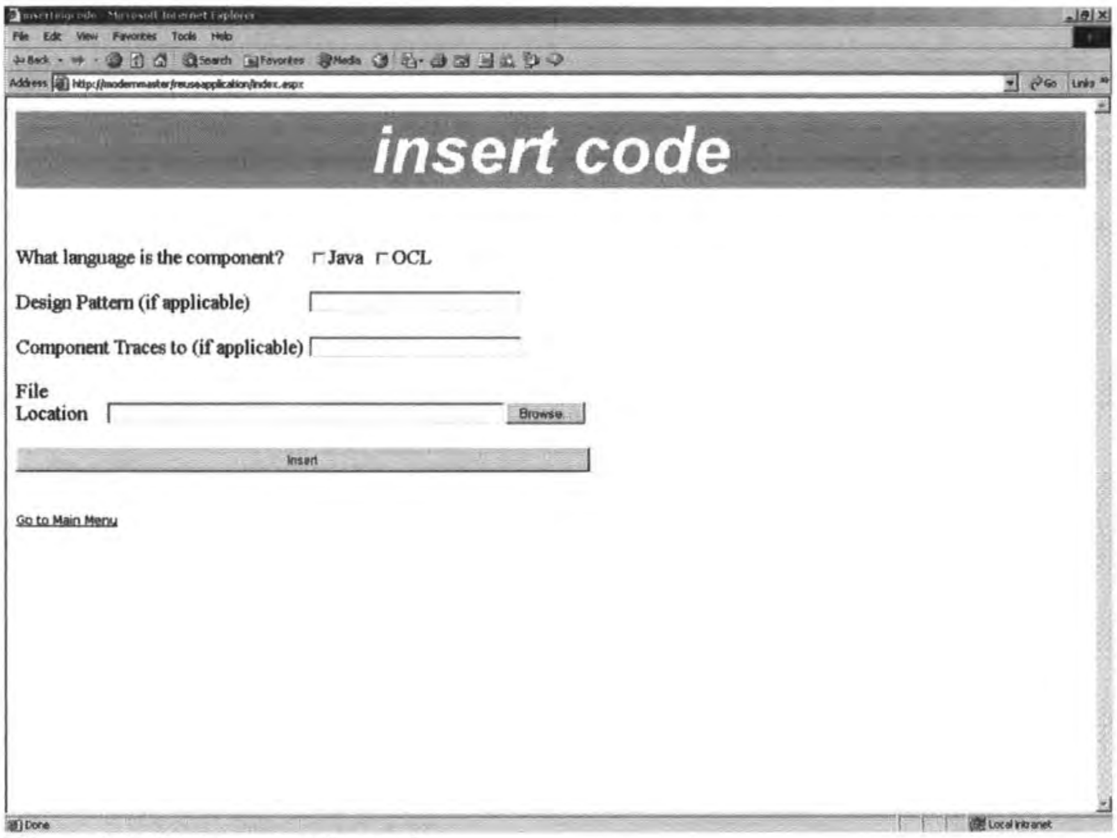


extract component

```
Rating Submitted
```

Return To Results
Search Again

Figure 1.1-16: '*extractcomponent.aspx*' (rating entered for component)


insertcode - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://modernmaster/reuseapplication/index.aspx
insert code
Browse...
Insert
Go to Main Menu
Done
Local intranet

What language is the component? ☐ Java ☐ OCL

Design Pattern (if applicable)

Component Traces to (if applicable)

File Location

Figure 1.1-17: '*insertcode.aspx*'

Appendix 6 XSD Descriptions

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="structure" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-
msdata">
    <xs:element name="structure" msdata:IsDataSet="true" msdata:Locale="en-GB">
        <xs:complexType>
            <xs:choice maxOccurs="unbounded">
                <xs:element name="class">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="ClassID" type="xs:int" />
                            <xs:element name="Classname">
                                <xs:simpleType>
                                    <xs:restriction base="xs:string">
                                        <xs:maxLength value="20" />
                                    </xs:restriction>
                                </xs:simpleType>
                            </xs:element>
                            <xs:element name="Package">
                                <xs:simpleType>
                                    <xs:restriction base="xs:string">
                                        <xs:maxLength value="40" />
                                    </xs:restriction>
                                </xs:simpleType>
                            </xs:element>
                            <xs:element name="Interfaces">
                                <xs:simpleType>
                                    <xs:restriction base="xs:string">
                                        <xs:maxLength value="20" />
                                    </xs:restriction>
                                </xs:simpleType>
                            </xs:element>
                            <xs:element name="Inherits">
                                <xs:simpleType>
                                    <xs:restriction base="xs:string">
                                        <xs:maxLength value="40" />
                                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="DesignPattern">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:maxLength value="40" />
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="Abstract" type="xs:int" />
            <xs:element name="Static" type="xs:int" />
            <xs:element name="TotalNumberOfComments" type="xs:int" />
            <xs:element name="Rating" type="xs:double" minOccurs="0" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="field">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="FieldID" msdata:AutoIncrement="true" type="xs:int" minOccurs="0"
/>
            <xs:element name="Name">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:maxLength value="20" />
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="Type">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:maxLength value="20" />
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="Accessibility">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:maxLength value="20" />
                    </xs:restriction>
```

```
                                </xs:simpleType>
                        </xs:element>
                        <xs:element name="Static" type="xs:int" />
                        <xs:element name="ClassID" type="xs:int" />
                </xs:sequence>
        </xs:complexType>
</xs:element>
<xs:element name="traceability">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="TraceID" type="xs:int" />
                        <xs:element name="ClassID" type="xs:int" />
                        <xs:element name="OCLID" type="xs:int" />
                </xs:sequence>
        </xs:complexType>
</xs:element>
<xs:element name="classtype">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="ClassID" type="xs:int" />
                        <xs:element name="Name" type="xs:string" minOccurs="0" />
                        <xs:element name="Type" type="xs:string" minOccurs="0" />
                </xs:sequence>
        </xs:complexType>
</xs:element>
<xs:element name="method">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="ComponentID" type="xs:int" />
                        <xs:element name="Name">
                                <xs:simpleType>
                                        <xs:restriction base="xs:string">
                                                <xs:maxLength value="20" />
                                        </xs:restriction>
                                </xs:simpleType>
                        </xs:element>
                        <xs:element name="ReturnType">
                                <xs:simpleType>
                                        <xs:restriction base="xs:string">
                                                <xs:maxLength value="20" />
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:element>
                        <xs:element name="Accessibility">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:maxLength value="20" />
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:element>
                        <xs:element name="Parameters">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:maxLength value="20" />
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:element>
                        <xs:element name="ClassID" type="xs:int" />
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
    </xs:choice>
</xs:complexType>
<xs:unique name="Constraint1" msdata:PrimaryKey="true">
    <xs:selector xpath=".//class" />
    <xs:field xpath="ClassID" />
</xs:unique>
<xs:unique name="field_Constraint1" msdata:ConstraintName="Constraint1">
    <xs:selector xpath=".//field" />
    <xs:field xpath="FieldID" />
</xs:unique>
<xs:unique name="method_Constraint1" msdata:ConstraintName="Constraint1" msdata:PrimaryKey="true">
    <xs:selector xpath=".//method" />
    <xs:field xpath="ComponentID" />
</xs:unique>
<xs:keyref name="Relation2" refer="Constraint1">
    <xs:selector xpath=".//method" />
    <xs:field xpath="ClassID" />
</xs:keyref>
<xs:keyref name="Relation1" refer="Constraint1">
```

```
                    <xs:selector xpath=".//field" />
                    <xs:field xpath="ClassID" />
                </xs:keyref>
            </xs:element>
        </xs:schema>
```

Appendix 7 OCL Translations

OCL Keyword	**ReSULT Translation**
Self	“ignore??”
>	Greater than
<	Less than
=	Is Equal to
:	Is of type
Inv	Invariant
Pre	Precondition
Post	Postcondition
::	Includes
Let	Let local variable
Def	Definition
.oclAsType	Casts to
->	Is
..	To and including
EmployeeRanking[bosses]	Set of employeeranking belonging to the collection of bosses
OclIsTypeOf	Is of type?
) :	Returns (watch out for number of spaces)
OclIsKindOf	Is of kind?
OclInState	Is of state?
OclIsNew	Is new?
oclAsType	Is of type
x@pre	At the precondtion value for x
Select	Selected with
Reject	Rejected for value
|	Where
Collect	Collected for
forAll	For all values of
<>	Are all unique to
Exists	Existed with value

Table 1.1-1: The translations between natural language used in the ReSULT system and OCL.

Appendix 8 ReSULT Interface Definitions

```
using System;
using System.Collections;

namespace reuse.insertcode
{
	/// <summary>
	/// Summary description for SourceCodeHandler.
	/// </summary>
	public interface
SourceCodeHandlerInterface:CodeHandlerInterface
	{
		void setStatic();
		void setAbstract();
		bool getStatic();
		bool getAbstract();
		ArrayList getInterfaces();
		void setInterfaces(string sDeclaration);
		ArrayList getImports();
		void addImport(string sStatement);
		void setInheritance(string sDeclaration);
		ArrayList getInheritance();
		string [] identifyMethod (string sSegment,int
iComponentID);
		ArrayList identifyMethodParams(string sSegment);
		ArrayList identifyBlocks(string sTemp);
		void identifyFields(string sSegment);
		void setSourceCodeObject();
		SourceCode getSourceCodeObject();
	}
}
```

Figure 1.1-18: '*CodeHandlerInterface.cs*'

```
using System;
using System.Collections;

namespace reuse.insertcode
{
	/// <summary>
	/// Summary description for CommentsInterface.
	/// </summary>
	public interface CommentsInterface
	{
		ArrayList identifyComments(int iComponentID,string
sSegment);
		void addLineComments(string sComment, int
iComponentID);
		void checkExpelledWords(string item,int
iLocationOfWord,int iComponentID);
	}
}
```

Figure 1.1-19: '*CommentsInterface.cs*'

```
using System;

namespace reuse.insertcode
{
      /// <summary>
      /// Summary description for CodeHandlerInterface.
      /// </summary>
      public interface CodeHandlerInterface
      {
            int getNumberOfWords();
            void setUniqueID();
            void setFields(string [] sField);
            string getPackageName();
            string getClassName();
            int getUniqueID();
            void setComponent(Component c);
            void setPackageName(string sName);
            void setClassname(string sTemp);
            string removeInvalidCharacters(string sWord);
      }
}
```

Figure 1.1-20: '*CodeHandlerInterface.cs*'

Appendix 9 Results Chapter

Metric ID	Metric	Properties	Goal Properties	Scoring Criteria	Weighting	Score
M1.1	Do code generation tools produce reusable source code that is documented?	Self-Descriptiveness	**Maintainability Testability Flexibility Portability Reusability**	Yes (1) No (0)	1/6	1
M1.2	Are comments accurate and describe the "what's and whys?"	Self-Descriptiveness	**Maintainability Testability Flexibility Portability Reusability**	0…5 0 = no comments. 5 = Precise Comments	1/30	2
M1.3	Are comments set off from code and of consistent style throughout?	Self-Descriptiveness	**Maintainability Testability Flexibility Portability Reusability**	0…5 0 = No Comments 5 = Consistent Style	1/30	4
M1.4	Is a standard format for organisations of modules implemented consistently?	Self-Descriptiveness	**Maintainability Testability Flexibility Portability Reusability**	0 No standard present 5 Consistent standard	1/30	4
M1.5	Is a standard prologue consistently implemented?	Self-Descriptiveness	**Maintainability Testability Flexibility Portability**	Yes (1) No (0)	1/6	1

			Reusability			
M1.6	Does the documentation specify a standard prologue?	Self-Descriptiveness	**Maintainability** **Testability** **Flexibility** **Portability** **Reusability**	Yes (1) No (0)	1/6	1
M2.1	Is there a representation of the design in the paper documentation?	Consistency	**Correctness** **Reliability** **Maintainability**	Yes (1) No (0)	1/15	1
M2.2	Is the software implemented in accordance with the design representation?	Consistency	**Correctness** **Reliability** **Maintainability**	Yes (1) No (0)	1/15	1
M2.3	Are there consistent global, unit, and data type definitions?	Consistency	**Correctness** **Reliability** **Maintainability**	Yes (1) No (0)	1/15	1
M2.4	Is there a definition of standard I/O handling in the paper documentation?	Consistency	**Correctness** **Reliability** **Maintainability**	Yes (1) No (0)	1/15	1
M2.5	Is there a consistent implementation of external I/O protocol and format for all units?	Consistency	**Correctness** **Reliability** **Maintainability**	Yes (1) No (0)	1/15	1
M2.6	Are data naming standards specified in the paper documentation?	Consistency	**Correctness** **Reliability** **Maintainability**	Yes (1) No (0)	1/15	1
M2.7	Are naming standards consistent across IPC calls?	Consistency	**Correctness** **Reliability** **Maintainability**	0…5 0 = Non Existent 5 =Standards always meet	1/75	2

M2.8	Are naming standards consistent across languages?	Consistency	**Correctness** **Reliability** **Maintainability**	0…5 0 = No Standards Met 5 = All Enforced	1/75	4
M2.9	Is there a standard for function naming in the paper documentation?	Consistency	**Correctness** **Reliability** **Maintainability**	Yes (1) No (0)	1/15	1
M2.10	Are the naming conventions consistent for functional groupings?	Consistency	**Correctness** **Reliability** **Maintainability**	Yes (1) No (0)	1/15	1
M2.11	Are the naming conventions consistent for usage?	Consistency	**Correctness** **Reliability** **Maintainability**	Yes (1) No (0)	1/15	1
M2.12	Are the naming conventions consistent for data type, etc.?	Consistency	**Correctness** **Reliability** **Maintainability**	Yes (1) No (0)	1/15	0
M2.13	Does the paper documentation establish accuracy requirements for all operations?	Consistency	**Correctness** **Reliability** **Maintainability**	Yes (1) No (0)	1/15	1
M2.14	Are there quantitative accuracy requirements stated in the paper documentation for all I.O?	Consistency	**Correctness** **Reliability** **Maintainability**	Yes (1) No (0)	1/15	1
M2.15	Are there quantitative accuracy requirements stated in the paper documentation for all constants?	Consistency	**Correctness** **Reliability** **Maintainability**	Yes (1) No (0)	1/15	1
M3.1	Is the structure of the design hierarchical in a top-down design within tasking threads?	Modularity	**Maintainability** **Testability** **Flexibility**	Yes (1) No (0)	1/9	1

			Reusability **Interoperability**			
M3.2	Do the functional groupings of units avoid calling units outside their functional area?	Modularity	**Maintainability** **Testability** **Flexibility** **Reusability** **Interoperability**	0…5 0 = Always 5 = Never	1/45	4
M3.3	Are machine dependent and I/O functions isolated and encapsulated?	Modularity	**Maintainability** **Testability** **Flexibility** **Reusability** **Interoperability**	Yes (1) No (0)	1/9	4
M3.4	Do all functional procedures represent one function (one-to-one function mapping)?	Modularity	**Maintainability** **Testability** **Flexibility** **Reusability** **Interoperability**	0…5 0 = Never 5 = Always	1/45	3
M3.5	Are all commercial software interfaces & APIs, other than GUI Builders, isolated and encapsulated?	Modularity	**Maintainability** **Testability** **Flexibility** **Reusability** **Interoperability**	Yes (1) No (0)	1/9	1
M3.6	Have symbolic constants been used in place of explicit ones?	Modularity	**Maintainability** **Testability** **Flexibility** **Reusability** **Interoperability**	Yes (1) No (0)	1/9	1
M3.7	Are all variables used exclusively for their declared purposes?	Modularity	**Maintainability** **Testability**	Yes (1) No (0)	1/9	1

			Flexibility **Reusability** **Interoperability**			
M3.8	Has the code been structured to minimise coupling to global variables?	Modularity	**Maintainability** **Testability** **Flexibility** **Reusability** **Interoperability**	Yes (1) No (0)	1/9	1
M3.9	Are interpreted code bodies protected from accidental or deliberate modification?	Modularity	**Maintainability** **Testability** **Flexibility** **Reusability** **Interoperability**	Yes (1) No (0)	1/9	0
M4.1	Is the data representation machine independent?	Machine Independence	**Portability** **Reusability**	Yes (1) No (0)	1/2	0
M4.2	Are the commercial software components available on other platforms in the same level of functionality?	Machine Independence	**Portability** **Reusability**	Yes (1) No (0)	1/2	1
M5.1	Does the software avoid all usage of specific pathnames/filenames?	Software system independence	**Portability** **Reusability**	Yes (1) No (0)	1/6	1
M5.2	Is the software free of machine, OS and vendor specific extensions?	Software system independence	**Portability** **Reusability**	Yes (1) No (0)	1/6	1
M5.3	Are system dependent functions, etc., in stand-alone modules (not embedded in code)?	Software system independence	**Portability** **Reusability**	Yes (1) No (0)	1/6	1
M5.4	Are the languages and interface libraries selected standardised and	Software system independence	**Portability** **Reusability**	Yes (1) No (0)	1/6	1

	portable?					
M5.5	Does the software avoid the need for any unique compilation in order to run?	Software system independence	**Portability** **Reusability**	Yes (1) No (0)	1/6	1
M5.6	Is the generated code able to run without a specific support runtime component?	Software system independence	**Portability** **Reusability**	Yes (1) No (0)	1/6	1
M6.1	How quickly does it take to complete one search of the reuse repository?	Execution Efficiency	**Efficiency**	1-x	1/3	0.7 to 1dp
M6.2	How long does it take to extract a component?	Execution Efficiency	**Efficiency**	1-x	1/3	0.9 to 1dp
M7.1	Are there restrictions to areas of the system?	Access Control	**Integrity**	Yes (1) No (0)	1/2	0
M7.2	Are there different levels of access?	Access Control	**Integrity**	Yes (1) No (0)	1/2	0
M8.1	Are actions of users monitored?	Access Audit	**Integrity**	Yes (1) No (0)	1/2	0
M8.2	Are individual actions of users logged?	Access Audit	**Integrity**	Yes (1) No (0)	1/2	0
M9.1	Are there training strategies for the system?	Training	**Usability**	Yes (1) No (0)	1/3	1
M9.2	Is there sufficient user documentation?	Training	**Usability**	Yes (1) No (0)	1/3	1
M9.3	Is intensive training need for users?	Training	**Usability**	Yes (1) No (0)	1/3	0
M10.1	Is the component suitable for the task?	Operability	**Usability**	Yes (1) No (0)	1/3	1
M10.2	Does the system conform to the	Operability	**Usability**	0-5	1/3	1

	users' expectation?					
M10.3	Is the system tolerant to errors?	Operability	**Usability**	Yes (1) No (0)	1/3	1
M11.1	Are the user interface forms self-descriptive?	Communicativeness	**Usability**	0…5 0 = None 5 = All are	1/10	3
M11.2	Has an effective colour scheme been used in the system that draws the users attention to important aspects of a screen?	Communicativeness	**Usability**	0…5 0 = Non Effective 5 = Meaningful	1/10	4
M12.1	Are interfaces designed into the system to allow 'plug and play' expansion?	Expandability	**Flexibility**	Yes (1) No (0)	1/3	1
M12.2	Has the system been design with inheritance?	Expandability	**Flexibility**	Yes (1) No (0)	1/3	1
M12.3	Are the structures of interfaces, and inheritance well documented?	Expandability	**Flexibility**	Yes (1) No (0)	1/3	1
M13.1	How long does it take to correct errors?	Simplicity	**Testability** **Reliability**	1-time taken	1	
M14.1	Does the system perform a broad range of functions?	Generality	**Flexibility** **Interoperability** **Reusability**	0…5 0 = Just one function 5 = Many varied functions	1/5	2
M15.1	Minimise the time to correct errors that occur in the system.	Conciseness	**Maintainability**	Yes (1) No (0)	1/3	1
M15.2	How long does it take for	Conciseness	**Maintainability**	1-x	1/3	

	someone else to read and understand code?					
M15.3	Has the reuse of code been applied in the development of the system?	Conciseness	**Maintainability**	Yes (1) No (0)	1/3	0
M16.1	Does the system implement open standards?	Communication Commonality	**Interoperability**	Yes (1) No (0)	1/3	1
M16.2	Is the system platform independent?	Communication Commonality	**Interoperability**	Yes (1) No (0)	1/3	1
M16.3	Is the tool operating system independent?	Communication Commonality	**Interoperability**	Yes (1) No (0)	1/3	1
M16.4	How easy is it to transfer the system to another environment?	Communication Commonality	**Interoperability**	0…5 0 = Impossible 5 = Seamless	1/15	4
M17.1	Can the data be compressed?	Storage Efficiency	**Efficiency**	Yes (1) No (0)	1	0
M18.1	Does the system implement all required capability defined during the analysis stage?	Completeness	**Correctness**	Yes (1) No (0)	1/2	1
M18.2	Does the system contain all references and required items?	Completeness	**Correctness**	Yes (1) No (0)	1/2	
M19.1	How well is the system implementation traced back to the defined use-cases set out in the analysis?	Traceability	**Correctness**	0…5 0 = Never 5 = All	1	5
M20.1	Number of incorrect search results found.	Accuracy	**Reliability**	0…5 0 = Many 5 = None	1/2	2
M20.2	Number of incorrect characters	Accuracy	**Reliability**	0…5	1/2	3

	parsed into the system when inserting code into the repository.			0 = Many 5 = None		
M21.1	Number of runtime errors occurred	Error tolerance	**Reliability**	0…5 0 = Many 5 = None	1	5

Table 7.4-2: The software metrics used to evaluate the ReSULT system

Access Audit	(A_a)
Access Control	(A_c)
Accuracy	(A_c)
Communication Commonality	(C_C)
Communicativeness	(C_{om})
Completeness	(C_{omp})
Conciseness	(C_{onc})
Consistency	(C_{ons})
Error tolerance	(E_t)
Execution Efficiency	(E_e)
Expandability	(E_x)
Generality	(G_e)
Instrumentation	(I_{ns})
Machine Independence	(M_i)
Modularity	(M_{od})
Operability	(O_p)
Self-descriptiveness	(S_D)
Simplicity	(S_i)
Software system independence	(S_{si})
Storage Efficiency	(S_e)
Traceability	(T_a)
Training	(T_r)

Table 7.4-3: Abbreviation table for Quality Factor Properties

<u>Maintainability</u> $M_a = \sum(C_{ons}+C_{onc}+ S_d+M_{od})$ <u>Flexibility</u> $F_l = \sum (S_d+E_x+G_e+M_{od})$ <u>Testability</u> $T_e = \sum (S_i+I_{ns}+S_d+M_{od})$ <u>Portability</u> $P_o = \sum (S_d +M_{od}+S_{si} +M_i)$ <u>Reusability</u> $R_{eu} =\sum (S_d+G_e+M_{od}+S_{si}+M_i)$ <u>Interoperability</u> $I_O = \sum (G_e+M_{od}+C_c)$	<u>Integrity</u> $I_n = \sum (A_c+A_a)$ <u>Efficiency</u> $E_f = \sum (E_e+S_e)$ <u>Usability</u> $U_s = \sum (O_p+T_r+C_{om})$ <u>Correctiveness</u> $C_o = \sum (T_a+C_{omp}+C_{on})$ <u>Reliability</u> $R_{el} = \sum (C_{on}+A_c+E_t)$

Table 7.4-4: Quality Factor Equations for McCall's Software Quality Model

Glossary

Adjective: The part of speech that modifies a noun or other substantive by limiting, qualifying, or specifying and distinguished in English morphologically by one of several suffixes, such as *-able, -ous, -er,* and *-est,* or syntactically by position directly preceding a noun or nominal phrase – [http://www.dictionary.com].

Artefact: An artefact is a man-made object taken as a whole – [http://workdnet.princeton.edu/perl/webmn].

Beacons: A beacon is a feature or detail that is visible in a program or documentation that serves as an indicator of the function of the particular operation or structure – [http://www.cise.ufl.edu/research/ParalletPatterns/PatternLanguage/Background/Glossary.htm].

Client: a client is a system that accesses a (remote) service on another computer by some kind of network. The term was first applied to devices that were not capable of running their own stand-alone programs, but could interact with remote computers via a network. These dumb terminals were clients of the time sharing mainframe computer – [http://en.wikipedia.org/wiki/client_(Band)].

COTS: Commercial Off-the-shelf (COTS) is a term for systems which are manufactured commercially, and then may be tailored for specific for specific uses. This is most often used in military, computer, and robotic systems. COTS systems are in contrast to systems that are produced entirely and uniquely for the specific application – [http://en.wikipedia.org/wiki/COTS].

CSS: Cascading Style Sheets is a style sheet language that allows authors and users to attach style (e.g., fonts, spacing, and aural cues) to structured documents (e.g., HTML documents and XML applications). By separating the presentation style of documents from the content of documents, CSS simplifies Web authoring and site maintenance – [http://www.perfectxml.com/glossary.asp].

Delocalised plans: Delocalised plans are pieces of code that are conceptually related that are physically located in non contiguous parts of a program – [http://www.cc.gatech.edu/reverse/glossary.html].

Domain: A problem area. Typically, many applications programs exist to solve the program in a single domain. The following prerequisites indicate the presence of a domain; the existence of comprehensive relationships among objects in the domain, a community interested in solutions to the problem in the domain, recognition that software solutions are appropriate to the problems in the domain, and a store of knowledge or collected wisdom to address the problems in the domain. Once recognised, a domain can be characterised by its vocabulary, common assumptions, architectural approach and literature – [http://www.cc.gatech.edu/reverse/glossary.html].

Domain Model: The domain model should serve as a unified, definitive source of reference when ambiguities arise in the analysis of problems or later during the implementation of reusable components, a repository of the shared knowledge for teaching and communication, and a specification to the implementer of reusable components. A model of a domain should include information on at least three aspects of a problem: concepts to enable the specification of systems in the domain; plans to describe how to map to the specification into code; and rationales for the specification concepts, their relations, and their relations to implementation plans – [http://www.cc.gatech.edu/reverse/glossary.html].

DTD: The purpose of a DTD is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements. A DTD can be declared inline in your XML document, or as an external reference. - Jan Egil Refsnes [http://www.xmlfiles.com/dtd/dtd_intro.asp].

Embedded System: An embedded system is a special-purpose computer system, which is completely encapsulated by the device it controls. An embedded system has specific requirements and performs pre-defined tasks, unlike a general-purpose personal computer – [http://en.wikipedia.org/wiki/Embedded_system].

Faceted Classification: A component can be classified among several dimensions (facets). A facet is a multi-valued attribute where each facet can be represented by a set of terms with any kind of structure [http://www.cc.gatech.edu/reverse/glossary.html].

Formal Methods: A formal method is some advocate applying rigorous mathematical analysis to compute programming especially proof of correctness. They believe that traditional engineering is carried out with mathematical rigor, while programming is an iterative, trial-and-error process. These advocates strive to make programming more rigorous – [http://en.wikipedia.org/wiki/Formal_methods].

Framework: A reusable design for a class of software that is high level and highly structured, and makes extensive use of design patterns. It inverts usual control, so you can insert into low level parts; however, it is very difficult to implement well [http://www.cc.gatech.edu/morale/local/morph_glossary.html].

Legacy System: A legacy system is an application that has been developed and maintained over a period of time; typically its original designers and implementers are no longer available to perform the system's maintenance. Often specifications and documentation for legacy systems are outdated, so the only definitive source of information about the system is the code itself – [http://www.cc.gatech.edu/morale/local/morph_glossary.html].

Metric: A metric is something that can be measured. Metrics are used to better define what is meant by more abstract or general statements. For example, the program outcomes are the metrics of the program objectives since the outcomes better define what is intended by the objective and are measurable - [http://ceaspub.eas.asu.edu/MAE-EC2000/glossary.htm].

Noun: The part of speech that is used to name a person, place, thing, quality, or action and can function as the subject or object of a verb, the object of a preposition, or an appositive – [http://www.dictionary.com].

Program Comprehension: Program comprehension is the process of acquiring knowledge about a computer program – [http://www.cc.gatech.edu/reverse/glossary.html].

Program Plan: A description or representation of a computational structure that the designers have proposed as a way of achieving some purpose or goal in a program – [http://www.cc.gatech.edu/reverse/glossary.html].

Reverse Engineering: Reverse Engineering is the process of analysing a subject system to identify the system's components and their interrelationships, and create representations of the system in another form or at a higher level of abstraction – [http://www.cc.gatech.edu/reverse/glossary.html].

Scalability: How well a solution to some problem will work when the size of the problem increases – [www.dictionary.com].

UML: In software engineering, Unified Modelling Language (UML) is a non-proprietary, third generation modelling and specification language. However, the use of UML is not restricted to software modelling. It can be used for modelling hardware (engineering systems) and is commonly used for business process models and organisation structure modelling – [http://wikipedia.org/wiki/Uml].

Use case: A complete sequence of related actions initiated by an actor; it represents a specific way to use the system – [www.cbu.edu/~lschmitt/BSI/glossary.htm].

Verb: The part of speech that expresses existence, action, or occurrence in most languages – [www.dictionary.com].

XML Schema: XML Schemas express shared vocabularies and allow machines to carry out rules made by people. They provide a means for defining the structure, content and semantics of XML documents – [http://www.w3.org/XML/Schema].

XSL Transformations: XSLT is an XML-based language used for the transformation of XML documents. The original document is not changed; rather, a new XML document is created based on the content of an existing document. The new document may be serialized (output) by the processor in standard XML syntax or in another format, such as HTML or plain text. XSLT is most often used to convert data between different XML schemas or to convert XML data into web pages or PDF documents – [http://en.wikipedia.org/wiki/XSLT].