An investigation into evolving support for component reuse

Janet Lavery

University of Durham

Department of Computer Science

M.Sc. Thesis

An Investigation into Evolving Support for Component Reuse

Janet Lavery

1999

Abstract

It is common in engineering disciplines for new product development to be based on a concept of reuse, i.e. based on a foundation of knowledge and pre-existing components familiar to the discipline's community. In Software Engineering, this concept is known as software reuse.

Software reuse is considered essential if higher quality software and reduced development effort are to be achieved. A crucial part of any engineering development is access to tools that aid development. In software engineering this means having software support tools with which to construct software including tools to support effective software reuse.

The evolutionary nature of software means that the foundation of knowledge and components on which new products can be developed must reflect the changes occurring in both the software engineering discipline and the domain in which the software is to function. Therefore, effective support tools, including those used in software reuse, must evolve to reflect changes in both software engineering and the varying domains that use software.

This thesis contains a survey of the current understanding of software reuse. Software reuse is defined as the use of knowledge and work components of software that already exist in the development of new software. The survey reflects the belief that domain analysis and software tool support are essential in successful software reuse. The focus of the research is an investigation into the effects of a changing domain on the evolution of support for component-based reuse and domain analysis, and on the application of software reuse support methods and tools to another engineering discipline, namely roll design. To broaden understanding of a changing domain on the evolution of support for software reuse and domain analysis, a prototype for a reuse support environment has been developed for roll designers in the steel industry.

Acknowledgement

I would like to thank my friends in the Department of Computer Science for their much-appreciated encouragement. I would especially like to thank my supervisors, Dr. Cornelia Boldyreff and Dr. Steven Bradley, for their guidance and support throughout the production of this thesis.

I would also like to thank British Steel who funded this research through the REMAIN project and who allowed me access to their staff, especially Brian Kendall who spent hours explaining the roll design process.

Many thanks to the members of the CARD and REMAIN projects for allowing me to use them as a sounding board for my ideas.

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 the work presented here is the sole work of the author and no one else.

Contents

List of Figures

Chapter 1 Introduction

1.1 Objectives of the Chapter

This chapter provides the introduction to the thesis. Section 1.2 provides an introduction to the general research area, Software Reuse. Section 1.3 provides an introduction to the research area specific to this thesis, evolving support for component reuse and the criteria on which the success of this research will be based. Section 1.4 provides an outline to the remaining chapters of the thesis. Section 1.5 provides the summary for this chapter.

1.2 Introduction

In many engineering disciplines there is a wide selection of tried and tested components common to the discipline with which engineers develop new products [SOM96]. This allows engineers to build most of any new product from existing components leaving only a small number of components unique to the product to be originally developed. However in software engineering, new products are traditionally developed from completely original components. Over the last decade the research into software reuse, the use of the knowledge and work components of software products that already exist in the development of new software products, has begun to gradually infiltrate the development of new software products [ZAN97]. Software reuse is thought to hold great potential for raising the level of quality of software products, known as software applications, while reducing the overall development time [ZAN97]. To achieve this it is necessary to provide a reuse support environment. [PRE97]

Similarly, little explicit design reuse is found among engineers in the steel industry. However, there is a growing recognition in the engineers of the roll design community that design reuse can improve design practice and contribute to improved product development.

Since, software applications are comprised of approximately 65% domain specific software, understanding a software application's domain through domain

analysis is essential for successful reuse [BIG98]. Steel products also exhibit a wide variation across their areas of application necessitating domain analysis necessary here as well. Domain analysis is a complex process that begins with the location of domain knowledge sources and ends with an extensive domain model, including a definition of a domain language or domain terminology [PRI91].

The necessity of understanding a domain's terminology was recently made apparent with the much-publicised crash of the Mars Climate Orbiter [DOW99]. Critical measurements sent to the Mars Climate Orbiter when it was preparing for orbit around the planet Mars were mistakenly sent in imperial measurements and not the metric measurements the spacecraft was expecting resulting in the loss of a spacecraft worth 230 million dollars [DOW99].

1.3 Research Area and Criteria for Success

Most real-world domains are relatively stable; however, they are subject to change over time as Arango and Prieto-Diaz explain:

> *"Domains change because the real world changes, implementation technologies change, and our understanding of the problems and the solutions improves over time."* [ARA91]

Whatever the causes for change domains will evolve over time as will the terminology of the domain. If software reuse is to be effective in aiding developers to achieve high quality software and improve development times the software reuse support environment will have to evolve with the domain and reflect the most current understanding of the domain [ARA91].

This research will examine the proposal that a thesaurus developed as part of a reuse support environment to define domain terms and their relationships can evolve as knowledge of the domain expands through reuse. And that increased understanding of the domain will reveal more opportunities for reuse. In addition, this research will aim to demonstrate that specific software reuse techniques can be applied to support reuse in other engineering disciplines.

The proposal will be investigated in the following ways:

- An investigation into software reuse and domain analysis as it applies to software reuse.

- An investigation into software tool support for software reuse and domain analysis, which will support the evolution of the domain that must be reflected in software reuse. The focus will be on supporting the evolution of a component-based reuse library and the associated domain terminology.

- Development of a prototype of a reuse environment that will support component-based reuse and will include a thesaurus that will evolve as the domain understanding is increased. The prototype will be developed for the roll design community at British Steel.

The prototype will be applied to the domain problems associated with the need to reuse roll design documents and share domain knowledge within British Steel's roll design community. This constitutes a novel application of reuse support in roll design engineering.

1.4 Outline of the rest of the thesis

This section contains a brief outline of each of the eight remaining chapters of this thesis.

Chapter 2 contains the results of a literature survey on software reuse. This includes a detailed examination of component-based reuse and a less detailed examination of generative reuse. Domain analysis is considered necessary for successful reuse; therefore this chapter includes an examination of domain analysis as it relates to software reuse.

Chapter 3 contains the results of a literature survey exploring support for component-based reuse and domain analysis. It contains a detailed examination

of the component-based reuse library. This chapter also contains a detailed examination of software tools to support reuse and knowledge acquisition and sharing, focusing specifically on the use of a thesaurus.

Chapter 4 contains the results of the domain analysis performed on the domain chosen for this research. The domain is British Steel's roll design, where difficulties have arisen as a result of their plan to centralise their roll design environment. This chapter includes an analysis of the domain that puts the problem domain in a software-engineering context.

Chapter 5 contains the requirements for a software tool to support the British Steel roll design community when performing domain analysis and reuse of domain assets concurrently. This chapter contains an initial set of requirements based on a general understanding of the problem domain, a discussion of an initial prototype based on those requirements and the final requirements based on the evaluation of the initial prototype.

Chapter 6 contains the design based on the final requirements specification for ReST contained in Chapter 5. The design of the final prototype of ReST consists of dataflow diagrams used to identify the entities, processes and data that comprise ReST and entity-relationship diagrams used to demonstrate the relationships between the entities that comprise ReST.

Chapter 7 contains the implementation details of the design of the final prototype of ReST contained in Chapter 6. This chapter includes a description of the final implementation of ReST, and examples of the user interface and sample data.

Chapter 8 contains the results of the testing and evaluation of the implementation of the final prototype of ReST. This chapter also includes the status of the prototype prior to testing, a description of the testing and evaluation method, and the results of a scenario based evaluation.

Chapter 9 contains the conclusion formed as a result of the research and an examination of possible further work.

1.5 Summary

The general research areas of this thesis are software reuse and domain analysis as it pertains to software reuse in Software Engineering. The focus will be on an investigation into the effects of a changing domain on the evolution of support for component-based reuse and domain analysis and on the application of software reuse support to another engineering discipline. To demonstrate the results of the investigation, a prototype for a reuse support environment, Reuse Support Tool (ReST), will be developed. Specifically, the prototype will be use to demonstrate the reuse of design artefacts produced as part of the steel industry's roll design process. The prototype will include support for a component-based reuse library and a thesaurus that contains the associated domain terminology. The way in which the prototype allows reuse support to evolve over time and accommodate changes in terminology will be demonstrated using scenarios.

Chapter 2 Background

2.1 Objectives of the Chapter

The main objective of this chapter is to provide the results of a literature survey on software reuse within software engineering. The practice of software reuse is one of the means necessary to achieve the development of high quality software faster and with less effort. This chapter includes a detailed examination of component-based reuse and a less detailed examination of the more sophisticated generative reuse. As an understanding of a software application's domain through domain analysis is considered necessary for successful reuse, this chapter includes an examination of domain analysis as it pertains to software reuse.

Section 2.2 provides the overview of software reuse and contains an investigation into the subjects of domain analysis, component-based reuse and generative reuse. Section 2.3 provides the summary for this chapter.

2.2 Software Reuse

Within the context of this thesis, software reuse[1] is defined as the process of using assets, which includes both knowledge and work products from previously developed software applications[2] in the development of new software applications. Reusable assets can be developed in any phase of the software life cycle. Domain models, requirement specifications, designs, code, test cases, and user documents are just a few examples of assets that should be available for reuse [POU97]. The use of the term 'assets' is intended to imply that the knowledge and work products of existing software applications have a lasting value. Additionally, it draws attention to the fundamental concept of reuse, that an asset is a resource to be used repeatedly, and not an item restricted to a single use [REI97]. Assets are examined in more detail in Section 2.2.3.1.

Reuse is intended as a method for significantly improving software quality and software engineering productivity [HAL91]. Sommerville [SOM96] identifies

[1] Referred to as reuse for the remainder of this thesis.

several areas of software development that will improve with the practice of reuse. These are listed below:

- Application reliability;
- Testing;
- Consistency;
- Productivity;
- Development time; and
- Cost estimates.

Sommerville [SOM96] states that reuse of software assets improves software application reliability. Assets used in the development of software applications, which have been in operation for some time, are assets that have been shown to be reliable and thoroughly tested in real world conditions. Therefore, reusing these previously tested assets in new software applications will increase the new software application's reliability and reduce the time needed for testing. Consistency across multiple software applications can be achieved by embedding standards in reusable assets, thereby enforcing the use of the standards. Reuse can aid with improvements to software engineering productivity by reducing the time to market for new software applications, by reducing the time needed to develop the software application. In addition, when software engineers develop a single asset for use in multiple software applications instead of developing assets individually for each software application the development time of new software applications is reduced. Improvements to the accuracy of estimating the cost of an applications development can be achieved when reusable assets are known to exist and have been reused in previous software application developments.

According to Bassett [BAS97] the reusability of an asset is based on the three factors listed below:

- Usability or fitness of purpose;
- Generality or scope of applicability; and
- Adaptability or ease-of-use.

[2] Also referred to as application or applications within this thesis.

However, as Brooks [BRO95] points out software applications will only be developed using reusable assets when the reuse of assets requires less effort than the development of new assets. Brooks [BRO95] uses mathematical libraries as an area where reuse of existing assets is more economic than development of new assets. Mathematics is a well-understood domain with a standardised terminology with which to discuss problems and design solutions. Replication of the effort required to build both the domain understanding and the standard terminology would be both expensive and a waste of time.

Poulin [POU97] breaks reuse down into two distinct classes, horizontal reuse and vertical reuse. Horizontal reuse is the reuse of assets that are common to a wide spectrum of problem areas, known as domains, such as graphical user interface software or mathematical libraries. Vertical reuse is the reuse of domain specific assets where the assets are constrained in some way be the domain. When discussing reuse both horizontal and vertical reuses are considered as one. But Poulin describes a typical software application as comprised of approximately:

- 15% application specific software;
- 20% domain independent software; and
- 65% of domain specific software.

Although little distinction is made between horizontal and vertical reuse, it is safe to assume that efforts in research and industry are concentrated in the area where more substantial gains are to be made, which is vertical reuse [POU97]. The 'driver' behind any reuse is the domain and domain analysis is essential for successful reuse [BIG98].

2.2.1 Domain Analysis

As software applications are comprised of approximately 65% of domain specific software, the understanding of the software application domain through domain analysis is essential for successful reuse [BIG98]. Prieto-Diaz [PRI91] describes domain analysis as a complex process that begins with the location of domain knowledge sources and the defining of the domain boundary. An application may in fact have more than one domain boundary, in which case the domain

boundaries and the points of interaction between the boundaries must be defined. Once the knowledge sources and the domain boundary have been defined, domain analysis methods are applied to provide a domain model [ARA91].

Domain analysis is a highly skilled and difficult activity, which can require a good deal of time and effort to be completed successfully. Arango and Prieto-Diaz [ARA91] believe that prior to analysis activities, a considered decision should be made as to whether or not the domain is stable enough to justify the effort required. This is necessary not only to create a useful domain model but also to maintain the model as the domain evolves. In addition, there should be a problem (or problems) that require a software solution (or solutions) within the domain, known as the problem domain.

Consideration must also be given to the user community that would benefit from the production of a domain model. To justify the effort required to develop a domain model, the user community must have a substantial interest in having the domain modelled. The user community must require software solutions to identified domain problems. Figure 2.1 provides an overview model of the domain analysis process. The model includes the various knowledge sources needed to perform domain analysis and the variety of work products that comprise the domain model. This model does not contain a specific domain analysis method. A detailed discussion of domain analysis methods is beyond the scope of this thesis.

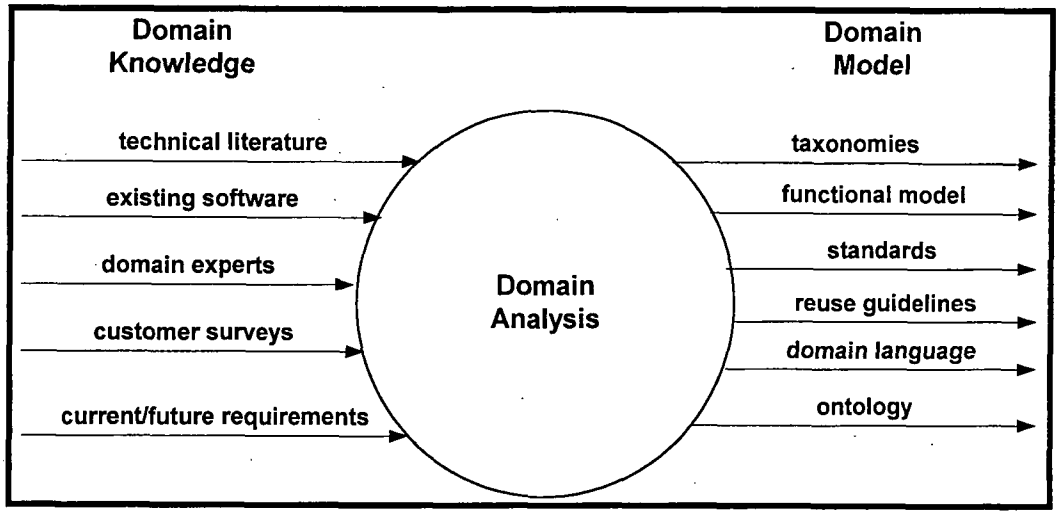

Figure 2.1 Domain Analysis Process

Though knowledge sources can vary from one domain to the next, there are several knowledge sources common to most domains including technical literature, existing applications, domain experts, customer information, and current and future requirements [ARA91, PRE97]. Arango and Prieto-Diaz [ARA91] state that domain analysis methods are usually based on a combination of both knowledge and software engineering methods. Pressman [PRE97] asserts that domain analysis must include the identification and classification of the items found within the domain. In addition, a representative sample of items found in the domain needs to be collected. This representative sample is then analysed in context of the domain model to ensure the model's accuracy. The end result of domain analysis is the domain model. The domain model consists of a variety of representations of the domain including taxonomies, standards, domain languages, and functional models [ARA91, PRE97]. In addition, the domain model should include a set of reuse guidelines to aid with the identification of existing reusable assets and the development of reusable assets [PRE97]. Included with the reuse guidelines should be examples of how domain assets could be used in the development of new software applications [PRE97]. Taken as a whole, the domain model is used to illustrate the generic objects and their operations, and the static and dynamic structures [REI97] of an application's domain. Prieto-Diaz [PRI91] states that the domain model provides the foundation on which all applications within a domain can be built. An important part of this foundation is the defining of a domain specific language or the

domain terminology. This definition of the domain terminology should include not only the meaning of the terms within the domain, but also the context in which the terms are used within the domain.

2.2.2 Domain Terminology

An important part of any domain analysis process is the defining of a domain specific language. The domain specific language provides the terminology that is used to model the domain [PRI91, PRE97]. As Figure 2.1 shows the domain model includes a taxonomy that is used to identify the classification of objects in the domain and a domain language that provides the terminology used within the domain. Also shown in Figure 2.1 as a part of the domain model is an ontology, which is in many respects a combining of the taxonomy and the domain language. In an ontology the objects of a domain and the relationships between them are identified, classified, and defined [CHA99].

2.2.2.1 Ontology

Within the Artificial Intelligence community the development of knowledge-based applications requires an in-depth and detailed understanding of the application's domain. Increasingly the foundation of the domain analysis is the construction of an ontology of the specific domain [SWA99]. Chandrasekaran, Josephson and Benjamins [CHA99] define an ontology as the means to classify the objects of a domain. Within the context of the domain, objects are identified, sorted, and defined, as are the relationships between the objects. In other words an ontology is the domain terminology used to represent the collection of domain specific terms and the concepts those terms represent. The domain terminology may be written in a knowledge representation language. Ontology contains a core layer of terms that are specific to the domain and an outer layer of terms that are more general or domain independent. Ontologies are generally classified in a tree structure from very general domain independent terms down to specific domain terms. Ontologies support knowledge acquisition, sharing and reuse by providing repositories for the general and detailed knowledge about specific domains [SWA99, VAL99]. Swartout and Tate [SWA99] believe that libraries of

ontologies from a wide variety of domains would aid in developing knowledge-based applications.

As part of the development of the Joint Forces Air Component Commander or JFACC[3] air campaign planning ontology, Valente et.al. [VAL99] investigated the reuse of existing ontology in the development of a new ontology. Two instances of reuse were examined: the inclusion of a publicly available ontology on time theory, and the merging and inclusion of two ontologies within the aircraft domain. In the first instance, a publicly available ontology on time theory, which uses the Ontoligua[4] knowledge representation language, was translated into Loom, the JFACC knowledge representation language, prior to inclusion in JFACC. This translation consisted of mapping the structure of Ontoligua on to Loom. A direct automated mapping between the knowledge representation languages was not possible. Some manual adjustments were required as the more general concepts represented at the upper or outer level of the ontologies have different structures. These differences highlight the bias constructed into each ontology. The bias is the result of the original ontology developer's view of the domain and the intended use of the application for which the ontology was originally created. In the second instance, two ontologies developed for the same general domain, aircraft, were merged together for inclusion in JFACC. Both ontologies used the same knowledge representation language as JFACC, but were constructed from different perspectives of the aircraft domain. As both ontologies were constructed for the same general domain, there was some overlap between the ontologies that was removed. There were also many differences which when combined created a richer and more widely useful ontology. In addition, the different perspectives of the domain meant that different views of the domain could be incorporated into JFACC, allowing users of the ontology to select the domain view most appropriate to the use of the ontology. Valente et. al. found that the reuse of ontology in the development of new ontology supported the usefulness and quality of the new ontology but translation from one knowledge representation language to another is difficult. Chandrasekaran, Josephson and Benjamins [CHA99] propose that domain ontology could be used in information

[3] Developed at USC Information Sciences Institute
[4] Development details at http://www-ksl.standford.edu/

retrieval applications to provide an organisational structure of the information and as a means to guide the search process. In this research, this proposal will be tested with the construction of a thesaurus.

2.2.2.2 Thesaurus

Like an ontology, a thesaurus is a collection of terms used to represent concepts within a specific domain and organised so that predefined relationships between the terms are made explicit [ISO2788, RAD90]. A thesaurus can be used to store and define a domain's terminology. Unlike an ontology that is constructed as an end product of the domain analysis process, a thesaurus can be developed over time outside the domain process. For example, as understanding of the domain increases when developers perform reuse [JAR95]. Thesauri are discussed in more detail in Chapter 3, Section 3.5.

2.2.3 Component-based Reuse

Reuse generally divides into two broad categories: component-based reuse, discussed in this section, and generative reuse, discussed in Section 2.2.4. In component-based reuse a reusable software component or a combinations of reusable software components[5] are placed into a developing software application with some or no modification. Figure 2.2 shows a model of component-based reuse.

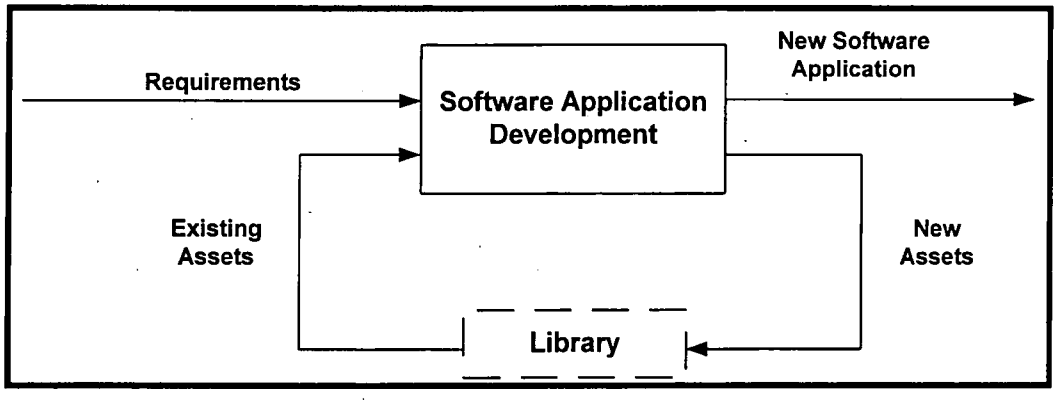

Figure 2.2 Component-based Reuse

[5] Referred to as asset or assets for the remainder of this thesis.

Kruzela [KRU93] describes component-based reuse process as follows. Existing software applications are examined and assets thought to be useful for reuse are identified and extracted. The assets are then modified to make them reusable assets and then stored in a repository or library. Once the requirements for a new software application are known, software engineers performing reuse, known as reusers, will search the library to find those reusable assets to be used in the new application. The reusable assets are then usually adapted and included in the development of the new application. Sommerville [SOM96] states that there are three conditions that must be met for successful reuse.

- There must be a well stocked library containing reusable assets that can be easily located, i.e. the assets are classified or catalogued in a manner that is conducive to search and retrieval.
- The reusers must have confidence in the reusable assets, i.e. the assets will behave as specified and be reliable. Sommerville suggests the use of a quality standard. Assets would then have to comply with the quality standard prior to inclusion in the library.
- There should be documentation accompanying each asset that will support reusers understanding of the asset. It is suggested that the documentation include examples of previous reuse of the asset including, a description of any areas that needed modification and any problems encountered in reuse or modification.

Assets includes both knowledge and work products generated as a result of software development. Domain models, requirement specifications, designs, code, test cases, end-user documentation and change reports are just a few examples of assets that should be available for reuse [POU97]. The use of the term 'assets' is intended to imply that the knowledge and work products of a software applications have a lasting value and that an asset is a resource to be used repeatedly, and not an item restricted to a single use [REI97].

2.2.3.1 Assets

In theory, any asset developed as part of an application's development and maintenance process is a reusable asset [POU97]. A non-exhaustive list of assets

may include: ontologies, project plans, cost estimates, requirements specifications, designs, source and executable code, test sets, user documentation and change reports. However, not all assets are directly suitable for reuse [KRU93]. Bassett [BAS97] states that reusable assets need to be usable, general, and adaptable; his position is summarised by the following tenets:

- Reusable assets must be judged as likely to satisfy some requirement in future application development.
- Reusable assets must be generalised sufficiently to be reusable in multiple future application developments.
- Adapting a reusable asset must be more cost effective than filling the needs of a new application development with a newly created asset.

In addition, reusable assets should be self-contained and be understood by the reusers [HAL91].

To summarise, a reusable asset must be generalised, adaptable, needed in future developments, self-contained and understandable. To illustrate this an example could be a single code module. What would be required to ensure a code module could be considered a reusable asset?

> Adherence to a reuse standard [PRE97] that included implementation guidelines such as naming conventions, code structure, header file information and module interfaces during the initial development would ensure that a code module is generalised. Modification of a code module so that it adhered to the reuse standard would also ensure that the code module was generalised.
>
> Adapting a code module usually means that some functionality needs to be added or removed prior to the use of the code module in the development of a new application. Adaptation of a code module is in many respects language dependent. For example, code developed using an object-oriented language achieves adaptation via inheritance, where the functionality contained in a base class is reused to provide the

foundation for the functionality contained in new objects (code modules)[SOM96].

As future requirements are one of the knowledge sources used during domain analysis, domain analysis should provide guidance as to which functionality is likely to be needed in multiple applications over time. A code module that fulfilled one or more future requirements would be reusable.

Code modules are self-contained, but would have to be accompanied by relevant documentation to be understandable [HAL91]. Examples of documents that are used to support reuser understanding of code modules are the module's requirement specifications, design documents and previous reuse history [PRE97].

2.2.3.2 Realising Reusable Assets

There are several ways to acquire reusable assets. Reusable assets can be developed specifically for reuse, or purchased from outside suppliers, or extracted from existing applications.

2.2.3.2.1 Developing Reusable Assets

Developing reusable assets is more costly than developing an asset that will used in only one application. Brooks [BRO95] declares that reusable assets are likely to be three times more expensive to develop than assets that are developed for a single application. Reusable assets 'cost' more to develop because more effort must be expended during development. Reusable assets need to be generalised, well documented, reliable, robust, and heavily tested. However, the cost of a reusable component can be amortised over more than one application [HAL91]. Wasmund [WAS95] found that the pressure placed on an application development team to get an application to market can undermine the effort and time that is required to develop assets for reuse.

2.2.3.2.2 COTS Commercial-Off-the-Shelf Software

Another means for acquiring reusable components is to purchase commercial off-the-shelf assets or COTS. COTS are tested software components intended to eliminate the need to design and develop the same software features each time they are needed in a new application development [KIE98]. COTS can be a broad spectrum of different types and sizes of reusable assets. Gentleman [GEN97] states that COTS can be an abstract data type, or subroutine, or a single class or even a class library. COTS can also be much larger and more generic such as databases or a domain specific information system. COTS can in fact be even broader; they can be problem-oriented languages for expressing problems and their solutions or even application generators. What is essential is that COTS already exist to be used in the development of new applications.

The benefits of using COTS are similar to those of reuse. The development and upgrading of COTS can be amortised over an entire customer base making the cost of purchasing COTS less than the cost of creating new assets for an application under development. Application development costs are also reduced because the expertise needed to develop the reusable asset is encapsulated in the COTS thereby eliminating the developing organisation's need to employ experts for every application developed. The time to market of the application under development will be reduced because the COTS have already been developed. COTS are usually of high quality, especially if they have been in use for some time. COTS used in industry will have been tested under operational conditions, and it is likely that errors will have been found and corrected. If the COTS user interface is familiar to the users of a new application, for example a net browser for a network centred application, then the time needed to train users is reduced.

There are, however, risks attached to using COTS. Buying in third party software has the risk of surrendering the application's future development to the vendor [MCP93]. Gentleman [GEN97] identifies several other problems with using COTS including the chance of the vendor going out of business leaving purchasers with no support. It is likely that reusers will be unfamiliar with the COTS and the time to develop new applications using COTS is lengthened by the time it takes the reusers to understand the COTS sufficiently to integrate them

into the application under development. This problem is compounded when the detailed specification of the COTS is incomplete or worse non-existent. It is unlikely that COTS will exactly match the requirements of the application being developed. Effort may have to be expended on supplementing the COTS functionality and masking unwanted functionality provided by the COTS. Additional time for testing and debug will be needed if the COTS are modified directly and not through interfaces or wrapping. It is not likely that COTS from different vendors will work together. A standard interface has yet to be adopted by the COTS industry, though at present Microsoft's Distributed Component Object Model (DCOM) and JaveBeans are contenders and the Object Management Group (OMG) is trying to make both interoperable with Common Object Request Broker Architecture) CORBA. Kiely [KIE98] points out that there needs to be a standardising of design notation to promote general understanding of COTS within the software engineering community. For instance, the Uniform Modelling Language[6] (UML) is gaining acceptance and adoption by the software engineering community.

Brooks [BRO95] believes the true potential of COTS lies in metaprogramming that would allow one or more applications to be reused as part of new development. For that to happen, a number of issues need to be addressed such as a metaprogramming interface language and a financially reliable market for COTS. Discussions need to be made on licensing agreements and how the COTS industry is to charge for COTS that are used over and over again in new developments and new versions of existing applications [KIE98].

2.2.3.2.3 Extracting Assets from Existing Applications

Acquiring reusable assets by extracting them from existing applications also requires a considerable amount of effort. Wasmund [WAS95] states that extracting reusable assets from an existing application is possible when the application's code is understandable and the applications' documentation is complete and consistent with the implementation of the application. However, he found that this is rarely the case. It is more usual for the source code to be the

[6] Developed by G.Booch, J.Rumbaugh, and I.Jacobson

only correct representation of the application. Therefore, considerable effort must be expended on re-establishing the application's documentation. If an implementation has not been developed for reuse, effort must be expended in modifying the code to make it suitable for reuse. Any modification of code could introduce errors into the code, which means that time and effort must be expended testing the assets once they have been modified for reuse. The situation worsens when the code is difficult to understand and the design rationale has been lost. Various tools and methods have been developed in an attempt to alleviate these difficulties and make it possible to extract reusable assets from existing applications. One example is the MIT project, Programmer's Apprentice that used standard program patterns or clichés as a means of identifying design strategies, thus enabling software engineers to abstract higher level descriptions of the software [HAL91]. This then provides the means to understand and document the application.

McParland [MCP93] cites domains such as stock control, or financial systems as examples of domains that have existed in a computerised format for many years. Existing applications within these stable domains provide a rich source of reusable assets from which to build templates for domain specific applications. Templates are generic specifications used to capture the common elements, data and functionality, of an application domain from relatively stable domains for reuse in new applications. Templates improve application quality, increase developer productivity and are adaptable as the domain evolves. When developers extract the commonalities from large numbers of applications within a domain, they can build and rigorously test the templates to ensure a high quality core foundation on which to develop a new application.

However, McParland [MCP93] cautions that even when using reverse engineering tools and domain analysis methods, template extraction is a difficult and time-consuming task. The effort that must be expended to develop the templates needs to be weighed against the benefits realised by using the templates.

2.2.3.3 Metrics for Reuse

Businesses need metrics to assist management in finding the economic reality of reuse within the limited time frame available in today's rapidly moving business environment. Businesses need to be able to judge whether it would be more cost effective to develop new assets for an application being developed or extract the reusable assets from existing applications. Sneed [SNE98] presents two metrics for measuring the cost of reusing existing applications in the development of new applications which are intended to provide management with realistic reuse cost estimates quickly. One metric measures the cost of converting existing code for reuse; the other measures the cost of wrapping existing code for reuse. Both metrics have been developed on the premise that the costs of reuse are relative to costs incurred in the development of a new application from scratch. The metrics are intended to provide a comparison of the cost of reuse with the cost of development from scratch. Additionally, the metrics can be used to provide the means to compare the two methods of reuse, conversion and wrapping.

Conversion, a form of re-engineering, of an existing application requires that an existing application be reconstructed in a new language, with modern databases and interfaces. Measuring the cost of conversion requires mapping the relationships between old and new code i.e. old and new statements and data types. Initially, the existing application's code is reviewed and each statement and data type is sorted into one of four possible relationship categories. The relationships between the old code and the new code in level of difficulty are:

- One to one (1:1) where for example when one data field in a hierarchical database is mapped to one data field in a relational database;
- One to many (1:M) where for example when one statement in COBOL EXAMINE is mapped to a function in C;
- Many to one (M:1) where for example several lines of Assembler can be mapped to a single line of C; and
- Many to many (M:N) where for example a loop construct in Assembler that begins with a start-label and a conditional branch back to it is mapped to a higher level language loop construct that begins with the conditional and ends with a loop terminator.

The total number of statements that fall into each relationship are tallied. Each relationship is weighted based on the level of expected difficulty in the conversion. The weights found in the formula are based on Sneed's own experience and research into conversion. He considers the simplest mapping to be 1:1 with the weight doubling with each level of difficulty. The convertibility of the existing statements is calculated as follows:

Convertibility (nw) = (1:1)1 + (1:M)2 + (M:1)4 + (M:N)8
/ Sum (Instructions + Data Types)

Once convertibility is calculated, the convertibility ratio is calculated and used to determine whether or not to convert the existing code for reuse. The ratio is:

The total number of statements and data structures (n) divided by the convertibility (nw) or **n/nw**.

The ratio is used to determine whether or not conversion of an existing application is cost effective, according to the following guidelines:

- A ratio greater than 0.75 means that reuse is considered approximately 33% or less than the cost of development and therefore reuse by conversion is recommended.
- A ratio between 0.75 and 0.50 means that the cost of reuse at best is about 50% of the cost of development and could in fact come close to the cost of development and therefore reuse by conversion is not recommended.
- A ratio of 0.50 or less indicates that reuse costs will approach, if not exceed the cost of development and applying reuse by conversion is not recommended.

Wrapping or reverse engineering of existing code for reuse requires that the existing code is sliced into reusable components and the components wrapped using application program interface, or in the case of a database, a data access layer. In wrapping, the existing code remains in its original language and current environment. Initially, existing code is sliced into potentially reusable modules or components with input, output and predicates documented. This process results in

a listing of callable functions. The listing is then compared with the requirements of the new application to discover to which of the four weighted categories the function belongs. The categories are weighted according to the amount of effort required to make the code reusable. The categories with weights are as follows:

- Reusable without modification, weighting of 1;
- Reusable with minor modification, weighting of 0.75;
- Reusable with major changes required, weighting of 0.50; and
- Functions that do not fit the new application, weighting of 0 (zero).

The wrappability of the existing code is calculated as the number of reusable operations and data divided by the sum of all the existing operations and data. The ratio of reusability is computed as:

> The sum of all weighted functions divided by the sum of all callable program slices (functions) or **fw/f**
> where
> fw = f1(1.0) + f2(0.75) + f3(0.5) + f4(0) and
> f = all callable program slices.

A resulting ratio of less than 0.5 indicates that wrapping existing code for reuse would require too many changes to the existing code and that it would be less costly to develop the new application without this method of reuse.

Sneed [SNE98] recommends that reuse using conversion or wrapping or some combination of both should only be considered when the cost of reusing existing code is less than 33% of the estimated cost of developing a new application without reuse.

2.2.4 Generative Reuse

Though component-based reuse is successful in areas such as mathematical libraries, problems arising during the understanding, location and modifying of components in other domains can be overcome with the change to the more productive generative reuse [JAR95, THI97]. McParland [MCP93] considers

generative reuse with the use of templates and automated code generation encapsulated in a CASE (Computer Aided Software Engineering) tool as a means to increase engineering productivity substantially.

Generative reuse is a process in which the knowledge about a domain and software engineering is reused to develop new applications. A new application is defined or specified using a domain specific language, the specification is then used as input for a code generator which transforms the input to output in the form of code for the new application [BIG98, SOM96]. Figure 2.3 illustrates a simple model of generative reuse based on Sommerville's [SOM96] model of generative reuse.

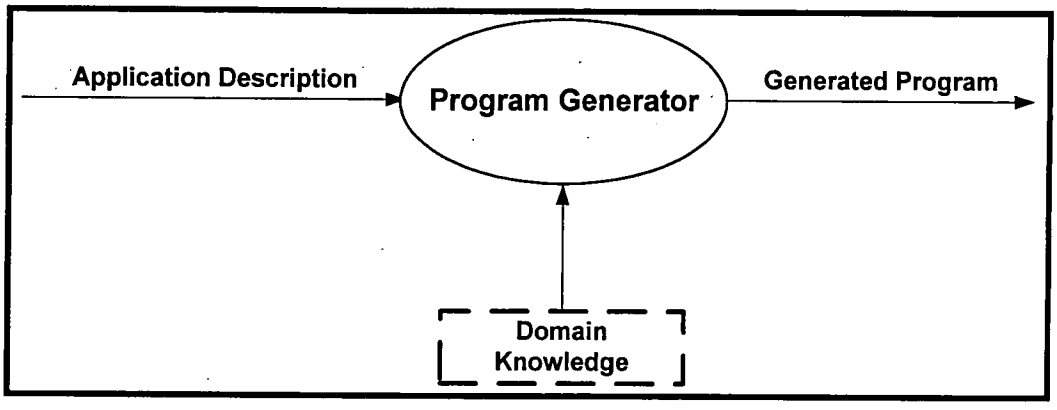

Figure 2.3 A Simple Model of Generative Reuse

For generative reuse to be practised successfully the domain must be very well understood and defined [SOM96]. In generative reuse systems such as Draco[7], several domains must be modelled and the progression to an implementation requires multiple transformations [BIG98, NEI89]. Brooks [BRO95] believes that generative reuse is only possible in areas where the domain is well understood and where both the problem and the various solutions are understood. In addition, a common language understood by all members of the domain community must be defined. This common language must be able to communicate both the domain problems and solutions to all members of the domain community. As well as a common language, the domain itself must be well disposed to generative reuse. The domain should require few parameters; have a variety of understood

solutions; and have been analysed sufficiently to provide the rules for selecting the solution once the parameters are known. Though domains that share these properties are considered to be exceptions rather than the rule, Brooks [BRO95] cites two domains, programs for sorting and systems for integrating differential equations, where generative reuse has routinely and effectively been used.

Sommerville [SOM96] cites application generators and parser generators as examples of successful generative reuse. Application generators are used to develop business data processing systems. Data processing for business is a well-understood domain, the tools used to specify the application such as a 4GL have been in use for some time. Parser generators are used in language processing. A parser generator takes as input the rules of a language or grammar and outputs a parser for the language. As with data processing, language processing is also a domain that is well understood.

Biggerstaff [BIG98] describes Draco, a generative reuse tool (system), which has been used successfully in the domain of telecommunications. Hall and Boldyreff [HAL91] state that Draco is intended to reuse analysis and design information. The approach taken is to define the problem domain in terms of objects and their operations and then to match them with objects and operations in terms of domains already analysed and known to Draco. When a match occurs, the software engineers interact with Draco to refine the known designs to develop the implementation of the application in the new domain.

In Draco the problem area for which a software solution is being sought, known within Draco as the world, is divided into one or more relevant domains [BIG98, NEI89]. Each domain contains a substantive collection of domain specific knowledge defined in a domain specific modelling language. Biggerstaff [BIG98] states that there have been approximately twenty domains modelled, including data structures, databases, SQL, and various network sub-domains. A reuser writes a specification for the new application in one of the domain languages. This specification is then refined (transformed) into one or more other domain

[7] Developed by J.M.Neighors

languages providing models of the next level of domains. The models are then refined further until they are brought together for transformation into the executable code. The refinements do not follow a linear path. The refinements, like the relationships between the domains, are graph like and can include recursion between domains and recursion within a domain such as a data structure being refined to simpler abstractions. Draco can also generate the documentation and the diagnostic or simulation tools for the target.

The emphasis in generative reuse is modelling the common elements of the domain. Jarzabek [JAR95] argues that generative reuse is the process of identifying and modelling both the commonalities and the variants of the application's domain. In this approach, the variants between existing systems are modelled and included in the domain model. One domain where generative reuse has been successful is programming language systems. Within the programming language systems domain, commonalties are modelled using finite state automata and parsing algorithms, whereas variants can be modelled using regular expressions notation and BNF. The problem is that to achieve a significant and accurate understanding of a domain, substantial effort and time is needed for domain analysis. Investigation into successful generative reuse domains, to aid with techniques needed for modelling variants, is limited by the narrowness of the domains in which generative reuse has been successful.

2.2.4.1 Generators

A generator is a form of translator that transforms expressions from one language to another, usually in the confines of a specific problem domain [BAS97]. Bassett [BAS97] uses a GUI generator as an example, where a visual representation language is used to develop screens which a generator then translates into C code. An area of concern with the use of generators is that the narrowness of the domain in which they works means that multiple generators are required to develop a useful application. Therefore there must be a mechanism in place, such as an interface language, to allow output from various generators to work together. Another area of concern is that when the generated output does not meet all the needs of the application under development, the output must then be

altered in some way. Bassett [BAS97] provides three strategies for coping with this concern:

- White box generation;
- Black box generation; and
- Grey box generation.

As expected, in white box generation, the developer is allowed to directly edit the generated output. The main drawback is the need to re-edit every time the output is re-generated. Black box generation provides the developer with specific 'exit points' where customising and editing may occur, for instance the addition of functionality. However, this means that some areas of the generated output, such as data structures, become sacrosanct, which can potentially place unnecessary restrictions on the new application. In grey box generation the generated output is a frame which uses parameter passing to handle variations.

In this context, Basset [BAS97] defines a frame as follows: "*A frame is a generic structure that can give rise to a variety of specific instances.*" Parameters are used to highlight the difference between a frame instance and its parent frame. The parameters contain default values that can be overwritten by other frames. Frames can be combined with other frames and contain frames. Bassett [BAS97] interprets the scepticism surrounding generators as the result of the incomprehensibility and rigidity of the generated output. Developers reject code generation because working around or correction to the generated output is more work than developing the source code. Bassett suggests that to overcome the problems associated with code generation, a bi-directional generator and frames be used to develop a new application. For example, a developer could use a graphical representation to form the input to a generator; the generator would generate a frame output in the graphical representation for further manipulation by the developer before translation into machine code for the computer. Henninger [HEN94] believes that the development of frames requires a very substantial effort in domain analysis and knowledge representation. However frames do provide conceptual closeness and can be used to effectively estimate modification effort.

Thibault and Consel [THI97] contend that industry has not adopted the use of generators because of the lack of tools to support the building of generators. They offer a two level 'framework' for the design of generators.

In the first level an abstract machine, which captures the fundamental operations that occur in the domain, is defined. The abstract machine contains the operations of the domain as they work on an explicitly defined state. This provides the means to reason about the operations and their interactions. The second level is to define a micro-language (or domain specific language) which will provide the interface to the abstract machine. Once both levels of the design 'framework' are complete; it is possible to build an interpreter for the micro-language, using the operations defined in the abstract state machine, and the abstract state machine itself. To generate the new application, the reuser must specify the application in the micro-language thereby producing a micro-program that will be mapped to operations in the implemented abstract machine using partial evaluation. Partial evaluation is a program transformation process that specialises a program based on the known values of some of the inputs. The operations in the abstract machine are then mapped to an optimised implementation.

2.2.5 From Component-based to Generative Reuse

Wasmund [WAS95] believes that component-based reuse cannot succeed in providing the improved quality and productivity needed by today's industries. The pressure on software engineers to get a product to market as quickly as possible is in direct conflict with additional time needed to develop reusable assets. The effort and cost associated with extracting reusable assets from existing applications to be reused in the development of a new application generally exceeds the effort and cost of developing the application from scratch. The use of COTS leaves a development company at the mercy of vendors for upgrades.

However, Reifer [REI97] proposes that the productivity and quality gains believed possible with the inclusion of reuse into the development of applications

can be achieved by the adoption of a reuse process maturity model. This model is similar to the Capability Maturity Model[8] (CMM) used to improve software development process. The Reuse Process Maturity Model suggested has five distinct levels presented below. They are as follows:

- Level one is ad-hoc reuse, and is practised randomly.
- Level two is project-wide reuse where reuse is practised within projects and assets are by-products of the project. At this level reuse is repeatable only on a project by project basis.
- Level three is organisation-wide reuse, where reuse is part of how an organisation does business. At this and all subsequent levels reuse is a repeatable process and reusable assets are products of the reuse process.
- Level four is product-line reuse where reuse is viewed as a business unto itself.
- Level five is broad-spectrum reuse where reuse is a significant part of the business culture and processes are optimised with reuse in mind.

Lim [LIM98] describes several other reuse maturity models, all of which are similar to the one presented above. All have an ad-hoc or chaotic level where reuse is not repeatable and end with a level where reuse is ingrained in the corporate culture.

Biggerstaff [BIG98] maintains that the 'driver' behind any reuse is the domain and that domain analysis is much more important than tools and methods for reuse, whereas Basset [BAS97] maintains that effective reusable assets are developed through their co-evolution with the systems that reuse them. Jarzabek [JAR95] asserts a combination of these two viewpoints and proposes that component-based reuse is the necessary start to understanding and modelling the domain sufficiently prior to the practice of generative reuse. As more component-based reuse is practised, the understanding of the domain is increased. With that increased understanding comes more opportunities for component-based reuse. Through the examination of the existing reusable components it should be

[8] Developed by the Software Engineering Institute for the U.S. Department of Defence

possible to identify the commonalities of the domain. The examination of the existing unique instantiations of the reused components is intended to locate the variants within the domain. Once commonalities and variants have been identified, suitable modelling language or languages need to be defined. The modelling language is then used as the basis for the application generator input and in the defining and development of the actual application generator.

2.3 Summary

This chapter contains the results of a literature survey on reuse within software engineering. It is maintained that successful reuse is dependent on a good understanding of the domain in which the assets are to be developed and reused [BIG98] and that a good understanding of the domain can be aided by the practice of component-based reuse [BAS97]. In addition, it is proposed that component-based reuse is the necessary start to understanding and modelling the domain sufficiently to enable the practice of the more gainful generative reuse [JAR95]. Ontologies are believed to be potentially useful in supporting knowledge acquisition and sharing, as well as an aid to effective component-based reuse by providing repositories for the general and detailed knowledge about specific domains [SWA99, VAL99]. In this research, this proposal will be tested with the construction of a thesaurus.

Chapter 3 Further Background

3.1 Objectives of the Chapter

The main objective of this chapter is to provide the results of a literature survey exploring support for component-based reuse and domain analysis. Specifically, it provides a detailed examination of the reuse library and the thesaurus, which have been proposed as tools to support reuse and knowledge acquisition and sharing.

Section 3.2 provides the component-based reuse library[9] overview including an examination of surrogates, library structures, the library size scaling issue, and covers the subject of search and retrieval of potentially reusable assets. Section 3.3 provides an examination of possible organisational approaches for libraries. Section 3.4 provides a comparison of the previously presented approaches to library organisation. Section 3.5 provides a detailed examination of thesauri including the identification and definition of the possible relationships between terms in a thesaurus; and an examination of the support that a thesaurus can provide for domain analysis and component-based reuse. In addition, Section 3.5 provides an examination of the issues surrounding the construction of a thesaurus. Section 3.6 provides the summary for this chapter.

3.2 Library

In component-based reuse, assets are predominantly natural language documents [MIL98]. Even code modules can be classified as natural language documents as they include header files and comments. In component-based reuse it is unusual for the actual assets themselves to be stored in the library. A. Mili et.al. [MIL98] state that assets are invariably large, detailed, and complex entities. This makes the tasks of search and retrieval of assets both time consuming and complicated. It is more usual for a library to contain a representation of each asset, which is known as a surrogate. The surrogates are more abstract, concise and lacking in unnecessary detail. Surrogates provide a summary of the asset in much the same

[9] Referred to as library for the remainder of the thesis.

way as an abstract provides a summary of a thesis. Surrogates reduce the effort and complexity of the search and retrieval of reusable assets and as an aside reduce the overall size of the library. Surrogates should promote understanding of the asset. If location and understanding do not happen then reuse cannot happen [FRA94]. Surrogates are based on some common language between those engineers that develop the surrogate and those reusers that query the library looking for reusable assets. Without a common language it will be difficult if not impossible for a reuser to locate and understand the assets contained in the library. Domain analysis can provide a common language. For instance a domain model consists of a variety of representations of the domain including the domain languages [ARA91, PRE97]. A more detailed possibility is an ontology, the domain terminology, which represents the collection of domain specific terms and the concepts those terms represent [CHA99]. Another possibility that is explored in this thesis is a thesaurus, which contains the collection of domain specific terms and the defined relationships between those terms. Section 3.5 of this chapter provides a detailed examination of thesauri.

3.2.1 Library Structures

There are a variety of possible structures for a component-based reuse library including flat file, hierarchical, database, multiple, and distributive [MIL98]. Frakes and Pole [FRA94] contend that the contents and structure of a library are only issues when the reuser community has insufficient knowledge about the library's complete structure and contents. Library issues are unimportant if staff turnover is low or the asset collection is small. In both cases, the reuser community would have sufficient knowledge of the assets to make finding and understanding them simple. However, Mili et.al. [MIL98] consider it important that the reusers and the maintainers of the library share some common knowledge of the structure of the library so that reusers can locate the reusable assets as they are required.

The library's structure is usually inhibited by the content of the surrogates it contains and the searching mechanisms used in the reuse process. If an exhaustive search method is applied then the structure of the library can be

arbitrary. However, if the searching mechanism is restricted to a library with a particular structure, say hierarchical, then the library's structure is restricted, and must be hierarchical. Obviously the library's maintainer must have an in-depth understanding of the library's structure.

3.2.1.1 Vertical Scaling

Brooks [BRO95] states that programs are comprised of conceptual chunks much larger than high-level language functions, modules or classes, and that by providing a reuse library of such conceptual chunks with variation via parameters, developers could construct new higher quality systems with less effort. Biggerstaff [BIG98] asserts that a significant factor in the success of reusing assets, which are code modules or components, is the size of the components available for reuse. The reuse of larger components reduces the amount of writing and debugging a developer must do. This in turn encourages the developer to use reusable components. A four year study of reuse at NTT software Laboratories (see reference ISO92 in [BIG98]) found that reusing larger modules achieved a higher rate of reuse. Though small modules, up to 50 lines of code, constituted 48% of the reusable components, their reuse ratio was only 6%; whereas with large modules, more than 1000 lines of code, constituted only 6% of the reusable components, their reuse ratio was 56%.

However, vertically scaling, increasing the component size, has the effect of narrowing the domain in which the component is reusable. Larger components are innately more domain specific, which in turn reduces the number of applications where it would be suitable to reuse the component. Since large components are more domain specific they can have more functionality than is required by a new application and be too large and complex to understand sufficiently to modify.

3.2.1.2 Horizontal Scaling

One way of combating the vertical scaling problem is to increase the number of variations of a module i.e. to provide customised versions of components held in the library. This is known as horizontal scaling [BIG98]. Ideally, a component

should be available for a large number of environments. However, as Biggerstaff [BIG98] states a single large code module reflects the consequences of many design decisions, any of which could have a number of different consequences. To provide a customised version of a large component for every possible consequence for every design decision would result in a combinatorial explosion of the number of components held in the library. The problems associated with large components could be reduced by the use of global standards, such as Win32 API, and by narrowing domains. However, global standards are not prevalent in the software industry and components that look as if they would be reusable across domains or in wider domains are restricted to use in very narrow domains, hindering efforts in reuse.

3.2.2 Search and Retrieval

Component-based reuse is only successful if the reusers can locate assets to be reused in the development of new software applications [FRA94]. Stated simply, assets are represented by surrogates that are stored in a library. A reuser queries the library in an attempt to locate reusable assets relevant to the development of new applications. The library is searched for assets that meet the criteria defined by the query. The search result, the subset of surrogates that meet the search criteria, is presented to the reuser. Using the subset of surrogates contained in the search result the reuser selects the candidate asset or assets for reuse.

The level of recall and precision achieved by the search traditionally measures the success of a search. Recall is a ratio of the number of relevant assets retrieved over the total number of relevant assets available in the library [MIL98]. Precision is a ratio of relevant assets retrieved over the total number of assets retrieved from the library as a result of the search [MIL98]. Difficulties arise when trying to define 'relevance' also known as the search goal [MIL98, HEN94]. The search goal can vary depending on the state of the development and the needs of the reuser. Henninger [HEN94] contends that a reuser's understanding of the problem domain and potential solutions start as an ill-defined need for information and increases with the development of the new application. Henninger maintains that an effective search and retrieval method

should be an aid to increasing the reuser's understanding of both the problem domain and the potential solution. CodeFinder was developed to test this view. It was designed to support incremental query construction and retrieval using spreading activation (which retrieves items related to the query). Henninger found that when the problem domain and potential solutions were ill-defined, the incremental query construction and spreading activation were useful, but as the problem and solution became clearer the users found the spreading activation problematic.

A searching mechanism must avoid search results that contain either false positives or false negatives. A false positive search result is an asset found in the search that matches the search criteria imposed by the search query but not an asset the reuser requires. A false negative search result is an asset that is required by the reuser that is not returned in the search result because it does not match the search criteria imposed by the search query. False positives generally occur when the search criterion is too broad and false negatives occur when the search criterion is too narrow. To be able to retrieve the assets required by the reuser, a query must be well articulated [HEN94]. Difficulties arise when a natural language is used to define surrogates and queries. Terminology mismatches can occur when surrogates are defined by one person and later searched for by someone else. Henninger [HEN94] claims that terminology mismatch is a major problem in search and retrieval. His studies have shown agreement on naming common objects occurs between 15 and 35 percent of the time. Even when up to 15 aliases are allowed for, agreement only rises to between 60 and 80 percent. Aid with query construction is necessary for effective reuse.

3.3 Library Organisations

There are a variety of approaches for library organisations that will support effective component-based reuse. Most support the reuse of natural language documents; however, some exploit the nature of code. Sections 3.3.1 to 3.3.5 contain a description of each of the five different approaches to library organisation listed below:

- Indexing

- Enumerated Classification
- Multi Faceted Classification
- Attribute-Value Classification
- Exploiting the Nature of Code

A comparison of these is presented in section 3.4.

3.3.1 Indexing

Traditionally an index is an alphabetised subject listing that is given at the end of the book. By means of an index a reader can search and locate the parts of the book relevant to the subject he/she needs to read about. When applied to reuse, indexing is a largely automated process of constructing surrogates for assets. Assets are indexed, providing a list of unique key words or phrases that are then associated with the asset and become the asset's surrogate. Kelledy and Smeaton [KEL97] state that within any asset there can be a number of terms (words or phrases) which do not contribute to defining the asset's surrogate or aid with discriminating between assets. Within a text document words such as 'the', 'it' and 'this' would appear frequently but would not contribute towards defining the surrogate or distinguishing the asset from others in the library. These frequently occurring words are included in a stoplist, which provides a listing of terms to be disregarded during indexing. It is possible for an asset to contain terms that are common to a large number of assets. These common terms if included in the surrogate would decrease the means of distinguishing between the assets. These common terms are candidates for addition to the stoplist; however, care must be taken to ensure that the term would not be relevant to reusers when searching the library before adding it to the stoplist.

Indexing can be performed with either an uncontrolled or controlled terminology or some combination of both. The premise behind an automated uncontrolled terminology is that terms extracted directly from the document with reasonable frequency are good indicators of the assets content [MIL97]. Frakes and Pole [FRA94] describe indexing, as a process that is highly automated and low cost as little effort is required to construct surrogates and populate the library. They go

on to say that uncontrolled terminology means that reusers can be very precise when constructing queries and improve search results by reducing the number of non-relevant assets retrieved (false positives). However, the terms used within the assets and then the surrogates must be known and understood by the reuser prior to the formation of the query, or the query may result in incomplete search results (false negatives) and in some cases incorrect search results.

With controlled indexing, the index terms are accumulated by domain experts who review the assets and build the associated surrogates of index terms. A controlled terminology places restrictions on the terms suitable for use in surrogates and searching. This restriction ensures that an engineer developing the surrogate and populating the library and the reuser searching the library are working with the same terms and can reduce search effort and promote reuse. However, manually developing a controlled terminology is a labour intensive activity and therefore costly to an organisation. More commonly there is a combination of the two where an automated process forms an uncontrolled terminology which is then edited by domain experts to form a controlled terminology suitable for using in surrogates.

In addition to term extraction, indexing can also provide a frequency count of the number of times a term appears in an asset. This frequency count can be used to select the relevant terms to be placed in the surrogate (if restriction required) or can be used later in searching the library. For example, CodeFinder [HEN94] uses inverse document frequency, which supports the argument that the frequency of an indexed term is significant when it occurs in a document less frequently. In this example, it is proposed that less frequent terms are a more precise representation of the contents of document. A precise representation of document contents reduces the number of false positives in a search result i.e. it reduces the number of documents that meet the search criteria but are in fact not relevant to the reusers needs.

An index can be comprised of terms that are either single words or single phrases. A phrase is a combination of one or more words that convey meaning. Kelledy and Smeaton [KEL97] found that when an index contains phrases the

search precision is higher than when an index is comprised of single words. Phrases by their definition are less ambiguous and increase a reuser's understanding of the asset in which they occur. Phrases occur less frequently in an asset, thereby providing a more precise representation of the contents of the asset.

In addition to using indexing to create surrogates for specific assets, indexing can also be used to create and maintain an inclusive index for all assets in the library. An inclusive index is used in domain analysis, surrogate definition, and asset retrieval. In domain analysis an inclusive index contributes to the development of the domain specific language. In surrogate definition an inclusive index is used to build an asset's index or as a tool in other asset classification schemes, such as faceted classification. An inclusive index of asset terms can be used in retrieval, to assist with the location of acceptable search terms.

3.3.2 Enumerated Classification

Frakes and Pole [FRA94] describe an enumerated classification scheme as a process in which the domain is described using controlled terms that are mutually exclusive and structured hierarchically, similar to a book's table of contents. These terms and their relationships within the hierarchy are then applied to the assets to form the surrogates. Figure 3.1 illustrates an example of an enumerated classification.

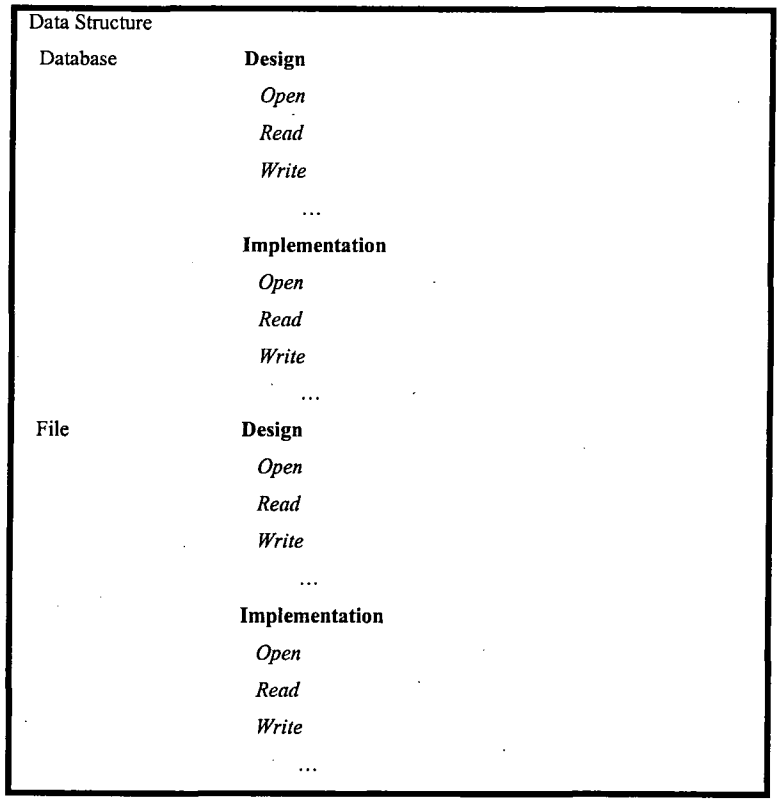

Figure 3.1 Enumerated Classification Example

The hierarchical structure helps to promote reuser understanding of the relationships between the terms, making the surrogates easy to define and understand. The hierarchical structure provides a logical searching structure, where a surrogate collection is searched using simple tree search algorithms. However, Frakes and Pole [FRA94] caution that much effort is required to perform the sufficient domain analysis to ensure that the terms and their relationships are correctly defined and that the hierarchy is complete. Any changes within the domain, either through increased understanding or evolution, may need to be reflected in changes to the hierarchy and require changes to some if not all of the surrogates contained in the library.

3.3.3 Multi Faceted Classification

H. Mili et.al. [MIL97] explain multi-faceted classification schemes as the process whereby the asset's surrogate is constructed from a set of common attributes (facets), and the values assigned to each facet. The facets are short textual

descriptions of the common attributes. There can be a large number of possible facets; however, surrogates are usually limited to a combination of between six and eight possible facets. The values assigned to each facet are selected from a controlled terminology. The terminology is organised in a hierarchical structure where the root terms are in fact the facets. Figure 3.2 illustrates a possible hierarchical structure for values to be used with the facets: Data Structure, Asset Type and Operation.

Asset Type	Data Structure	Operation
Design	Database	Open
Implementation	Table	Close
	Records	Read
	Files	Write
		Copy
		Delete

Figure 3.2 Multi Faceted Classification Example

The presence of several values provides alternative values, as opposed to partial values. The terminology is controlled which will limit the possible values in a facet, but this provides for the use of common terminology when defining and searching the surrogates. To ensure that the choice of values is not too rigid a hierarchical structure of the terminology can allow a choice of broader or narrower values as shown in Figure 2.4 where "Data Structure" is a broader value than "Database" and "Table" is a narrower value than Database. A. Mili et.al. [MIL98] contend that a thesaurus can provide additional richness to the terminology. A thesaurus can be developed to include additional relationships between terms such as preferred term, non-preferred term, synonyms or any relationship required by domain. A thesaurus can be used to help a reuser to define a search query by allowing a reuser to locate any synonyms for the facet value being considered for use in the search query [PRE97]. For example a synonym for 'delete' is 'remove'.

Prieto-Diaz [PRI89] states that the order of the facets in the surrogate can be used to further define the asset. The order of the facets can be used to convey the

intention of the reusable asset or to reflect the priorities of the reuser community. Domain analysis is essential to support multi-faceted classification. It is only through a thorough understanding of the domain that the facets, the predefined terminology (values), and the needs of the reusers, can be discovered. Frakes and Pole [FRA94] state that, unlike enumeration classification, multi-faceted classification schemes for surrogates adapt easily to changes in the domain. New facets can be defined and values can be added or reorganised without reordering part or all of the library.

3.3.4 Attributes-Value Classification

Pressman [PRE97] characterises attribute - value classification of assets as a process that is similar to multi-faceted classification in that extensive domain analysis is required. The domain analysis is used to find attributes with which to describe the assets and suitable values for each attribute to contain. However there are differences, in attribute - value classification, there are no limits to the number of attributes assigned to an asset, there is no structure, and a thesaurus is not used. An example stated by Frakes and Pole [FRA94] provides the kind of information available from the domain that can be used in attributes - value classification. Possible attributes may be parts of an asset such as its function, data type, language, and author. A possible value for each of these attributes could be {sort, queues, Pascal, F. Bloggs}.

3.3.5 Exploiting the Nature of Code

Historically most software engineers practice some form of reuse, i.e. they reuse code that they or respected colleagues have written. Software engineers are comfortable reusing code, especially code modules with which they are very familiar, module where they understand the functionality and side effects if any, and where they have confidence in the module's quality. Productivity studies have shown that good software engineers have good filing systems that allow them to quickly locate modules they consider fit for reuse [BAS97]. This form of reuse is commonly known as 'ad-hoc' reuse. Ad-hoc reuse can entail the reuse of unchanged modules: however, it is more common for ad-hoc reuse to require some changes to the reusable asset, i.e. modification, addition or deletion of

functionality. Basset [BAS97] refers to ad-hoc reuse as copy-and-modify where existing assets are copied then modified to work on the new application. Though effective to some extent, in reducing effort by reusing working and tested assets, a copy-and-modify reuse strategy has several inherent problems. When assets have not been designed for reuse, the effort to understand, modify and then reuse an asset can be greater than developing the asset from scratch. Typically software engineers modify assets one character at a time; a process that is as tedious as it is time consuming. Changes made to an original reusable asset that has been copied and modified for use in several applications must be manually made to some or all of the copies. The major reason ad-hoc or copy-and-modify reuse is not effective reuse is that it is difficult, if not impossible, to find the common elements in a group of similar assets and identify the elements that are unique to each application.

Though ad-hoc reuse is generally practised in the software engineering community, a more planned and managed reuse process is needed. In component-based reuse there are methods that exploit the traits inherent in code. A. Mili et.al. [MIL98] describe reuse methods which take advantage of the traits inherent in code, specifically the executable nature of code or the patterns found in source code. Though these methods restrict the assets to that of either executable or source code software engineers are likely to be familiar with the concept of reuse code.

Reusable executable code assets are located by matching sample input data with the desired output result. It is expected that by using a reasonable data sample that the new application will need to process, and by knowing the intended result of the processing it is possible to locate the correct executable code assets for reuse in the new application.

Code skeletons as described by Bassett [BAS97] are a mechanism for dealing with the variations required to make a reusable program part of a new application. The code skeletons contain the general or common reusable elements of a program. A developer uses an editor to 'flesh out' the particular instance of the program. Code skeletons are considered to be better than ad-hoc reuse as they

allow developers to show where the variation of each reuse has occurred. However, when changes must be reflected in each existing instance of a code skeleton, every existing reused code skeleton must be altered manually.

3.4 Comparing Library Organisations

Among the different approaches to the organisation of a component-based reuse library, indexing is considered the least expensive and easiest to develop, maintain, and use. Indexing using an uncontrolled terminology can be completely automated and requires no domain analysis [FRA94]. Conversely, structured classifications such as multi-faceted classification require extensive domain analysis prior to development and the intervention of domain experts during maintenance [HEN94]. To ensure they do not incorrectly limit the search to a few branches, reusers using a structured classification method need to understand both the structure of the library and the terminology used in the surrogates [HEN94].

Frakes and Pole [FRA94] conducted an empirical study to compare four different approaches to component-based reuse library organisation, including three structured classification methods. Specifically, they compared indexing, enumerated classification, multi-faceted classification and attribute-value methods. The study used Proteus, a reuse library system that supports multiple component-based reuse methods. The study measured the search effectiveness and search times of each method. Search effectiveness was measured using precision, recall and overlap. Overlap is a ratio of the number of relevant assets in the intersection of two methods divided by the number of relevant assets in their union. Also, test subjects were asked to rate their preference for each method and the method's assistance with understanding the reuse assets.

Frakes and Pole [FRA94] found that though there was no significant difference in the recall and precision measures between the four methods, all four methods being moderately effective in searching, each method found different assets for the same search queries. The average overlap for the methods ranged between 72 and 85 percent. Search times did vary significantly; there was an average

difference of 60 percent between the slowest (indexing) and the fastest (enumerated classification). Test subjects did not favour any particular method. Each method was rated as best and worst and no one method was consistently rated as satisfactory. It was found that there was no significant difference between methods for helping subjects understand assets. All four methods were judged to be only moderately helpful.

Frakes and Pole [FRA94] conclude that the asset collection in the library should be represented in as many ways as is possible. Using more than one method will increase a reusers chances of finding relevant assets and having a variety of methods ensures that reusers have access to the method they prefer. In addition, they concluded that none of the methods advance reusers' understanding of the assets and that techniques such as domain analysis are probably needed to support understanding.

H. Mili et.al. [MIL97] conducted a study comparing approaches to component-based reuse library organisation and found that searching surrogates consisting of the asset's index terms using an uncontrolled terminology performed better than searching surrogates comprised of multi-faceted classification using a controlled terminology. They hypothesise that there are two distinct searching stages, neither of which can be satisfied by a multi-facetted classification method. In the first stage the reuser does not have a clear idea of what is needed for the application under development. The reuser needs to explore the library to find potentially reusable assets. A multi-facetted classification method is too rigid to be useful in this stage. In the second stage the reuser has a very clear idea of what is needed for the application under development. The reuser needs to be able to find reusable assets that precisely fit those needs. A multi-facetted classification method does not provide the level of detail a reuser must have about each asset. This supports Henninger's [HEN94] theory that the relevance of the search results varies with the needs of the reuser. Reusers start with an ill-defined need to understand a new applications domain and progress to a specific need to develop the new application.

3.5 Thesaurus

There is a need for support of knowledge acquisition and sharing, as well as an aid to effective component-based reuse. It has been proposed that an Ontology would be useful by providing repositories for the general and detailed knowledge about specific domains [SWA99, VAL99]. In this research, this proposal will be tested with the construction of a thesaurus. Like an ontology, a thesaurus is a collection of terms used to represent concepts within a specific domain and organised so that predefined relationships between the terms are made explicit [ISO2788, RAD90]. Also like an ontology, a thesaurus can be used to promote reusers understanding of a domain. However, an ontology is an end product of extensive domain analysis, whereas a thesaurus can be developed as part of the reuse process.

3.5.1 Thesaurus Overview

A thesaurus is a collection of terms used to represent concepts within a specific domain and organised so that predefined relationships between the terms are made explicit [ISO2788, RAD90]. It can also be used to increase reusers understanding of a domain. Development of a thesaurus can be made part of the reuse process. The terms within a thesaurus and their relationships can be defined as knowledge of the domain expands through reuse. Increased understanding of the domain, brought about by developing and maintaining the thesaurus and the practice of reuse will reveal more opportunities for reuse [JAR95]. The Standard ISO[10] 2788 - 1986 (E) Documentation - Guidelines for the establishment and development of monolingual thesauri, ISO2788[11], (ISO 2788 standard) [ISO2788] defines terms as a word or collection of words used to define by way of their specific meaning within the domain, the domain concept they represent, and their relationship to other terms. For example a bank can be defined as an institution where money is deposited or lent etc. but a bank is also a more specific type (narrower term) of a financial institution. A thesaurus would provide not only the definition for the terms bank and financial institution but also their relationship with each other. Terms held in a thesaurus have a single domain specific definition attributed to them; i.e. a term represents a single domain

[10] International Organization for Standardization
[11] Prepared by Technical Committee ISO/TC 46, Documentation

concept. The ISO 2788 standard [ISO2788] defines three types of relationships between terms. These relationships are listed below:

- Equivalence
- Hierarchical
- Associative

These three relationships between terms are described in more detail in subsections 3.5.1.1 to 3.5.1.2.

3.5.1.1 Equivalence Relationships

The ISO 2788 standard [ISO2788] describes equivalence relationships as relationships that cover synonyms and quasi-synonyms. Synonyms are terms that have the same, or nearly the same, meaning. Quasi-synonyms are terms that when used in natural languages are considered different but when used within a domain are treated as synonyms. Within equivalence relationships terms are designated as either preferred terms or non-preferred terms. Preferred terms are the most likely term to represent the domain concept within the user community. The equivalence relationship is defined as either USE or USED FOR, as in non-preferred USE preferred, and preferred USED FOR non-preferred. As an example, 'software maintenance' USE 'software evolution' would indicate that 'software evolution' should be used instead of 'software maintenance'.

3.5.1.2 Hierarchical Relationships

The ISO 2788 standard [ISO2788] defines hierarchical relationships as superordination and subordination relationships. The more general or broader term is SUPERORDINATE to a more specific or narrower term and a narrower term is SUBORDINATE to a broader term. There are three types of hierarchical relationships: generic, hierarchical whole-part, and instance. Generic relationships are used to identify the link between a class and its members, where a broader term is a class and narrower term is a member of a class as in the class 'employee' and the member 'department manager'. Hierarchical whole-part relationships are for a limited range of relationships where the actual working of the narrower term implies the name of its broader term; as in Durham (narrower term), England (broader term). Instance relationships occur between general

terms, the classes, and individual instances of a term. For example, Roll Mill (class) and British Steel Roll Mill at Teesside (instance) illustrates an instance relationship. A domain concept that cannot be described by a more general domain concept is said to be a top term. A domain concept that cannot be narrowed is said to be a bottom term.

3.5.1.3 Associative Relationships

Aitchison and Gilchrist [AIT72] state that associative relationships are the relationships that exists between terms which are bound conceptually in the minds of the users within the community but cannot be defined hierarchically or equivalently. An associative relationship is defined as related terms, in that when applying one term, for example in a search query, a user would profit by being reminded of the existence of the related term. As an example consider the relationship between a discipline and its objects [ISO2788], such as Software Engineering and programs. The ISO 2788 standard [ISO2788] present ten possible associated relationships listed below:

1. Discipline and objects
2. Process and instrument
3. Action and product of action
4. Action and its patient
5. Concepts related to their properties
6. Concepts related to their origin
7. Concepts linked by causal dependency
8. A thing and its counter agent
9. A concept and its unit of measure
10. Syncategorematic phrases and their embedded noun (the embedded noun of 'model ship' is 'ship' [ISO2788])

3.5.1.4 An Example of Software Tool Support Using a Thesaurus

Practitioner, an academic and industry collaboration project funded by ESPRIT, identified both methods and software tool to improve the theoretical and technical aspects of reuse. As part of the project a thesaurus was developed to aid with domain terminology understanding and improve searching for reusable assets

[MIL94]. H. Mili et.al. [MIL94] describe Practitioner as a project where the emphasis of the project was on the reuse of assets developed early in a software application's development. The software tools delivered as part of this project were PRESSTO, PRESSTIGE, and MUCH (Multiple User Creating Hypertext). The tools were developed in a specific order, with lessons learned from experiments conducted on one tool influencing the development of a subsequent tool. The first tool developed was PRESSTO a quickly developed indexing and retrieval tool which enables developers to classify, store and retrieve reusable word-based documents. A matrix is built, where the rows are designated by the index terminology and the document identifiers designate columns. Asset retrieval is a matter of searching the matrix for a list of appropriate document identifiers. The index terminology contains either terms in the thesaurus, or terms defined by the user or all the terms contained in the documents excluding those terms contained in a stoplist. The evaluation of PRESSTO highlighted some shortcomings of the tool. It was found that on tasks where the user needed to understand the retrieved document the tool provided inadequate help. Additionally, the cross-referencing between documents caused the retrieval of documents that were not actually relevant to the search query that is to say it cause false positive search results as defined in Section 3.2.1.2. This was not overcome until the use of hypertext links in MUCH. PRESSTIGE was developed to improve with user understanding of retrieved documents; this required substantial analysis of the industrial domain and the software systems used in steel mills.

PRESSTIGE was developed as a more powerful software tool to support reuse. PRESSTIGE supported the storage and retrieval of surrogates (defined in Section 3.2) as the means to find reusable assets. It contained a domain specific thesaurus and generic frames called questionnaires which when completed comprise the surrogates. An extended Boolean retrieval language, the common command language (CCL), supported retrieval of assets. The questionnaires were searched for terms held in the thesaurus and quantified and constrained by CCL. The thesaurus was developed so that it could be assembled automatically by importing external thesauri or semi-automatically by indexing documents and defining relationships. Some manual development and maintenance of the

thesaurus was necessary. The thesaurus held concepts of the domain (terms), the terminology of the domain (terms), and the relationships connecting the terms. The thesaurus was used to promote user understanding of the context of the terms.

The surrogates or questionnaires were comprised of three parts. Firstly, the assets administration information such as date created and who created it. Secondly, a black box description of the asset's properties such as interfaces with other assets, and its relationships to other assets such as a specific code module's test cases. Thirdly, a clear box description stating the internal structure of the asset such as its sub-parts and their interrelationships.

A software tool, the TeamWork CASE tool, was used to provide a data flow diagram representation of an asset's sub-parts and their interrelationships. The evaluation of PRESSTO demonstrated the difficulties that could arise when documents contain cross-referencing. The evaluation of PRESSTIGE identified problems users could have understanding the relationships between assets and between an asset's sub-parts. This led to the development of MUCH, a collaborative working tool which contained the indexing, searching and browsing functionality of PRESS and PRESSTIGE with additional hypertext functionality, and 'knowledge' of document structures to promote user understanding of reusable assets. MUCH allowed for importation of text documents and provided a predetermined sequence of hypertext links within the document to support understanding of assets. Predefined generic document structures, called outlines, were applied to a document, and then the document was indexed. The index consisted of the document headings that fell into specific areas of the outline.

Rada [RAD90] describes MUCH, as a tool developed to support reuse with a metathesaurus; a thesaurus made up of more than one thesauri. MUCH provided a collaborative work environment for the indexing, browsing, searching and retrieval of reusable documents. Additionally MUCH provided guidance to users on the structure of the thesaurus. Users of MUCH could dynamically generate new documents from existing documents. A user would select terms from the thesaurus, which would then be applied to an established document outline.

MUCH used hyper-link technology to locate paragraphs in existing documents that contained those terms contained in the user-defined outline. MUCH also provided a tool for automatically building thesauri from text. However, the resulting thesauri were not useful. Domain expertise and manual effort are required in the development and maintenance of a thesaurus.

3.5.2 Thesaurus Assisted Understanding

The success of reuse is directly linked to the reuser community's ability to define and understand the reusable assets of the domain [BIG98]. Brooks [BRO95] suspects that one of the problems facing reuse today is the extent of the terminology that must be learned; for example, a class library with over 3000 objects can have objects requiring between 10 to 20 parameters and optional variables. Anyone using the library would need to understand both the external interface (syntax) and the functional behaviour (semantics) of all the objects. As difficult a task as this sounds, Brooks concludes that it is achievable as people do learn the syntax and subtle semantics of a language while acquiring an average terminology of 10,000 words. However, studies have shown that agreement when naming common objects is only likely to occur between 15 and 35 percent of the time. Increasing the number of possible aliases, even to as many as 15, for a common object will only increase agreement to between 60 and 80 percent [HEN94].

Brooks [BRO95] concludes that more research should be done into how people acquire an understanding of language. However, one feature that is understood, is that people's understanding of a language increases when they can place the terms (words and phrases) of the language in context. By placing terms into context and using them, people learn to understand the syntax and semantics of a specific terminology. A thesaurus can be used to help people place terms in context by providing broader terms, narrower terms and related terms. A thesaurus can provide additional richness to the terminology. A thesaurus can be developed to include additional relationships between terms such as preferred term, non-preferred term, synonyms or any relationship required by domain [MIL98]. Practitioner used a thesaurus to promote reuser understanding of the

domain terms and the context in which they were used [MIL94]. In Practitioner the thesaurus aided in the definition of the surrogates and the defining of search queries [MIL94].

3.5.3 Thesaurus Assisted Searching

As a reuser's comprehension of the domain increases their ability to perform reuse improves. Conversely, effective reuse will improve a reuser's understanding of the domain [JAR95]. Henninger [HEN94] asserts that an effective search and retrieval method should be an aid to increasing the reuser's understanding of both the problem domain and the potential solution. Terminology mismatch is a major problem in search and retrieval. Support with surrogate definition and query construction is essential for effective reuse. Difficulties arise when a natural language is used to define surrogates and queries. Terminology mismatches can occur when surrogates are defined by one person and later searched for by someone else. A.Mili et.al. [MIL98] maintain that a thesaurus can provide additional richness to the terminology. A thesaurus that contains a definition for domain terms and incorporates the relationships between terms will help the reuser place the terms into a context thereby improving the construction of search queries. The work done in Practitioner supports the concept that a thesaurus can be used to improve search effectiveness.

3.5.4 Thesaurus Construction

This section presents the issues related to the construction of a thesaurus. The discussion covers both the development and maintenance issues surrounding the construction of a thesaurus. The construction of a thesaurus is constrained by the intended use of the thesaurus. In this discussion the construction of the thesaurus is constrained by the intention of this research to develop the thesaurus as a software tool to support reuse and to record the results of on-going domain analysis i.e. the increased understanding of the domain terminology resulting from the practice of reuse.

3.5.4.1 Developing a Thesaurus

It is generally recognised that the information attached to a term should contain its definition, a broader term, any narrower term and any related terms [RAD90]. When a term can represent more than one domain concept, one interpretation is selected as the standard definition and the others are entered into the scope note [ISO2788]. A scope note is an area designated to hold information about the term that falls outside the range of its definition and relationships.

The ISO 2788 standard [ISO2788] states that it is hierarchical relationships that distinguish a thesaurus from a dictionary or glossary. The hierarchical relationships between terms provide a hierarchical structure for relating the terms. A term is related to either broader (more general) terms and / or narrower (more specific) terms. Within hierarchical relationships care should be taken when adding proper names which can overload a thesaurus. However, if deemed as necessary then when using the proper name to define a surrogate, enter both the proper name (instance) and its broader term (class) in the thesaurus. Highly specific terms should be restricted to terms understood to be at the core of the domain. The inclusion of highly specific fringe terms would unbalance the structure of the thesaurus, thereby making it awkward to navigate.

Terms are designated as related terms when they have an associative relationship. Terms that have an associative relationship but share a common broader term are not designated as related terms. As individual related terms do not form part of the hierarchical structure, a facet indicator is assigned to the related terms. A facet indicator is a word or phrase that does not represent a domain concept but is used to indicate the basis on which a thesaurus has been structured. A facet indicator would not be used in indexing, surrogate definition, querying or understanding assets. Care should be taken to ensure that terms are not being designated as a related term simply because to do so would be easier than locating the correct place in the hierarchy for the term.

Prior to constructing a thesaurus, the form of its structure should be settled. An alphabetical structure provides an alphabetical listing of all preferred or non-preferred terms. Non-preferred terms provide only a reference to the preferred

term. Preferred terms provide all other information, such as definition, broader term, narrower terms, related terms, and scope note. An alphabetical structure is easy to construct and maintain. However, it does not convey hierarchical structure of the concepts within the domain. Domain analysis of the structure of the concepts leads to insight into the systematic or hierarchical structure for a thesaurus.

In an enumerated approach subject areas of the domain would group terms. This method is best when domains include multiple subject areas or for thesauri that are intended to cover multiple domains. The subject areas would need to be defined prior to construction of the thesaurus, as they are difficult to change after terms have been assigned.

In a faceted approach basic features would describe terms. This approach is best when the thesauri are intended for a single subject domain or a volatile domain. It is easy to change a faceted structure, and a higher level of agreement between constructors and users is usually obtained. However, a faceted structure means that the terms are scattered and the hierarchical structure of the domain is not immediately apparent. Aitchison and Gilchrist [AIT72] state that when constructing the initial thesaurus, a faceted structure can mean that complex terms within the domain are missed. This is a result of missing terms that do not fit in the facet under consideration.

A combination of structures is also possible; for instance, a thesaurus can have enumerated subject areas but a faceted structure for each subject area. Effective domain analysis and understanding of user needs will generally provide knowledge required to select the appropriate structure for the thesaurus.

Theoretically, it is possible to build a thesaurus using one of two distinctly separate methods, deductive or inductive [ISO2788]. Rada [RAD90] and the ISO 2788 standard [ISO2788) state that in the deductive method an index comprising an uncontrolled terminology of terms taken from existing assets is created then given to domain experts who construct the thesaurus. This method is useful when a large store of assets are readily available. In the inductive method the domain

experts apply their knowledge of the domain to construct the thesaurus, which is then used to index the assets or define the surrogates using the controlled terminology in the thesaurus. This method is useful when the domain is well understood and the store of assets is as yet small. It is not common practice to apply only one of the methods, but instead to apply a combination of both. This means that the thesaurus and the asset's controlled terminology index are developed side-by-side. For example a number of assets are indexed, those terms plus others added by a group of domain experts are placed in the thesaurus. Over time more assets are indexed, any terms not known in the thesaurus are reviewed, defined and given to domain experts as candidates for addition to the thesaurus. It is possible to automate the indexing of an asset and to provide not only the terms contained in the asset but also the frequency with which the term occurs in the asset.

From his review of Practitioner (see Section 3.5.1.3), Rada [RAD90] ascertained that thesauri automatically developed from text were not useful and that the development of successful thesauri requires considerable human effort. The ISO 2788 standard advises that prior to inclusion in the thesaurus, a term, its meaning and its relationships be verified in technical sources and with domain experts. Additionally, consideration should be given to the user community's understanding of the terms. The source of the term, its date of inclusion in the thesaurus and the names of any authorities consulted on its definition or relationships must be recorded. During initial development of the thesaurus it is likely that a domain expert will include terms which have not occurred during indexing or the assets. These terms must be carefully identified. Once the term is encountered in an index the identifier is removed.

It is recommended that a thesaurus be the subject of a pilot scheme prior to general publication. A selected group of users from the intended user community should be given the opportunity to recommend changes to the terms and their defined relationships. Though obliged to review the recommendations, the developers should not be obligated to incorporate the recommendations into the thesaurus.

Aitchison and Gilchrist [AIT72] state that it is possible to reuse existing thesauri, if they are available for the domain under consideration. The decision to reuse existing thesauri should be based on their availability and whether the effort required to review and update the existing thesauri would be less than developing a new thesaurus from scratch.

The ISO 2788 [ISO2788] standard advises that during development, consideration must be given to the issues surrounding the maintenance of the thesaurus. The issues pertaining to the maintenance of a thesaurus are presented in the next section.

3.5.4.2 Maintaining a Thesaurus

The ISO 2788 [ISO2788] standard advises that the issues surrounding the maintenance of the thesaurus be given consideration during the development of the thesaurus. The two main issues surrounding the maintenance of a thesaurus are changes to the domain and usefulness of the thesaurus. Change to the domain and the user community must be reflected in the contents of the thesaurus. The usefulness of the terms within the thesaurus must be measured.

Actual modification of the thesaurus should be restricted to those who have an expert understanding of the domain and the structure of the thesaurus. However, users of the thesaurus must have the means to communicate their need for changes to the thesaurus. The standard recommends that the mechanism for requesting change to the thesaurus should be in place when the thesaurus is initially used. Even something as simple as filling in a change request form would be sufficient.

The standard advises that use of terms within the thesaurus be measured over some predetermined period. What to measure and how to record the measurements are issues that should be settled prior to general release, and therefore should be considered during development. Measurement should begin with the general release of the thesaurus. It is suggested that the use of a term be measured i.e. the number of times a term is interrogated, used in defining a

surrogate, used in defining search queries, and found in an indexed asset. Unused terms over some defined time period are candidates for deletion, however, if a term has been used in surrogate definition then deletion is not possible. It is therefore recommended that the thesaurus allow for a term to be marked as "for retrieval purposes only" without any actual deletion occurring. Terms that are over used are terms that are candidates for splitting into two or more specific, probably narrower, terms.

It is recommended that thesauri be rigorously reviewed on a regular basis. These reviews are considered necessary to ensure that the thesaurus reflects the changes in the domain and the needs of the user community.

3.6 Summary

This chapter contains the results of a literature survey exploring support for component-based reuse and domain analysis. Specifically it contains a detailed examination of the library based component reuse and the thesaurus proposed in Chapter 2 as an aid to component-based reuse and domain analysis.

Component-based reuse is only successful if the reusers can locate assets to be reused in the development of new software applications [FRA94]. When understood a domain's terminology can be used to aid reusers in providing consistency to the terms used in surrogate definitions and search query constructs. In addition, effective search and retrieval method should be an aid to locating the potentially reusable assets while increasing the reuser' understanding of both the problem domain and the potential solution [HEN94]. To promote effective reuse and increased domain understanding a domain's terminology must be defined in a way that places the terms in context. A thesaurus can be used to help reusers place terms in context by providing the additional richness to the terminology that comes with defining the relationships between the terms within the domain [MIL98]. Practitioner has shown that a thesaurus can promote reuser understanding of the domain terms and the context in which they were used [MIL94].

Though it is possible to automate some aspects of the construction of a thesaurus, such as indexing assets to find domain specific terms for inclusion in the thesaurus, developing and maintaining a thesaurus requires considerable effort from domain experts [RAD90]. The domain experts need to understand the domain terminology and the structure of the thesaurus [ISO2788].

In this research, the approaches to software reuse reviewed in the previous chapter and this chapter will be applied to support design reuse in the domain of roll design within the steel industry. The aim is to show that specific software reuse techniques can be applied to support reuse in other engineering disciplines.

Chapter 4 Domain Analysis

4.1 Objectives of the Chapter

The main objective of this chapter is to provide the results of the domain analysis performed as part of this research within British Steel's roll design community. Analysis of this domain consisted of discussions with domain experts including British Steel roll designers, and academic members of CARD[12] and REMAIN[13]. In addition, an examination of a small sample of British Steel roll design documents was performed.

Section 4.2 provides an overview of the domain. Section 4.3 provides details of the domain that are relevant to this thesis. Section 4.4 provides the details of the problem domain when put into a Software Engineering context. Section 4.5 provides the summary for this chapter.

4.2 The Domain Overview

British Steel is an international company in the steel products manufacturing industry. British Steel has several steel rolling mills. The purpose of a rolling mill is to take steel in some initial form such as an ingot or a slab and transform it into a final structure required by a customer such as an I-beam or rail. The transformation process requires that the initial steel form (e.g. a slab) be passed though a series of one or more sets of shaping rolls forming a predetermined final shape (e.g. an I-beam). The steel is passed one or more times through the section forming a series of intermediate shapes before reaching the desired final shape (e.g. an I-beam). The number of passes through the section is dependent on the mill being used, the initial shape of the steel, the rolls design, the intermediate shapes and the final shape to be achieved. A mill site may contain a number of sections. It is the job of the roll designers to design the rolls in the section in such a way that will ensure the finished product is of high quality while at the same time containing the manufacturing costs.

[12] Computer Aided Roll Design is an EPSRC SEBPC project

British Steel's roll design division is in the process of moving from decentralised roll design environments, where each mill designs its own rolls autonomously, to a centralising roll design environment, where design documents are stored and accessed within a central repository. The design documents vary in both size and format. The advantage of a centralised design environment is that all design documents will be available to the entire company's roll design community. British Steel would like the means of exploiting this advantage to improve their design process.

In conjunction with the move to a centralised design environment, British Steel is faced with the imminent retirement of several experienced roll designers. The company would like to be able to capture and make available for reuse the wealth of knowledge and experience held by the retiring roll designers. Ideally high-level design documents would contain a large portion of that knowledge. However, at present there are a limited number of high-level design documents written, though the number is expected to increase. However, there are lower level design documents, known as D++ documents that contain a substantial portion of the knowledge held by the retiring roll designers.

4.3 Domain Details

Within the scope of this thesis the domain has two major components, the design documents and the roll designers (designers). Section 4.3.1 provides the domain details on the design documents. Section 4.3.2 provides the domain details on the roll designers.

4.3.1 The Design Documents

The design documents represent the roll design at various levels in the design hierarchy and provide different levels of design detail. The lowest level of design documents are intricate and detailed drawings. Above these are D++ design documents that are machine-readable documents used to automatically generate

[13] A British Steel funded project on the Application of Software Engineering Techniques for Re-use and Maintainability to Computer Aided Roll Design

drawings. D++ design documents are a slightly higher level of design document intended to provide a diagrammatic design with a degree of analysis specific to the steel industry. Above the D++ diagrams are HTML design documents. The HTML design documents are generated directly from the D++ diagrams. The HTML design documents represent the design in increasing layers of detail and provide hypertext links amongst their layers. Above the HTML documents are high-level design documents consisting primarily of free text in fixed formats. The high-level design documents provide detailed information pertaining to industry methods and/or specific products.

Not only is there a variation in the kinds of design documents but there is also a variation on the availability of the different kinds of design documents. There is a low-level design document (drawings) for every product manufactured at each mill. There are a large number of D++ design documents for a cross section of products at each mill. There is at least one, possibly more, HTML design document for each D++ design document. There are a limited number of high-level design documents. However, it is the high-level design documents that have the potential to include the largest amount of design knowledge and the production of these documents is being encouraged at British Steel.

Designers often retrieve existing design documents to assist them in creation of new designs. The roll designers are most likely to retrieve documents with which they are familiar. Often these are design documents that they themselves have written. At present each rolling mill has its own design environment. Documents for each mill are written and stored within the mill's design environment. Though in the same industry and designing similar products, each mill has idiosyncrasies in the terminology that it uses, which is reflected the design documents. As an example the measurement terms "depth" and "width" are equivalent in that they measure the same thing. The difference between the terms is only evident if the roll designers know whether or not they are reading a design document that is describing an I-beam or an H-beam. This can be seen in Figure 4.1 below.

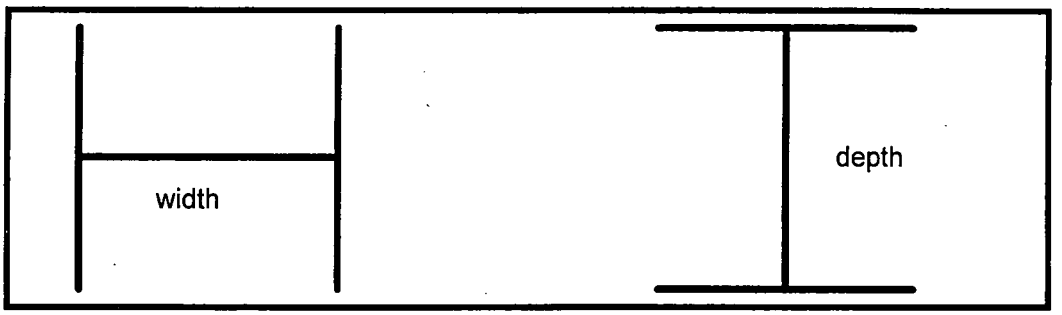

Figure 4.1 Illustration of Equivalent Terms

The beams are manufactured at different mills. There is no formal standard for the terminology to be used in design documents. The terminology in a design document will reflect the mill the roll designer works in. The terminology used in a design document may also reflect the product the roll designer designs for and the designer's personal preferences.

There is a glossary of terms available to the roll designers, the Glossary of Roll Design Terms [BS97]. This glossary was an attempt by a few roll designers at British Steel to help the roll design community overcome problems that resulted from misunderstanding related to their domain terminology. The glossary was intended to be an alphabetised listing of all roll design terms, where each term was to have a definition and a listing of associated terms. For the most part this is what was achieved. However, the glossary was difficult to compile as a few willing roll designers constructed it by hand. The glossary's development was not automated in any way; domain experts selected and defined the terms. Once developed the glossary was not significantly maintained. Though a substantial piece of work it is possible to use the contents of the glossary to demonstrate the terminology problems within the domain. Below is a list of domain terminology problems, with examples taken directly from the glossary.

- Many of the terms have one or more alternative definitions.
  For example, the term "Wobbler" is defined as "Type of universal joint used at roll end" and has the alternate meaning "Fluted end of roll".

- In some cases the alternative meaning is used as an area to write a note to the reader of the glossary.
  For example, the term "Camber" is defined as "A tendency to bend sideways as the work piece comes out of the roll gap". The alternative meaning for the term "Camber" is "Not the same as thermal camber".

- Some terms have the same or very similar definitions.
  For example, the terms "Groove", "Pocket" and "Pass". "Groove" is defined as "The shape actually cut into one roll for one pass shape". The term "Pocket" is defined as "Shape cut in roll". The term "Pass" is defined as "The shape formed between grooves cut in two rolls".

- A number of terms have no definition and no associated terms.
  For example, the terms "Billet" and "Box Hole Collar".

- The association or relationship between the terms is not defined.
  For example, the term "Barrel" is defined as "The working portion of the roll available for cutting grooves". The alternative meaning for "Barrel" is "Rolls used for flat products often have a barrel shape - slightly larger in diameter in the middle". A term associated with "Barrel" is "Neck". The term "Neck" is defined as "The end of the roll that accepts the bearings and drive couplings". The alternative meaning for "Neck" is "the straight part of a sheet piling section leading to the lock". A term associated with "Neck" is "Barrel". The four definitions provided are not identical or similar even though the terms "Barrel" and "Neck" are associated

- The glossary does not indicate if a term is mill specific though some definitions imply a restriction to a particular mill or product.
  For example, the definition for the term "Droop" is "Lock hanging down on Larssen Piling".

- The glossary does not make clear which of the terms is an accepted standard either in the roll design community or the industry in general.

4.3.2 The Roll Designers

There is a wide variance in the industrial and design experience of the roll designers in British Steel. This variance ranges from roll designers with many years experience as roll designers for British Steel, to recent graduates with no relevant work experience. Experienced roll designers may have experience of several mills or sections, or their in-depth knowledge may be restricted to a single mill or section. British Steel would like to capture the knowledge of roll design held by the experienced roll designers and make it available to less experienced roll designers. The need to capture this knowledge is made more urgent by the fact that some of the more experienced roll designers are due to retire. There is at present no formalised process or mechanism in place to aid with the capture and sharing of roll design knowledge. The intended centralised roll design environment will provide a central area for storage of all design documents, but at present there are no tools to aid with search and retrieval of design documents. Experienced roll designers are able to locate the existing design documents that they need when developing a new design. However, there is no formal mechanism in place to help less experienced roll designers to locate existing design documents that could be used when they are developing new design documents. There is no formal mechanism in place to help roll designers of any level of experience to develop new design documents so that they can later be used to help other roll designers develop design documents. There is no formal mechanism in place to help roll designer to browse potentially useful design documents i.e. design documents that can be used to increase knowledge and aid in clarifying what is needed in a new design.

4.4 The Problem Domain in a Software Engineering Context

British Steel's roll design community is moving from decentralised roll design environment to a single centralised roll design environment. A centralised design environment has the potential to improve the roll design process, improve the quality of the design documents produced by the community and improve the roll design communities ability to capture and share domain knowledge. By providing a central repository (a library) of design documents (assets) it is anticipated that roll designers (reusers) will be able to search and retrieve existing design

documents (reusable assets) when developing new design documents (assets). British Steel needs to stop the depletion of the knowledge base (domain knowledge) that the imminent retirement of several experienced designers is likely to cause. Knowledge capture and sharing is possible through domain analysis and domain engineering or domain modelling. In software engineering terms this means that British Steel's roll design community needs to be able to reuse existing assets, which are for the most part natural language documents. In addition, British Steel's roll design community needs to be able to develop new assets for reuse while increasing their understanding through analysis of the domain.

To date there has been no planned or managed process for the practice of domain analysis and reuse. Software tool support is needed so that the roll design community (reusers) can begin to do the domain analysis and reuse concurrently, and in a planned and managed way. There needs to be software tool support to enable the reusers to search and retrieve assets from the library to meet varying search goals, including the gleaning of knowledge, browsing to aid when the problem definition is unclear, and to reuse existing assets in the development of new assets. This will require not only software tool support but also an understanding of the terminology used in the domain. There needs to be software tool support to assist reusers in developing new assets in a manner that will support the assets' later reuse. This will require not only software tool support but also an understanding of the terminology to be used in the new assets. There needs to be software tool support to aid reusers in the extraction of knowledge contained in existing assets so that the knowledge can be stored and shared with the reuser community. It should be noted that extraction of knowledge from existing assets is likely to be performed by reusers with varying levels of domain knowledge. Software tool support is needed to help the reusers overcome the terminology issues raised as a result of reusing domain specific natural language assets.

There are six issues surrounding the reuse of the natural language assets developed by the British Steel roll design community. They are as follows:

1. The terminology contains terms that have more than one meaning. Which of these meanings the reuser community more commonly applies is not apparent.
2. The terminology contains groups of terms (two or more) with the same meaning. Which of these terms is more commonly in use in the reuser community is not apparent.
3. There are relationships between the terms that have not been defined. These relationships may be equivalence, hierarchical or associative.
4. Reusers may not understand terms used by other reusers.
5. Reusers with limited experience in the domain are not familiar with some of the terms in the domain.
6. There is no single source the reuser can access that will clarify the meaning of the terms in the terminology or the relationships between the terms in the terminology.

4.5 The Summary

British Steel's move to from a decentralised roll design environment to a centralised design environment has the potential to improve the roll design process, improve the quality of the design documents produced by the community and improve the roll design communities ability to capture and share domain knowledge. The roll designers need to be able to reuse existing design documents, develop new design documents that are suitable for reuse while increasing their overall understanding of the domain. The practice of domain analysis and reuse will require software tool support. The requirements for the software tool support are presented in Chapter 5.

Chapter 5 Requirements for Tool Support

5.1 Objectives of the Chapter

The main objective of this chapter is to provide the requirements specification for a prototype of the software tool support needed by British Steel roll design community to enable them to perform domain analysis and reuse of domain assets concurrently. This chapter presents an initial set of requirement based on a general understanding of the problem domain. An initial prototype based on the initial set of requirements will be developed to test the accuracy and completeness of the initial requirements. A review of the initial prototype will be performed to identify any misconceptions about the problem domain and to glean any additional requirements. A final set of requirements for the prototype will be based on the review.

Section 5.2 provides an overview of the requirements. Section 5.3 provides the initial requirements for the software solution. Section 5.4 provides details of an initial prototype of the software tool support and how it was used to glean additional information about the roll design community. Section 5.5 provides the final requirements for the required software tool support, which reflects the additional information discovered by using the initial prototype. Section 5.6 provides the summary for this chapter.

5.2 Overview of the Requirements

British Steel's roll design community is moving from decentralised roll design environments to a centralised roll design environment. British Steel's roll design community needs to be able to perform, concurrently, ongoing analysis of the roll design domain especially vocabulary/terminology analysis to support component-based reuse. To perform ongoing domain analysis roll designers (reusers) need to locate and extract domain knowledge from assets that exist and to store that knowledge in a manner that will allow other reusers to share it. The reusers will need to locate and extract domain knowledge from newly developed assets and store that knowledge in a manner that will allow other reusers to share the domain knowledge. To perform reuse the reusers need to populate a library with

those assets that already exist and those that are to be developed. The assets and the library must be structured to promote search and retrieval of reusable assets. The reusers need to search and retrieve library assets to satisfy one of the three distinct reuse needs listed below.

- Reuse to increase domain knowledge.
- Reuse to develop new assets.
- Reuse to clarify the development needs of new assets.

The reusers need to develop new assets in a manner that will support the assets' later reuse. Software tool support is required to sustain component-based reuse, while performing domain analysis. The terminology issues that arise as a result of reusing natural language assets without an enforced standard can have a detrimental effect on both component-based reuse and domain analysis. Software tool support will be needed to assist the reuse in resolving the following issues. There are terms in the terminology with more than one meaning. The terminology contains different terms that have the same meaning. The relationships between the terms are not defined. There is no single source the reuser can access that will clarify the meaning of the terms in the terminology or the relationships between the terms in the terminology. Reusers from one work area may not understand terms used by reusers in another work area. Reusers with limited experience in the domain are not familiar with some of the terms in the domain.

5.3 Initial Requirements

This section contains the initial requirements for a prototype of a software tool that could be used to support reuse and increase domain knowledge. As part of the requirements process a first pass or initial prototype, Reuse Support Tool (ReST) is to be developed to demonstrate software tool support for reuse and ongoing domain analysis.

It is proposed that as reusers' understanding of the terminology used in the domain increases understanding of the domain will also increase. A thesaurus is a single store for the terms used in the domain. It contains a definition for each term held in the store, and is used to identify the relationships between the terms

of the domain. The prototype, ReST, will be used to demonstrate the reuse of existing assets and the development of new assets for reuse. ReST will be used to demonstrate that a thesaurus could be used to support reuse by providing a reuser with assistance when defining an asset's surrogate, constructing search queries and understanding the domain's terminology. Eight initial requirements are listed below:

1. Develop a domain specific thesaurus that will hold domain specific terms, a definition for each term and the relationships between the terms. The thesaurus should make clear which of the terms are considered standard terms (preferred terms) within the reuser community.

2. Populate the thesaurus with domain specific terms that are found in the assets.

3. Assist a reuser to populate the reuse library by constructing surrogates for each potentially reusable asset. A surrogate should contain only those terms that occur in its asset. A surrogate should contain only those terms that are defined by the reuser community as standard terms. A surrogate should contain only those terms that are defined in the thesaurus as standard or preferred terms.

4. Allow a reuser access to the thesaurus when constructing the surrogates. The thesaurus will identify those terms that are standard (preferred) terms within the reuser community.

5. Enable a reuser to search the library for reusable assets. As surrogates consist of standard terms only, search queries will consist of standard terms only.

6. Allow a reuser access to the thesaurus when defining a search query. The thesaurus will identify those terms that are standard (preferred) terms within the reuser community.

7. Assist reusers in understanding the terminology of the domain. This assistance will be in the form of access to the thesaurus. The thesaurus will

help the reuser put the terms in context by providing domain specific terms, the definition for each term and the defined relationships between the terms.

8. Index the assets to provide a list of unique terms and the frequency with which those terms occur in an asset. The index will be evaluated to provide a quality assessment of the terms used in the asset to the reuser. The index will be used to define the surrogate, populate the thesaurus, and populate a stoplist, which is a tool used in indexing.

5.4 Initial Prototype

As part of the requirements gathering process an initial prototype of the Reuse Support Tool (ReST) was developed. The overall intention for the development of the first prototype of ReST was to increase the understanding of the problem domain, better assess the needs of the reuser community and to demonstrate the initial requirements. The first prototype of ReST was constructed and then used in a demonstration that examined examples of possible input and the likely resulting output.

Although previously studied reuse support systems such as those developed by the Practitioner project have contributed to the development of the prototype at the requirements level, the prototype has been constructed independently. Reuse of the earlier systems was not feasible given their implementation technology.

The initial prototype of ReST was developed using Microsoft ACCESS version 1.1, which provided a simple user interface. Background documents used to glean samples of possible input and output data were the Glossary of Roll Design Terms [BS97] and Expert Roll Design [SML98]. No actual processing was implemented; instead database tables holding samples of input and expected output data were used to simulate actual processing. Microsoft ACCESS forms were exploited to create a simple user interface. Internal macros such as OPENFORM were used to simulate the user interface functionality.

The initial prototype of ReST was developed and demonstrated in conjunction with two scenarios. Scenarios are an ordered list of simple tasks that a reuser would need to perform to complete a body of work, such as creating an asset's surrogate and adding it to the library. The initial prototype of the ReST was developed to show the understanding of simple tasks and demonstrate how ReST could be used to overcome terminology problems likely to occur while performing those tasks. It was shown to a potential reuser (designer) and interested academics working on the CARD and REMAIN projects at a formal demonstration.

Each scenario is based on an ordered list of simple tasks that need to be performed to accomplish a body of work. For example to create a document surrogate and add it to the library a reuser must perform several tasks. To begin with the document is indexed. This provides a list of terms used in the performance of a quality check. This provides a breakdown of terms into categories, such as preferred terms, non-preferred terms, and terms that are undefined, as yet, within the domain. Decisions on actions to be taken concerning the terms in each category need to be made. Terms that are to be placed into the surrogate need to be identified. Accompanying each series of tasks was a series of questions. These questions arose during the development of the initial prototype of ReST and identified areas of the problem domain and solution that were unclear. The scenarios with the questions were used in conjunction with the initial prototype of ReST to demonstrate the initial requirements, clarify domain understanding and generate general discussion. Copies of the scenarios were distributed during the demonstration of ReST. Each scenario was enacted, while questions were asked and general discussion encouraged. Answers to questions explicitly asked were recorded during the demonstration. Where questions were not explicitly asked it was possible to glean answers from the general discussion.

Figure 5.1 contains a screen shot in the initial prototype of ReST and is one of the screens used during the demonstration of the initial prototype of ReST. It contains the result of indexing an asset and the processing that is available to a reuser once indexing is complete. The asset was not actually indexed. Terms

were manually selected from a British Steel document "Expert Roll Design" [SML98] and entered into a database table where they were stored.

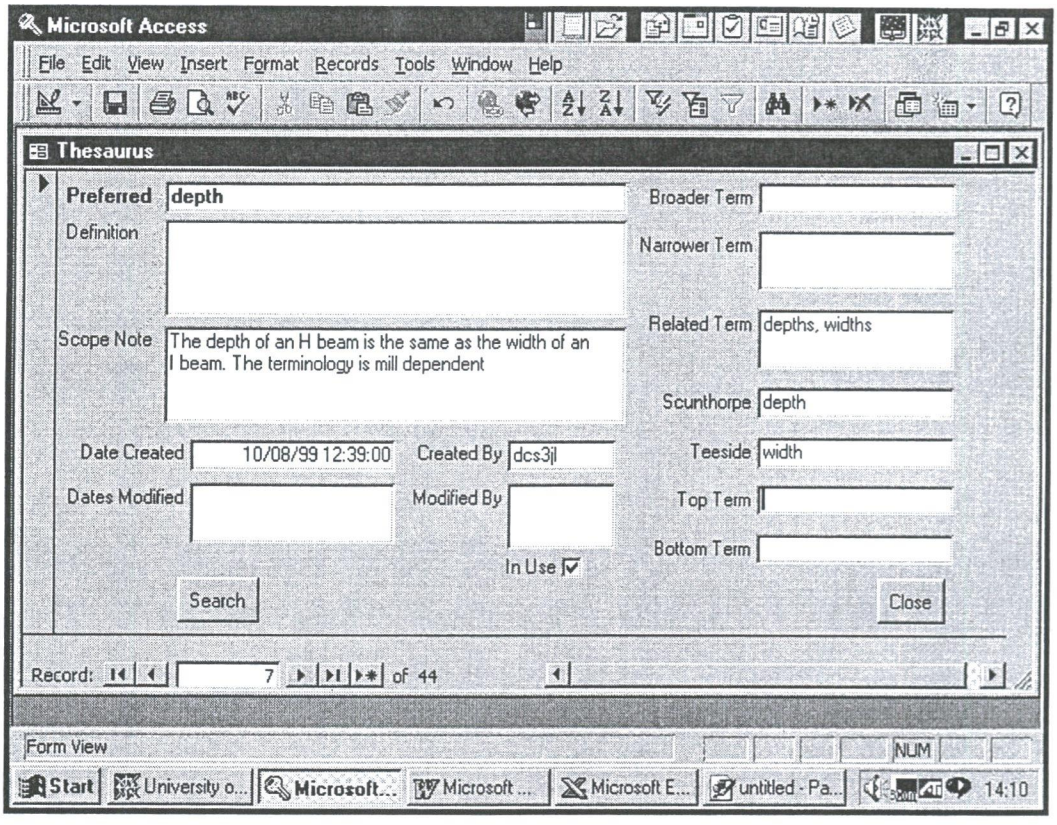

Figure 5.1 ReST Index Document Complete

5.4.1 ReST Scenario One

The body of work is the creation of a document's surrogate and its addition to the reuse library.

Actions/Tasks	Questions	Answers
Index the document.	Which of the following should be included with the list of unique terms: a frequency count; and / or the place term occurs in the document;	Both - frequency count will help define relevance of terms in surrogate, experienced designers will only need to see relevant sections. Would be useful if terms could be highlighted in document.
Do quality check on index terms	Index terms to be included in the surrogate are: all index terms; all index terms also in the thesaurus; or all index terms also in the thesaurus and defined as a preferred term?	All terms used in surrogate should be uplifted to preferred terms.
From the index terms extract the subset that are in the thesaurus, but not a preferred terms.	Should the non-preferred terms found in the document be changed to preferred terms?	No
From the index terms extract the subset which are not in the thesaurus.	Would it be useful to track the number of non-preferred terms and unknown terms that occur in documents over time?	Unknown
Index terms not in the thesaurus should be either: added to stoplist; added to thesaurus; saved to file or ignored.	No questions.	No comments.

If all terms needed for surrogate are now in the thesaurus, enter the surrogate details.	What form does the surrogate take? Text summary, a list of key phrases or other?	Copies of actual surrogates to be sent

Figure 5.2 contains a screen view of the Thesaurus in the initial prototype of ReST. Not all of the possible relationships between terms in the domain were demonstrated in the thesaurus. The only relationship demonstrated was the equivalence relationship which is defined as either USE or USED FOR, as in non-preferred USE preferred, and preferred USED FOR non-preferred. The intention of the initial thesaurus was to demonstrate the concepts behind using and maintaining a thesaurus and not domain knowledge therefore the definition was not included.

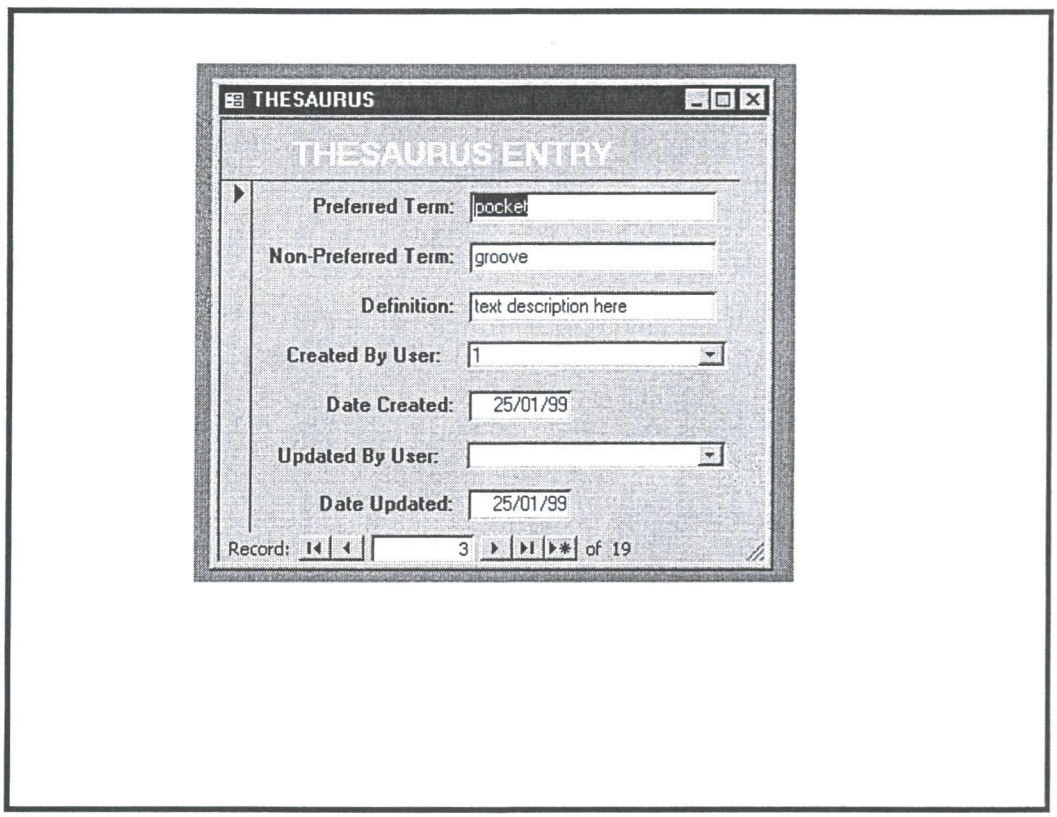

Figure 5.2 Thesaurus in the initial prototype of ReST

5.4.2 ReST Scenario Two

The body of work is the retrieval of relevant documents from the reuse library.

Actions/Tasks	Questions	Answers
Enter Search Term e.g. open flange	Which of the following would be useful: terms with a minimum frequency count; and / or BOOLEAN operators (AND, OR, NOT). Should search terms contain only standard (preferred) terms or any term?	Frequency counts - Yes BOOLEAN - Not Asked Query contains any term, which are automatically uplifted prior to initiating search.
Do search	Should the search results contain the document ID and: the full surrogate; the relevant sections of the document; the location of the relevant sections in the document; the entire document; or the surrogate and the functionality available to allow the user to select one or more documents from the search result to be retrieved?	All, experienced designers would only need to see limited sections of a document, new designers may need to make use of a wider amount of the documents
Refine the search.	When search results are too large would it be useful to refine the search by searching previous results only? Would it be useful to refine searches using relationship used in the thesaurus such as broader or narrower terms? Should terms be automatically found or selected by user?	Yes Yes Both - with the ability to switch automation on/off.
General Observation	During searching would it be useful to have the thesaurus open and on screen or invisible until explicitly called by the user?	Explicitly called.

5.4.3 Results of the Demonstration of the Initial Prototype of ReST

The demonstration of ReST with the scenarios established that generally the proposed solution met the needs of potential reusers. Additional insights into the reuser community were gleaned. Inexperienced reusers are likely to require most of the functionality proposed by ReST all the time. They will be using it not only to locate reusable assets but also to increase their knowledge of the domain and its terminology. As a reuser's level of experience in the work place increases their need to use ReST will decrease. In this case the constant presence of ReST on the screen would be annoying and automating the assistance would be considered intrusive. It was agreed generally that the best approach would be to provide as much functionality as possible but to allow the reuser to decide when to bring the functionality into play.

After the demonstration British Steel provided a hard copy of an additional high-level design document to be used as sample data for a functioning prototype of ReST. The document "Notes on Designing Primary Rolls with One Beam Shape

Forming Pass" [ORD99] is a long detailed text document. Four pages of this document were entered as a text only document and used as an additional sample data in the development of the final prototype for ReST.

5.5 The Final Requirements for the prototype ReST

This section contains the requirements specification for the final prototype of ReST (Reuse Support Tool). The requirements are based on the domain analysis contained in Chapter 3, the initial requirements for ReST and the results of the formal demonstration.

5.5.1 Requirements Specification for ReST

This section contains a listing of the overall requirements for ReST, the characteristics of potential reusers and general constraints and assumptions that will affect the development of ReST.

5.5.1.1 Functionality of ReST

The final prototype for ReST should provide sufficient functionality to demonstrate how to index a design asset, construct a surrogate, search and retrieve relevant surrogates, and capture and maintain the domain knowledge to be held in the thesaurus.

The final prototype ReST, Reuse Support Tool, will be used to demonstrate how a software tool could be used to assist reusers to:

- index a design asset providing a list of unique terms and the frequency with which those terms occur in the asset;
- classify each term contained in the index as either a preferred term, or a term defined in the thesaurus, or a term not in the thesaurus, or an internal stoplist term, or a stoplist candidate term, or a candidate term for inclusion in the thesaurus, or a search term, that is included in an asset's surrogate;
- construct surrogates using preferred terms only;
- construct search queries using preferred terms only;
- populate the thesaurus with terms found in the assets; and

- maintain the thesaurus and the stoplist.

5.5.1.2 User Characteristics

There is a wide variance in the experience and educational backgrounds of the company's rolling mill designers. The most likely users for a software support tool that is the implementation of ReST fall into four categories:

- Reusers with many years experience within the company particularly in rolling mill design section.
- Reusers with several years experience in the company, but not necessarily in the rolling mill design section.
- Reusers with several years experience as engineers or designers in a similar industry.
- Reusers with little practical experience, such as recent university graduates or apprentices.

It should be noted that only users with experience in British Steel rolling mill design should be allowed to write to the stoplist and the thesaurus.

5.5.1.3 General Constraints and Assumptions

ReST will be developed using Microsoft Access 97. Data will be stored and manipulated in tables.

It is assumed that:

- there will be a central repository for design assets;
- individual design assets can be uniquely identified;
- all reusers will be able to read the design assets;
- all reusers will be able to define surrogates;
- all reusers will be able to propose candidate terms for inclusion in the stoplist and thesaurus including suggested definitions and relationships; and
- only a subset of reusers will be allowed to write to the stoplist and the thesaurus.

5.6 Summary

This chapter contained the requirements specification for the software tool to support needed by the British Steel roll design community to enable them to perform domain analysis and reuse of domain assets concurrently. The final prototype for ReST will demonstrate the functionality needed to: index a design asset; construct a surrogate; search and retrieve surrogates; and populate and maintain the thesaurus. The prototype will demonstrate how the thesaurus can be used to: assess the quality of an asset; identify an asset's terms to be included in a surrogate; aid with the construction of a search query; and store the domain terminology.

Chapter 6 Design of ReST

6.1 Objectives of the Chapter

The main objective of this chapter is to provide the design details of the final prototype of ReST (Reuse Support Tool). The design is based on the final requirement in Chapter 5. Dataflow diagrams are used to identify the entities, processes and data that comprise ReST. Entity-Relationship diagrams are used to show the relationships between the entities that comprise ReST.

Section 6.2 contains the design details of ReST. Section 6.2.1 contains a series of dataflow diagrams with textual descriptions explaining the data and the processes of the proposed solution. Section 6.2.2 contains a series of entity-relationship diagrams with textual descriptions explaining the relationships between the entities of the proposed solution. Section 6.3 provides the validation of the design against the requirements for the final prototype of ReST stated in Chapter 5. Section 6.4 contains a summary of this chapter.

6.2 ReST Prototype Design

This section provides the design details of the final prototype of ReST. The design includes dataflow diagrams that are used to identify the entities, process and data that comprise ReST. In addition, entity relationship diagrams have been provided to define the type of relationships that exist between the many entities that comprise ReST.

6.2.1 Dataflow Details

In this section a series of dataflow diagrams illustrates the design of the final prototype of ReST. The first diagram is the context diagram. This diagram is used to portray ReST in the context of British Steel's proposed centralised design environment. It identifies those entities outside of ReST with which ReST will need to co-operate in order to function. The remainder of this section follows the logical decomposition of the Context Diagram into the level one, two and three diagrams. Decomposition is based on the identification of processes and the data

needed as input. Also shown is each process' resulting output if any, as well as any associated internal data stores.

6.2.1.1 ReST in Context of the Domain

Figure 6.1 contains the dataflow context diagram for ReST. The main purpose of the context diagram is to identify the three distinct users of ReST. All three users of ReST are represented as external entities in the context diagram. Each entity:

- represents a sub-set of British Steel rolling mill designers;
- interacts with ReST to perform distinctly different tasks;
- provides specific external data needed to initiate processing in ReST; and
- must possess a different level of knowledge and experience to perform the tasks and provide the specific external data.

The main difference between the entities is the tasks that the designer is performing. There may be any number of Reusers, Librarians, or Maintainers. A single designer may be a Reuser, a Librarian and a Maintainer. Whether a designer is interacting with ReST as a Reuser or Librarian or Maintainer is dependent on the task the designer is performing and the external data they are entering into ReST. A single bubble entitled ReST is used to encase all the processes, data stores and internal data of the prototype.

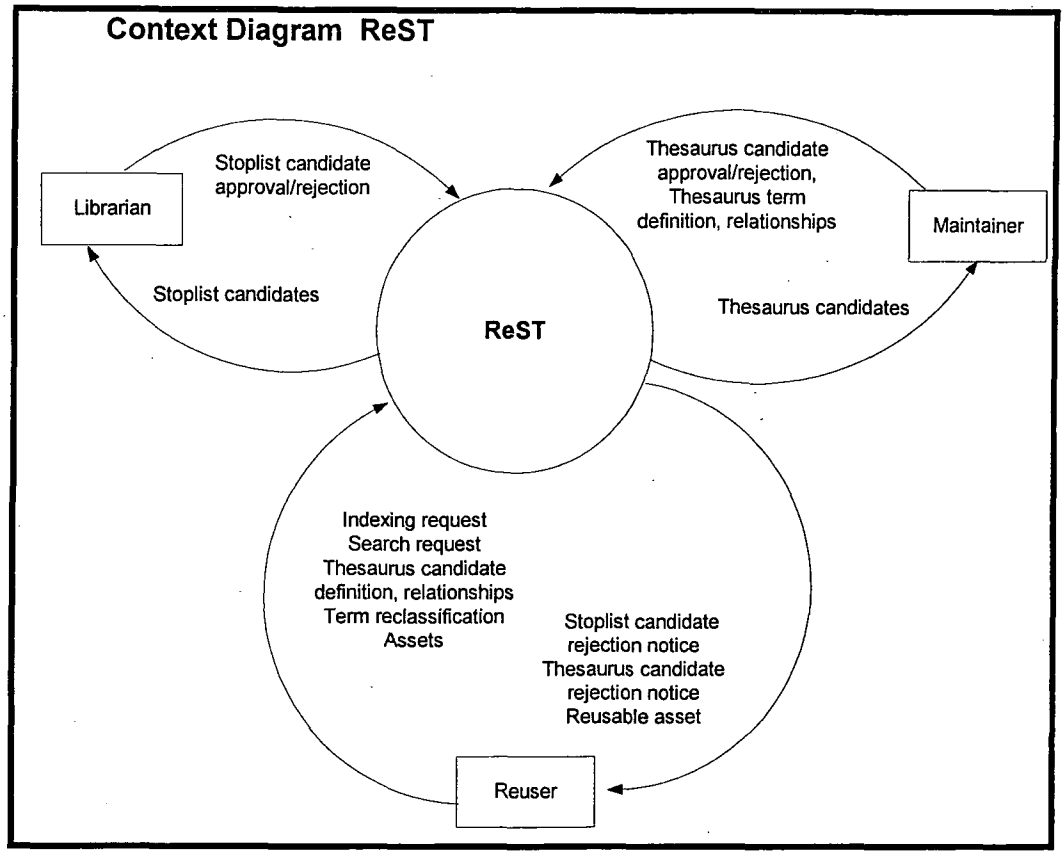

Figure 6.1 Context Diagram

A Reuser is any British Steel rolling mill designer from a recent graduate to an experienced designer with many years experience within British Steel's rolling mill design community. A Reuser will use ReST to perform any one of three main tasks. The first is to provide the design community with reusable design assets by indexing a design asset and then classifying the index terms. The second is to locate reusable design assets. And the third is to nominate index terms as candidates for either the stoplist or the thesaurus.

A Librarian is any British Steel rolling mill designer with sufficient knowledge of the domain vocabulary to be able to maintain the stoplist and with the permission to do so. Though any Reuser can nominate a word for inclusion in the stoplist, the actual addition to the stoplist requires the intervention of a Librarian. The Librarian must explicitly approve a stoplist candidate before it can be included in the stoplist. The Librarian can also reject any candidate for inclusion in the stoplist. For example the word "line" can occur often enough in an asset's index as to render it meaningless in context of the asset. It would be logical for a Reuser

to submit it as a candidate for inclusion in the stoplist. However, a Librarian would have to reject "line" for inclusion in the stoplist because it is a word that is necessary within domain specific terms such as "centre line". "Centre line" is a standard industry term and would need to be included in the thesaurus as a preferred term.

A Maintainer is any British Steel rolling mill designer with sufficient knowledge of the domain to maintain the thesaurus and with the permission to do so. The Maintainer not only needs to understand the domain vocabulary, but must also understand the relationships between the terms in the vocabulary and the precise structure of the thesaurus that contains the terms and their defined relationships. Though any Reuser can nominate a term for inclusion in the thesaurus, the actual addition to the thesaurus requires the intervention of a Maintainer. The Maintainer must explicitly approve a thesaurus candidate before it can be included in the thesaurus. Any Reuser may propose a term's definition and proposed relationships for the term. However, only a Maintainer can explicitly define the term and enter its relationships in the thesaurus. A Maintainer can reject any candidate for inclusion in the thesaurus.

6.2.1.2 The Processing within ReST

Figure 6.2 contains a level one dataflow diagram of ReST. This diagram is used to illustrate the four main process areas of ReST. Each process area is encased in a bubble that contains sub-processes that are illustrated in the level two and three dataflow diagrams provided later in this section.

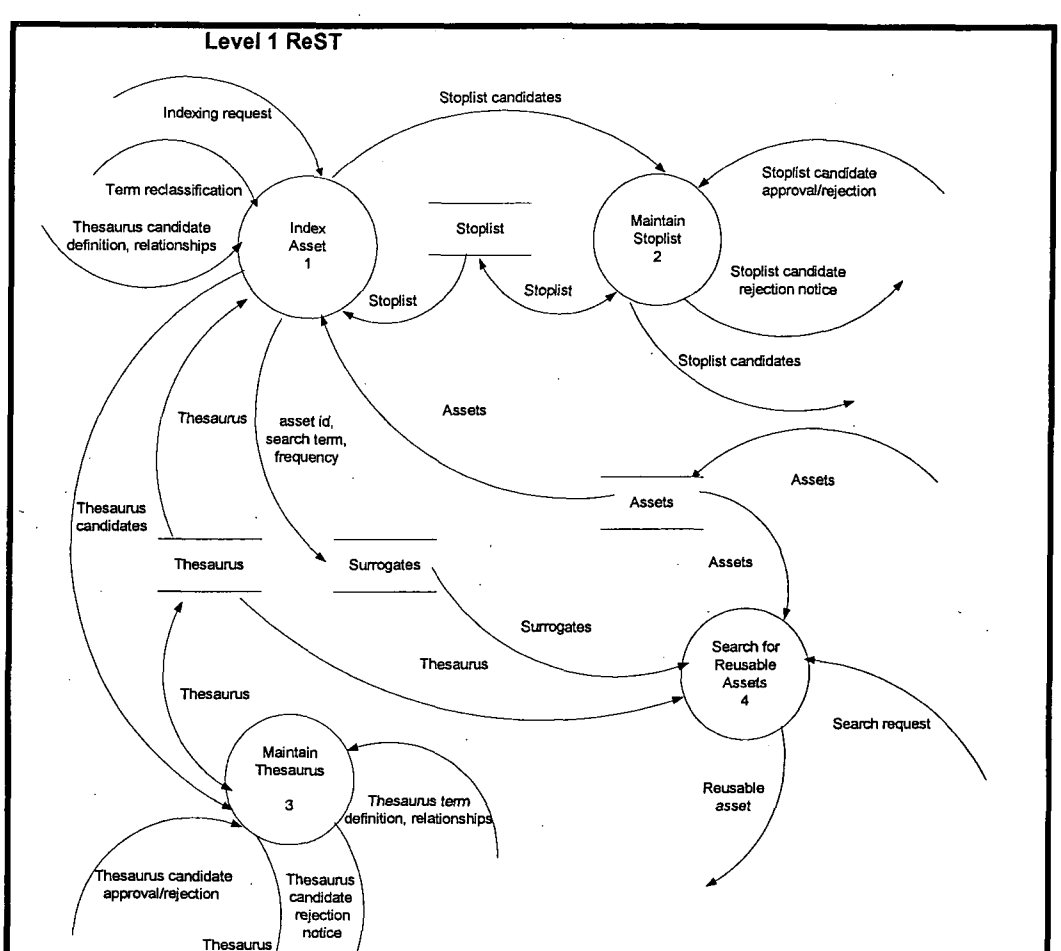

Figure 6.2 Level 1 Dataflow Diagram of ReST

Process one entitled "Index Asset" is the main process used by Reusers to provide the design community with reusable design assets by indexing a design asset and then classifying the index terms. As part of the classification of the index terms the Reuser will nominate index terms as candidates for either the stoplist or the thesaurus. When reclassifying a term as a thesaurus candidate the reuser may propose the term's definition and relationships. "Index Asset" is illustrated in more detail in Figures 6.3, 6.4 and 6.5. Process two entitled "Maintain Stoplist" is used by the Librarian to maintain the stoplist and is illustrated in more detail in Figure 6.6. Process three entitled "Maintain Thesaurus" is used by the Maintainer to maintain the thesaurus and is illustrated in more detail in Figure 6.7. Process four entitled "Search for Reusable Assets" is

used by the Reuser to retrieve reusable design assets and is illustrated in more detail in Figure 6.8.

6.2.1.2.1 Indexing the Asset

The "Index Asset" process is illustrated as a level two dataflow diagram in Figure 6.3 below. The diagram shows the decomposition of "Index Asset", and the internal and external data need for the process.

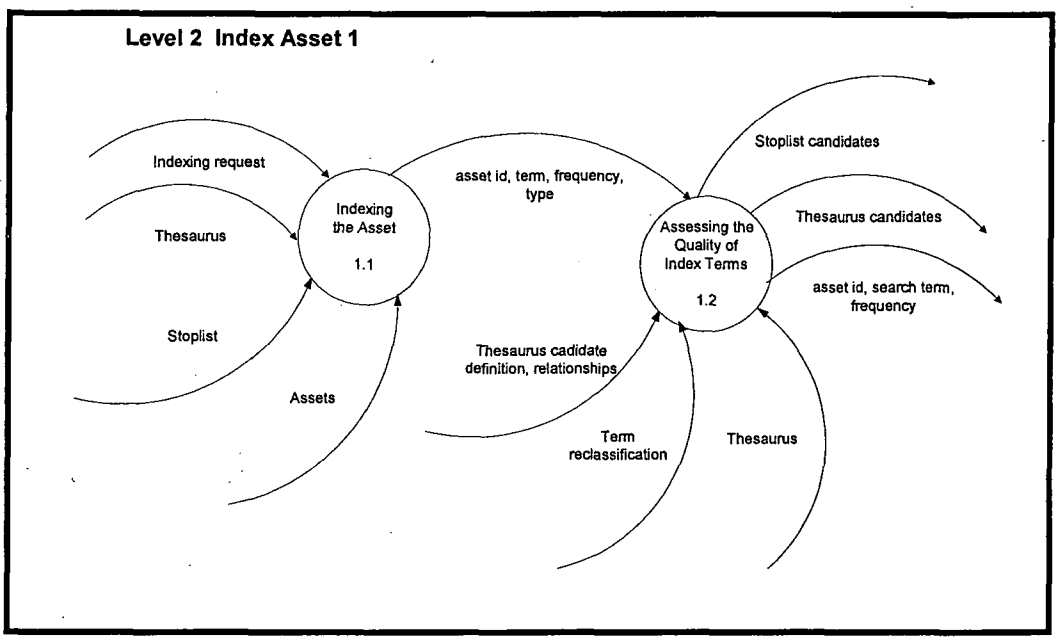

Figure 6.3 Level 2 Dataflow Diagram of Index Asset

A Reuser's indexing request will identify the asset to be indexed. Individual design assets are indexed, to provide a list of unique terms contained in the asset and the frequency with which those terms occur within the asset. The "Indexing the Asset" process then outputs the indexed terms. This process is illustrated in more detail in Figure 6.4. The "Assessing the Quality of Index Terms" takes the indexed terms and automatically classifies each term. A Reuser then explicitly reclassifies each term. When a Reuser reclassifies a term as a thesaurus candidate the Reuser will propose a term definition and relationships. The "Assessing the Quality of Index Terms" process is illustrated in more detail in Figure 6.5.

6.2.1.2.2 Indexing the Asset

The "Indexing the Asset" process is illustrated as a level three dataflow diagram in figure 6.4. The diagram illustrates the decomposition of the process, the internal and external data used during processing, and the data stores that are accessed during processing.

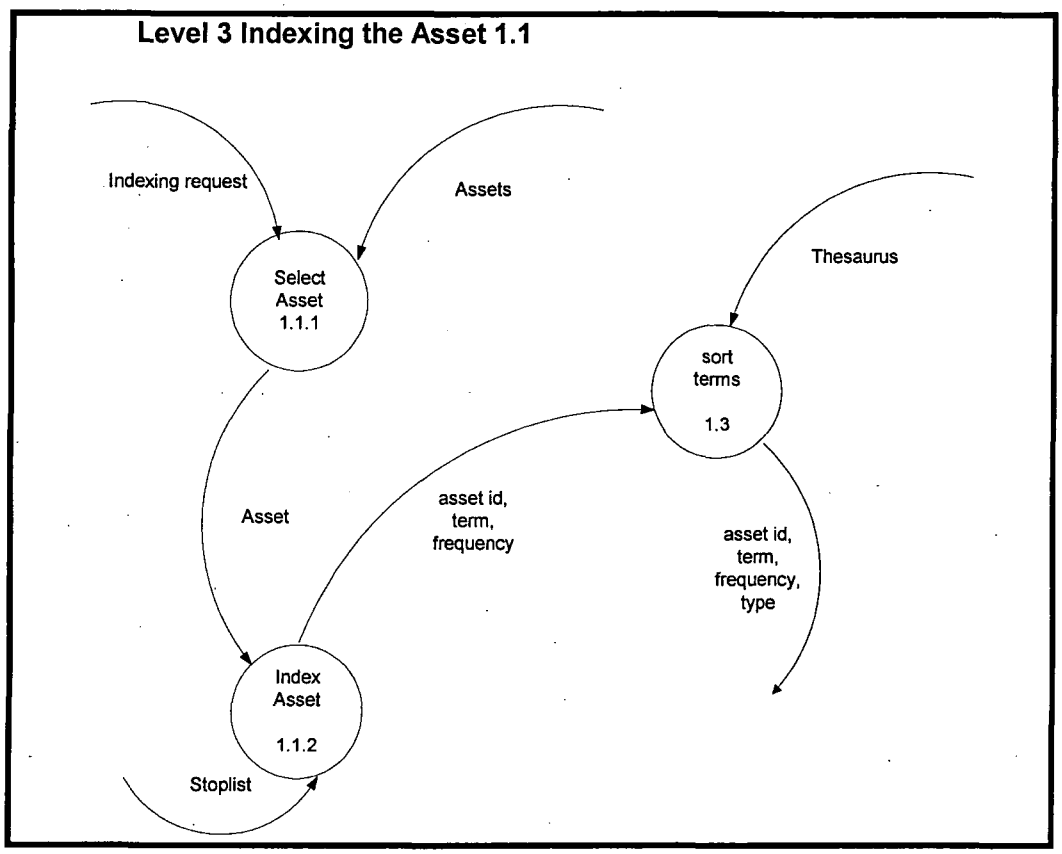

Figure 6.4 Level 3 Dataflow Diagram of Indexing the Asset

The asset to be indexed is selected from the assets store. The asset is then indexed to provide a list of unique terms contained in the asset and the frequency with which those terms occur within the asset. The index terms have the following properties.

- They are comprised of one or more consecutive words.
- They do not begin with any words that exist in the current stoplist.
- They do not end with any words that exist in the current stoplist.
- They do not contain any words that exist in the current stoplist.

The index terms are then sorted into their initial classification or type. The initial classification is carried out automatically by ReST. The index terms will be automatically classified as one and only one of the following three types.

- Preferred terms
- Defined terms
- Undefined terms

An index term classified as a preferred term is a term that has been defined within the thesaurus as a preferred term. An index term classified as a defined term is a term that has been defined within the thesaurus but not as a preferred term. An index term classified as an undefined term is a term that has not been defined within the thesaurus.

6.2.1.2.3 Assessing the Quality of Index Terms

The "Assessing the Quality of Index Terms" process is illustrated as a level three dataflow diagram in figure 6.5. The diagram illustrates the decomposition of the process, the internal and external data used during processing, and the data stores that are accessed during processing.

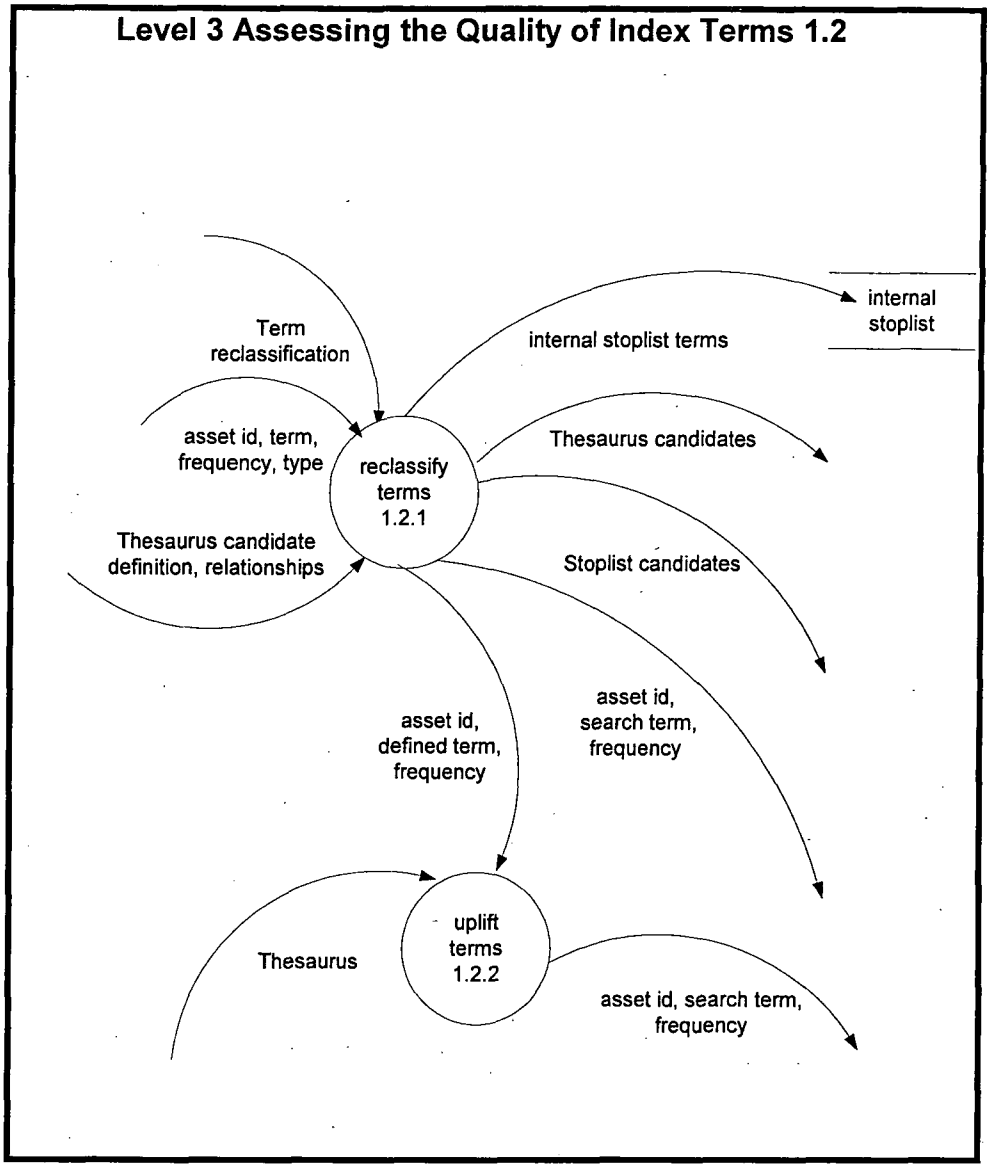

Figure 6.5 Level 3 Dataflow Diagram of Assessing the Quality of Index Terms

Upon completion of the automatic classification a reuser is required to intervene in the reclassification of the index terms. The index terms must be reclassified into one and only one of the four categories or types listed below:

- Internal stoplist term;
- Stoplist candidate;
- Thesaurus candidate; or
- Search term.

An index term classified as an internal stoplist term is a term that has meaning within the domain but is judged not useful when searching for the asset. An index term classified as a stoplist candidate is an index term that a reuser has nominated for addition to the stoplist. An index term classified as a thesaurus candidate is an index term that a reuser has nominated for inclusion in the thesaurus. With this classification the Reuser may enter a proposed definition and relationships for the term. An index term classified as a search term is an index term that is defined in the thesaurus as a preferred term and is judged useful, by the Reuser, when searching for the asset. An index term defined in the thesaurus as something other than a preferred term and judged useful when searching for the asset is also classified as a search term; however, the index term is exchanged for its preferred term before being classified as a search term.

An asset's surrogate is the collection of all the search terms for that asset. When a defined term is reclassified as a search term, the defined term is exchanged for its preferred term before the term is included in the asset's surrogate. This exchange process is known as 'uplifting' the vocabulary.

A term's initial classification provided during the "Indexing the Asset" process restricts the reclassification that the Reuser may perform within ReST. A preferred term may be reclassified as either a search term or as an internal stoplist term. A defined term may be reclassified as either a search term or as an internal stoplist term. An undefined term may be reclassified as an internal stoplist term, or a stoplist candidate, or a thesaurus candidate.

6.2.1.2.4 Maintain Stoplist

The "Maintain Stoplist" process is illustrated as a level two dataflow diagram in figure 6.6. The diagram illustrates the decomposition of the process, the internal and external data used during processing, and the data stores that are accessed during processing.

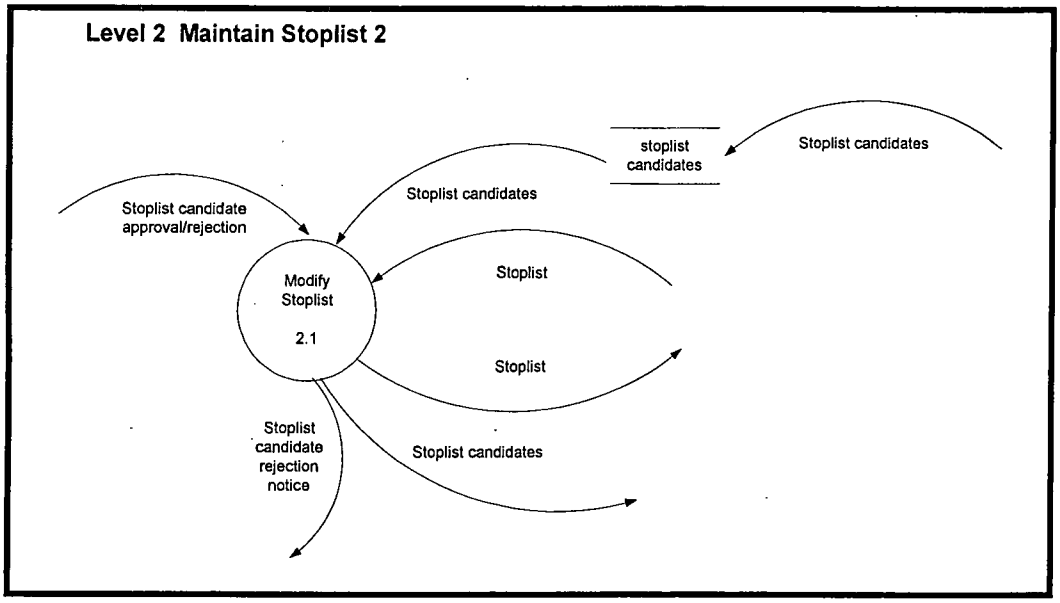

Figure 6.6 Level 2 Dataflow Diagram of Maintain Stoplist

The Librarian uses the "Maintain Stoplist" process to populate and modify the data contained in the stoplist. The stoplist is a store holding a collection of unique words that are meaningful within the context of natural language but have no specific meaning within the domain. Words such as "then" and "that" are examples of stoplist words. The stoplist is populated by index terms found in the design assets. These terms are initially classified as undefined terms in the "Index the Asset" process and then reclassified by a Reuser as stoplist candidates in the "Assess the Quality of Index Terms" process. The Librarian reviews the stoplist candidates and selects or rejects the candidates. The selected stoplist candidates are added to the stoplist. When the Librarian rejects the candidates a candidate rejection notice is sent automatically.

6.2.1.2.5 Maintain Thesaurus

The "Maintain Thesaurus" process is illustrated as a level two dataflow diagram in figure 6.7. The diagram illustrates the decomposition of the process, the internal and external data used during processing, and the data stores that are accessed during processing.

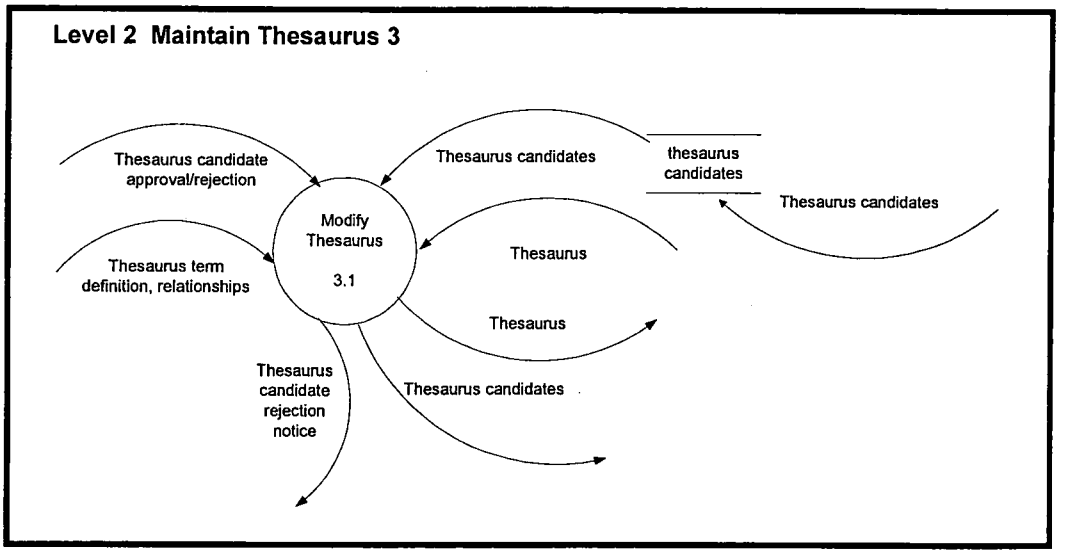

Figure 6.7 Level 2 Dataflow Diagram of Maintain Thesaurus

The thesaurus is the store for the domain specific knowledge found in the design assets. It holds the terminology used within the domain and defines the relationships between the terms in the domain. The Maintainer uses the "Maintain Thesaurus" process to populate and modify the data contained in the thesaurus. The Maintainer must maintain not only the data contained in the thesaurus but also the structure of the thesaurus that is defined by the relationships between the terms contained in the thesaurus. The thesaurus is populated by index terms found in the design assets. These terms are initially classified as undefined terms in the "Index the Asset" process and then reclassified by a Reuser as thesaurus candidates in the "Assess the Quality of Index Terms" process. A Reuser can propose the term's definitions and its relationships when they reclassify the term as a thesaurus candidate. The Maintainer may reject, modify or add the proposed definition and relationships to the thesaurus. A Maintainer may also enter any term definitions or relationships not provided within the thesaurus candidate term. Population of the thesaurus cannot be automated to any useful extent. Direct intervention by a Maintainer is required for data to be entered or modified in the thesaurus. The Maintainer reviews each thesaurus candidate and either selects or rejects the data for entry into the thesaurus. When a Maintainer rejects a thesaurus candidate, an automated candidate rejection notice is output.

6.2.1.2.6 Search for Reusable Assets

The "Search for Reusable Assets" process is illustrated as a level two dataflow diagram in figure 6.8. The diagram illustrates the decomposition of the process, the internal and external data used during processing, and the data stores that are accessed during processing.

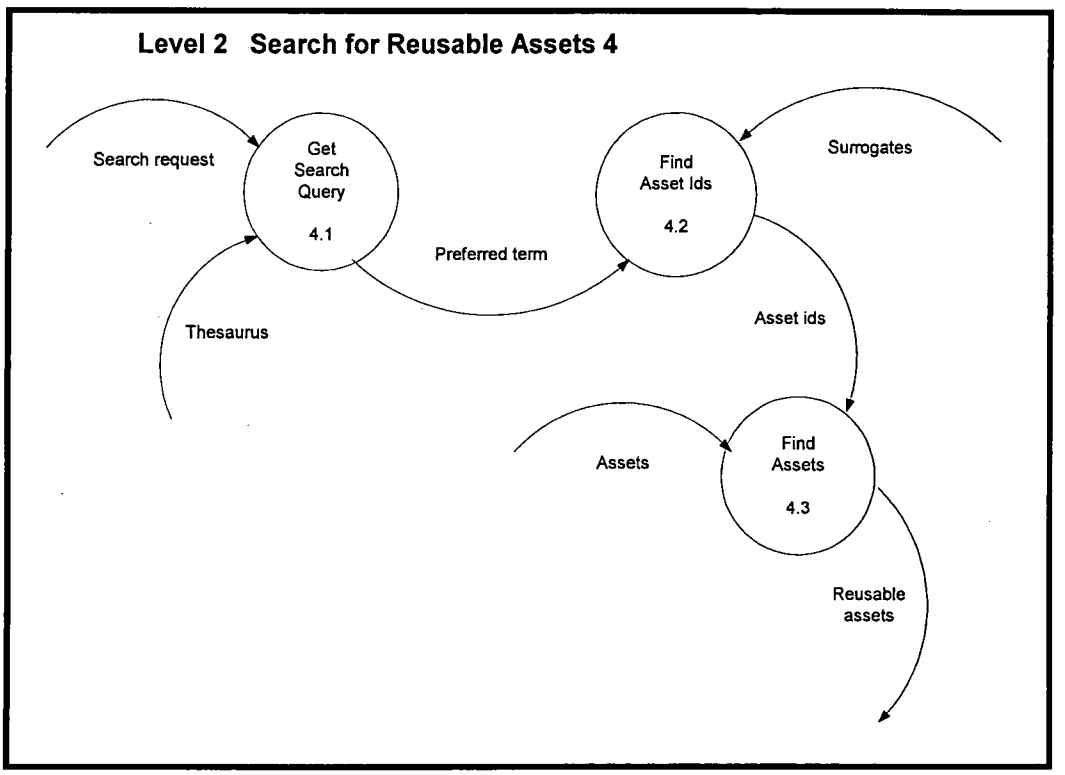

Figure 6.8 Level 2 Dataflow Diagram of Search for Reusable Assets

Each asset has a surrogate that contains a group of terms selected by a reuser as potentially useful search terms. The surrogate also contains the asset's unique identifier. Every term in a surrogate is a term that is defined in the thesaurus as a preferred term. Therefore, search queries used to locate reusable assets' surrogates are comprised of preferred terms only. The thesaurus contained in ReST is available during the definition of a search query to provide a reuser with assistance in locating the preferred term for any term defined in the thesaurus. The intention of the search is to find all the reusable assets in the reuse library that meet the search criteria i.e. contain the search term or terms as defined by the search query. The search through the surrogates will provide the reuser with a list

of reusable assets' unique identifiers. Then the reuser must browse the actual reusable assets to locate precisely the potential asset for reuse.

6.2.2 Entity Relationships

This section contains a series of four entity-relationship (e-r) diagrams. Each diagram is provided to demonstrate the relationships that exist between the entities of the final prototype of ReST. The entities in the e-r diagrams are comprised of the external data entities and the data stores identified in the previous section dataflow diagrams. The relationships are either one to one (1:1) or one-to-many (1:M) or many-to-many (M:N) each relationship is named. Figure 6.9 illustrates the relationships that exist between the Reuser and the other entity within ReST. Figure 6.10 illustrates the relationships that exist between the Librarian and the other entity within ReST. Figure 6.11illustrates the relationships that exist between the Maintainer and the other entity within ReST. Figure 6.12 brings all three of the previous e-r diagrams together to illustrate the relationships that exist between all the entities in ReST.

6.2.2.1 Reuser's Relationships

Figure 6.9 is an entity relationship diagram illustrating the relationships that exist between the Reuser and the other entities within ReST. Entities within ReST that are not related to the Reuser are not shown in this e-r diagram.

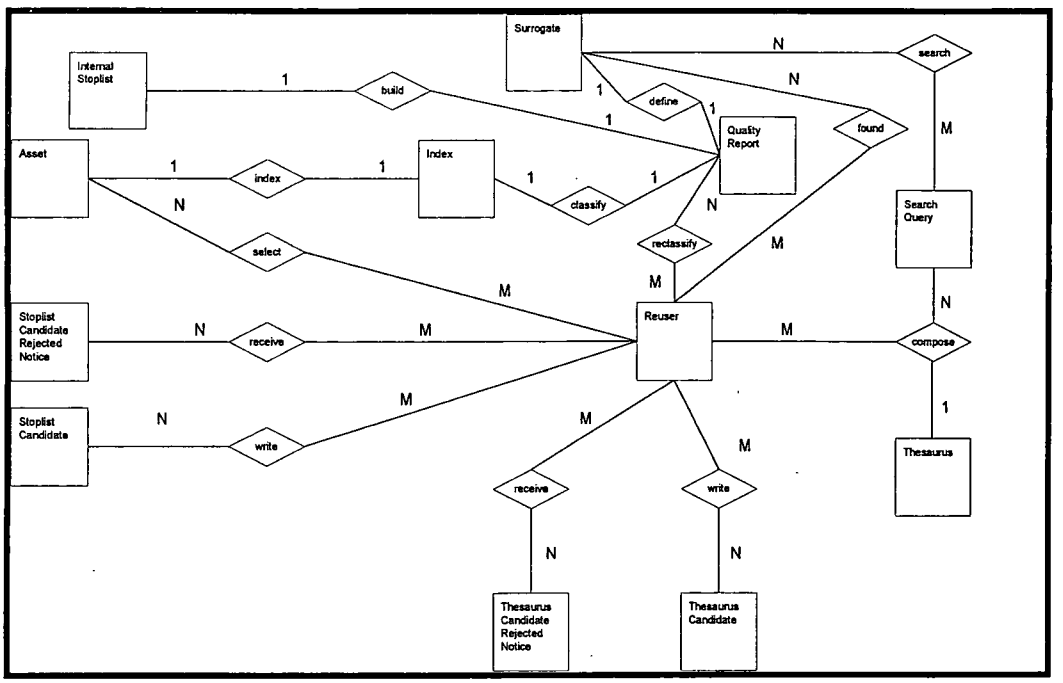

Figure 6.9 Reuser's Entity-Relationship Diagram

Though there can be any number of assets, each asset can have only one index. An index is classified to provide its quality report (the index terms and the term classifications). An index can have only one quality report. The Reuser can reclassify the terms in a quality report. This reclassification is used to construct the internal stoplist (a sub-set of quality report terms) and to define the surrogate (a sub-set of quality report terms). There can be only one internal stoplist for a quality report. There can be only one surrogate for a quality report. It is therefore possible to deduce that an asset can have associated with it only one index, one surrogate, one internal stoplist, and one quality report.

The Reuser compiles and sends both thesaurus candidates and stoplist candidates. The Reuser receives both thesaurus candidate rejection notices and stoplist candidate rejection notice. These rejection notices are only received when a reuser's candidate has been rejected. There is no notice sent when a candidate has been approved.

To locate the surrogates of potentially reusable assets the Reuser must compose a search query. When composing search queries a Reuser may use the thesaurus to overcome any vocabulary difficulties that they may encounter. The search of the

surrogates is restricted to the definition of the search query i.e. the search result will be only those surrogates that meet the search criteria as defined by the search query. The search results are returned to the Reuser.

6.2.2.2 Librarian's Relationships

Figure 6.10 is an entity relationship diagram illustrating the relationships that exist between the Librarian and the other entities within ReST. Entities within ReST that are not related to the Librarian are not shown in this e-r diagram.

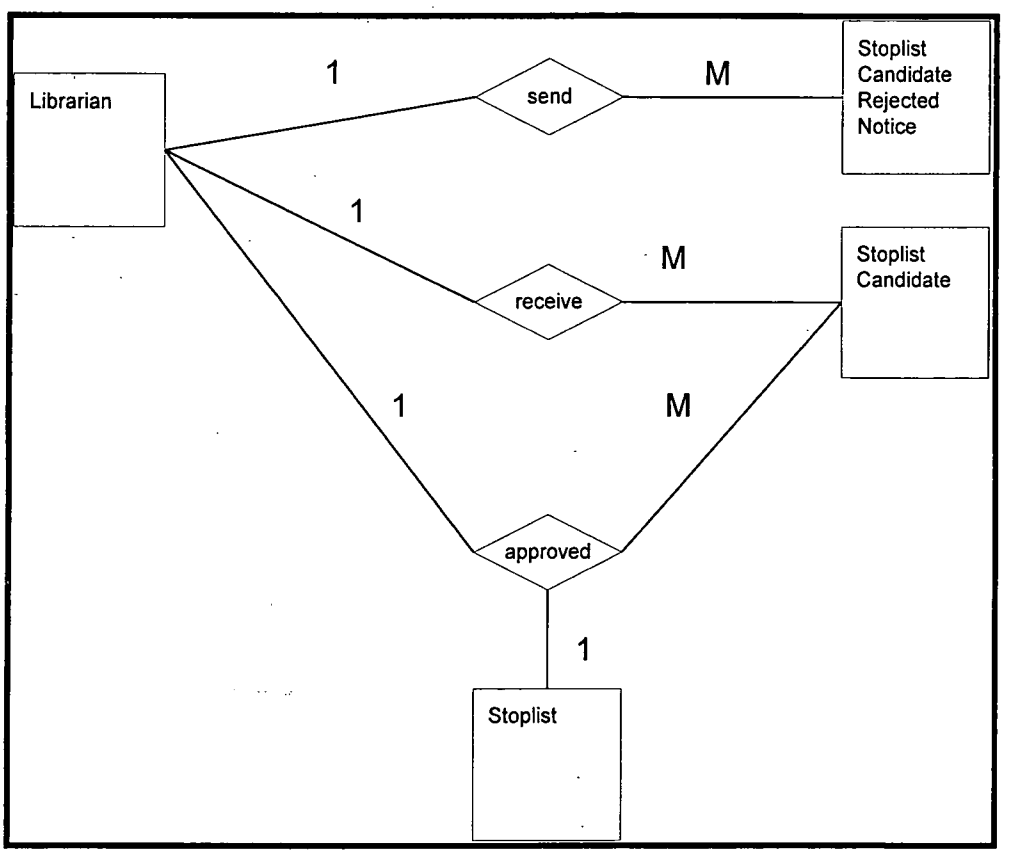

Figure 6.10 Librarian's Entity-Relationship Diagram

Figure 6.10 identifies entities that have a relationship with the Librarian. The Librarian is responsible for the maintenance of the Stoplist. The Librarian receives stoplist candidates. If the Librarian elects to reject the candidate then the Librarian sends a stoplist candidate rejected notice. If the Librarian approves the candidate the stoplist is modified accordingly. It should be noted that there is only ever one stoplist.

6.2.2.3 Maintainer's Relationships

Figure 6.11 is an entity relationship diagram illustrating the relationships that exist between the Maintainer and the other entities within ReST. Entities within ReST that are not related to the Maintainer are not shown in this e-r diagram.

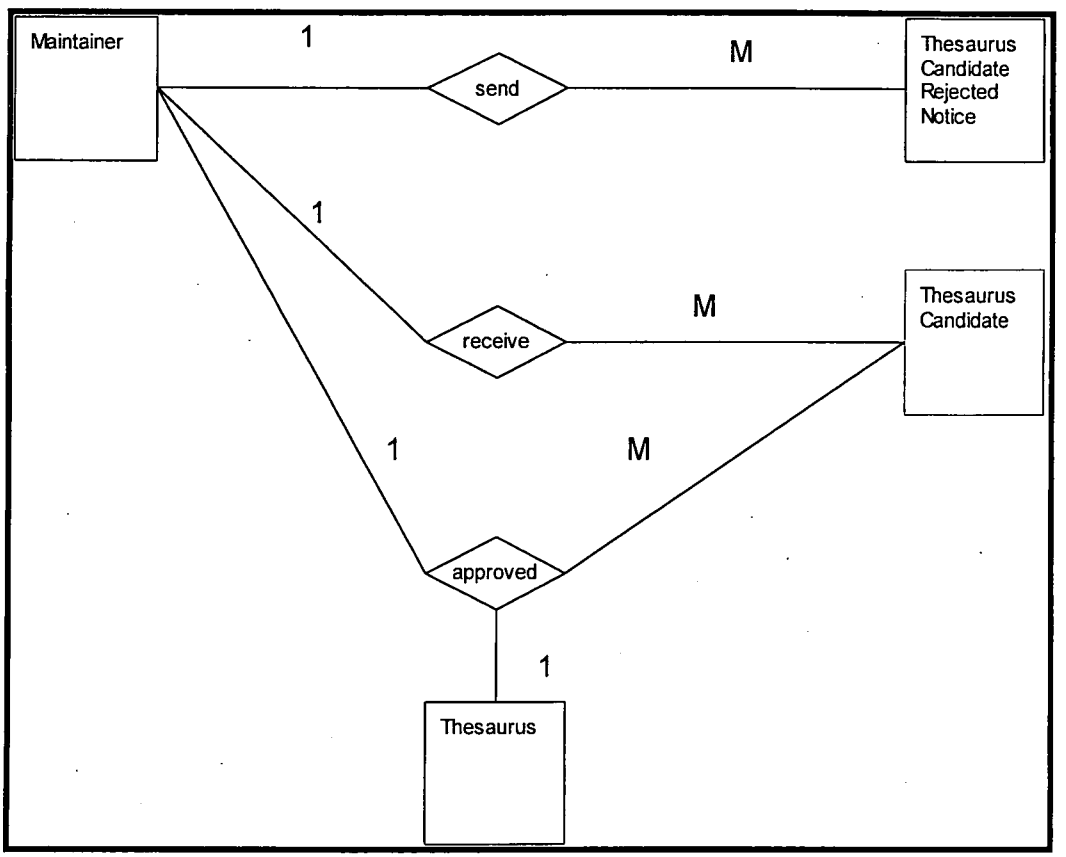

Figure 6.11 Maintainer's Entity-Relationship Diagram

Figure 6.11 identifies entities that have a relationship with the Maintainer. The Maintainer is responsible for the maintenance of the Thesaurus. The Maintainer receives thesaurus candidates. If the Maintainer elects to reject the candidate then the Maintainer sends a thesaurus candidate rejected notice. If the Maintainer approves the candidate, the thesaurus is modified accordingly. It should be noted that there is only ever one thesaurus.

6.2.2.4 All Entity Relationships for ReST

Figure 6.12 is an assemblage of Figures 6.9, 6.10 and 6.11there are no additional entities or relationships and no entities or relationships have been removed. Figure 6.12 has been added for completeness and to illustrate all the entities connected with ReST and the relationships that exist between them.

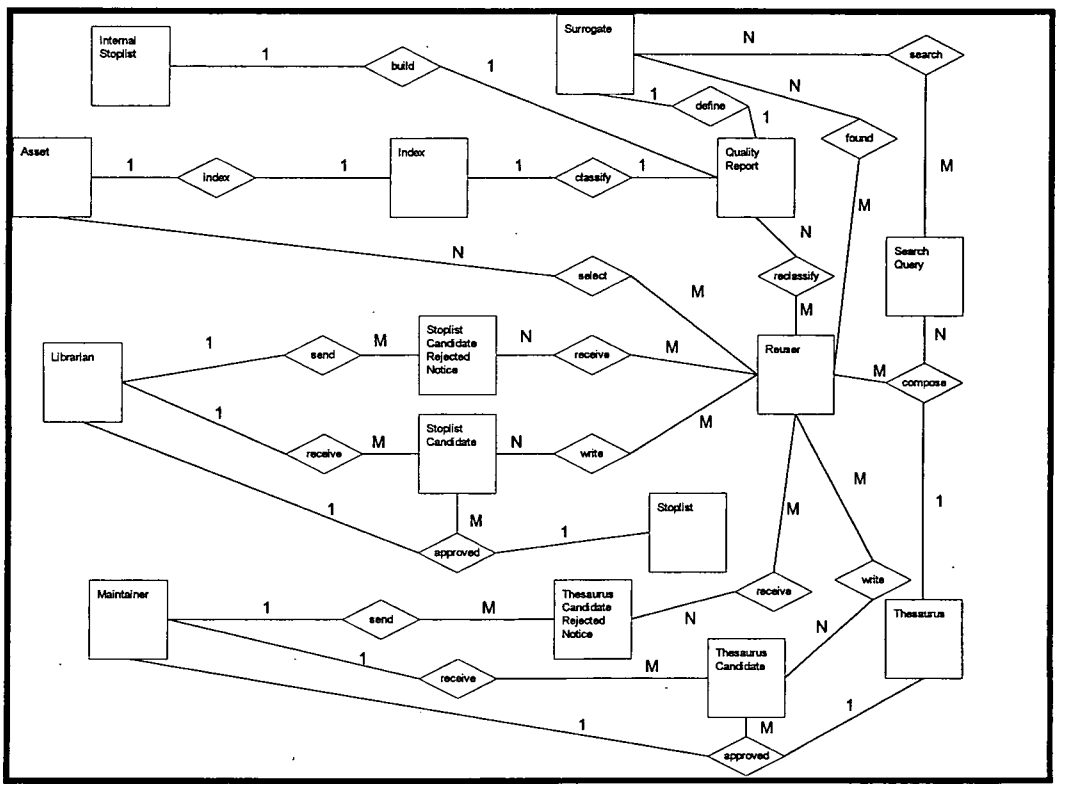

Figure 6.12 Entity Relationship Diagram for ReST

6.3 Validation of Design against Requirements

This section provides the validation of the design presented in this chapter. The design is shown to satisfy the functional requirements presented in Section 5.5.1.1 of Chapter 5 of this document. Listed below are each of the functional requirements and the number of the design section(s) that contains the design relating to the requirement.

Requirement	Design Section Number
• Index a design asset providing a list of unique terms and the frequency with which those terms occur in the asset.	6.2.1.2.1 and 6.2.1.2.2
• Classify each term contained in the index as either a preferred term, or a term defined in the thesaurus, or a term not in the thesaurus, or an internal stoplist term, or a stoplist candidate term, or a candidate term for inclusion in the thesaurus, or a search term, that is included in an asset's surrogate.	6.2.1.2.1, 6.2.1.2.2 and 6.2.1.2.3
• Construct surrogates using preferred terms only.	6.2.1.2.3
• Construct search queries using preferred terms only.	6.2.1.2.6
• Populate the thesaurus with terms found in the assets.	6.2.1.2.3 and 6.2.1.2.5
• Maintain the thesaurus and the stoplist.	6.2.1.2.4 and 6.2.1.2.5

6.4 Summary

This chapter contains the design details of the final prototype of ReST (Reuse Support Tool). This includes a series of dataflow diagrams with textual descriptions explaining the data and the processes in the final prototype of ReST and a series of entity-relationship diagrams with textual descriptions explaining the relationships between the entities in the final prototype of ReST. There are three external entities: the Reuser, the Librarian, and the Maintainer. All three external entities are British Steel roll designers. However each entity performs specific tasks when interacting with ReST. There are four main processes: Index Assets performed by the Reusers; Maintain Stoplist performed by the Librarian; Maintain Thesaurus performed by the Maintainer; and Search for Reusable Assets performed by the Reusers.

Chapter 7 Implementation of ReST

7.1 Objectives of the Chapter

The main objective of this chapter is to provide the implementation details of the final prototype of ReST designed in Chapter 6.

Section 7.2 provides the description of the final implementation of ReST and includes examples of the user interface and some sample data. Section 7.3 provides the validation of the implementation against the functional requirements stated in Chapter 5 Section 5.5.1.1. Section 7.4 provides the summary of this chapter.

7.2 Implementation of the final prototype of ReST

The final prototype of ReST is a functioning prototype to demonstrate the proposed solution to the domain problem stated in Chapter 4. ReST was implemented using Microsoft Access 97. Microsoft Access 97 provides a user interface that is relatively easy to define, tables for data storage, data manipulation functions and its own version of Visual Basic for developing the more complex functionality of the prototype.

The user interface for ReST was constructed using Access forms that are linked by commands initiated by the user of ReST. Data is stored in Access tables and manipulated through a series of commands entered by the user. Access contains some pre-constructed functionality such as functions to find or delete records that is exploited in ReST. The remaining functionality is contained in a group of related modules and coded in the version of Visual Basic available in Access. Functions such as the indexing of assets and assessing the quality of the index terms are written in Visual Basic. Access is capable of accepting output generated by Perl scripts and able to issue command line instructions. These features were utilised to provide the initial data included in the stoplist. The opening screen display for the user interface for ReST is illustrated in Figure 7.1.

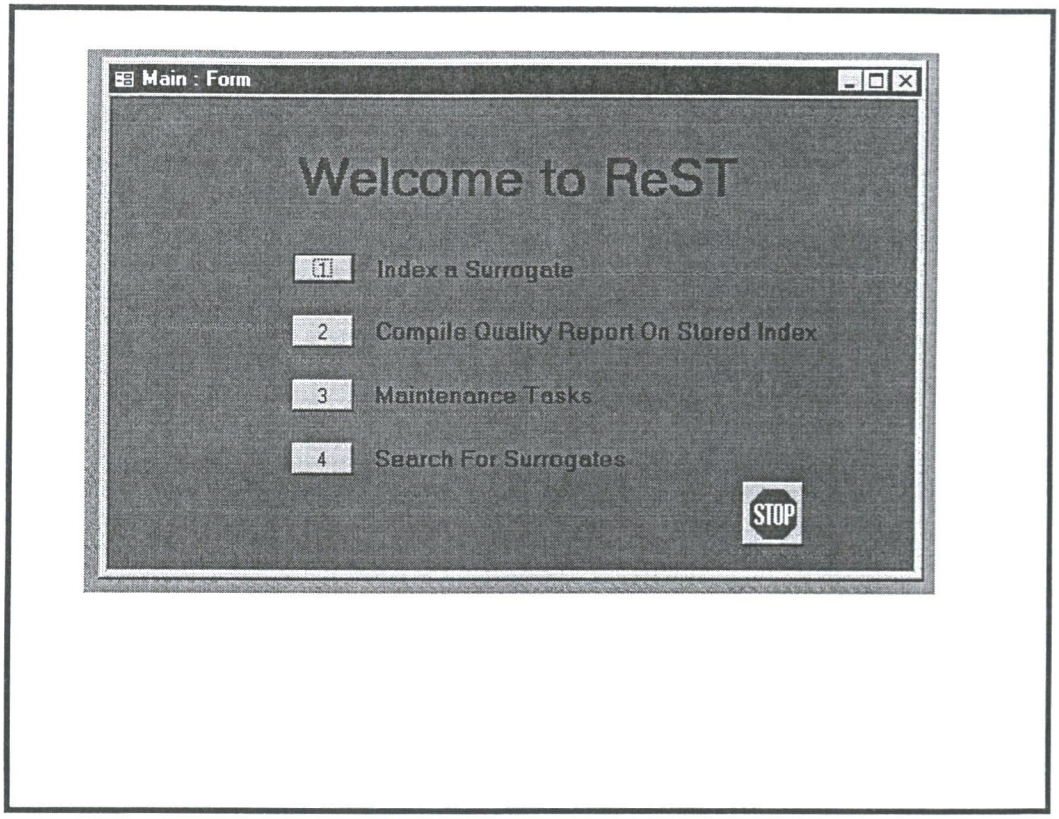

Figure 7.1 Opening Screen for ReST

All areas of ReST are open to every user. To index an asset, a Reuser selects 1, "Index an Asset". The Reuser is then asked to login and selects the asset for indexing. ReST automatically indexes the asset. After an asset has been indexed the index of terms can be saved for later use, deleted, or classified. Indexing an asset is explained more fully in section 7.2.1. Though not explicitly stated in the design of ReST, the need for a Reuser's id is implied in the design contained in Chapter 6. A Reuser's id is necessary if automated e-mails containing thesaurus and stoplist candidate rejection notices are to be sent.

To classify a saved index a Reuser selects 2, "Compile Quality Report on Stored Index". The Reuser is then asked to login and selects the asset's index for classification. Classifying the index terms is explained more fully in section 7.2.2.

To review the contents of the stoplist or the thesaurus the Reuser selects 3, "Maintenance Tasks".

To review the stoplist candidates and modify the stoplist a Librarian selects 3, "Maintenance Tasks". Modifying the stoplist is explained more fully in section 7.2.3.

To review the thesaurus candidates and modify the thesaurus a Maintainer selects 3, "Maintenance Tasks". Modifying the thesaurus is explained more fully in section 7.2.4.

To search for assets for reuse the Reuser selects 4, "Search for Surrogates". The reuser defines the search query and is shown the asset ids of potentially reusable assets. Searching for reusable assets is explained more fully in section 7.2.5.

7.2.1 Indexing an Asset

The indexing function is written in Visual Basic code and is called from the user interface. An asset is selected and indexing occurs automatically. For trials during implementation two small text files were used. These files are contained in Appendix B. A single asset's index is held in the index table at any one time. The data in the index table is cleared at the end of every work session. The index table holds the asset's unique identifier, the term, the term's frequency, the date of indexing, and the identifier of the reuser who initiated the indexing of the asset. The Reuser is presented with a subset of the fields in the table when reviewing the index. The Reuser is presented with the asset's id, the index term, and the frequency count. A sample of the Reuser's view of the index table is shown below in Figure 7.2. This sample contains only a subset of the data that was compiled when asset "SML101-ts" was indexed.

Document Id	Term	Frequency
SML101-ts	bstp	1
SML101-ts	centre line	1
SML101-ts	diagonal	1
SML101-ts	dimensions	3
SML101-ts	elongation	1
SML101-ts	expansion	1
SML101-ts	fillet	1
SML101-ts	finishing pass	1
SML101-ts	finishing pass profile	1
SML101-ts	foot line	2
SML101-ts	gap	1

Figure 7.2 Sample of Index Data

An asset's index can contain a large number of terms. It was therefore necessary to demonstrate the need for functionality and storage space for indices that are still to be assessed. Saved indices are stored in a single table, and contain the same data as the index table. Unlike the index table which holds the index for only one table the saved index table can hold zero or more asset's indices.

In indexing an asset the asset's text is parsed one word at a time. If the word is contained in the stoplist the word is ignored. Once a word that is not in the stoplist is found it is added as the first word of a new term. Words from the asset are added to the term until a word contained in the stoplist is found. Then the stoplist word is ignored; the term is considered complete and added to the index. This procedure continues until the end of the asset.

It is intended that ReST be used to demonstrate that population of the reuse library, the stoplist and the thesaurus can occur concurrently. However, index terms are phrases (a group of one or more words) contained in the asset that do not contain any stoplist words. Therefore, it was necessary to populate the stoplist with some words prior to developing the indexing functionality. To populate the stoplist a Perl script written by a colleague[14] was adapted to provide data for the stoplist. A copy of this Perl script is contained in Appendix A. Design assets were indexed using the Perl script. This provided a list of unique words and the number of times each unique word occurred within the asset. From the index list words with high frequency and no specific meaning in the domain were selected

and entered into the stoplist. Once the stoplist numbered over one hundred and fifty words it was possible to index an asset for terms that would have meaning within the domain that could be useful input data for demonstrating the quality assessment procedure developed for ReST. Figure 7.3 contains a sample of the words contained in the stoplist. During reclassification of index terms a Reuser can nominate new terms for inclusion in the stoplist. These candidates are held in the stoplist candidate table. The Librarian will review the stoplist candidate table and approve or reject each candidate. If the stoplist candidate is rejected, the reuser that generated the stoplist candidate record should be notified of the rejection via e-mail. The automating of the rejection e-mail has not been implemented. A small sample of the data contained in the stoplist is presented below.

Stoplist term
a
above
account
across
ad
add
address
adequately

Figure 7.3 Sample of Stoplist Data

7.2.2 Assessing the Quality of the Index Terms

The terms in the index table can be assessed immediately after indexing, or the index can be stored then assessed at some later time. To assess the quality of the terms contained in an asset, each term is given a classification and the total number of terms in each classification is calculated. ReST performs the initial classification of the index terms automatically. Each term is classified into one and only one of three categories: Preferred Terms, Defined Terms and Undefined Terms. The Reuser is presented with an Access form displaying the total number of terms for each classification. For the Reuser to reclassify any or all of the terms the Reuser must elect to display the terms in each classification separately and manually select the reclassification category. Figure 7.4 contains a sample of

[14] James Ingham, Department of Computer Science, University of Durham

data from asset SML101-ts (see Appendix B) is presented below. This sample contains those terms that were initially classified as preferred terms. Figure 7.5 shows the form view of the same data.

Preferred Term	Frequency
elongation	1
finishing pass	1
head	1
open flange	2
spread	1

Figure 7.4 Sample of Index Terms Classified as Preferred Terms

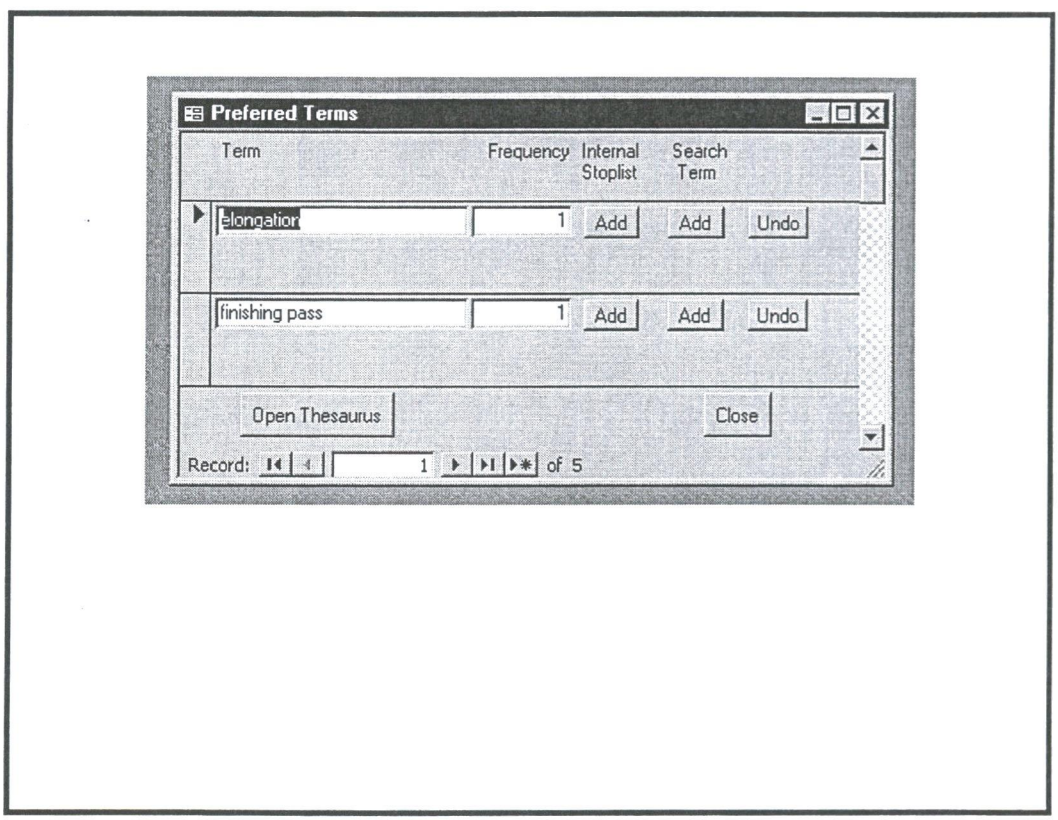

Figure 7.5 A Reuser's View of the Preferred Terms for SML101-ts

ReST was developed using only two small text documents to represent design assets. It was not therefore necessary to construct and file separate surrogates for each "asset". The concept of an asset's surrogates is represented in ReST as an Access table structure. The table contains records consisting of three fields: the name of the assets the surrogate is for, the search term, and the frequency count

for the search term. Figure 7.6 shows examples of the data contained in the search table.

Surrogate For	Term	Frequency
Designing Primary Rolls	depth	18
Designing Primary Rolls	edging	11
Designing Primary Rolls	elongation	2
Designing Primary Rolls	finishing pass	3
Designing Primary Rolls	web	16
SML101-ts	finishing pass	1
SML101-ts	open flange	2

Figure 7.6 Sample of Surrogate Representation (Search Table)

Note that the index terms classified as preferred terms "finishing pass" and "open flange" have been reclassified as search terms and comprise the surrogate for an asset uniquely identified as "SML101-ts".

7.2.3 Maintaining the Stoplist

Reusers nominate candidates for inclusion in the stoplist. The Librarian reviews the stoplist candidates and either accepts or rejects the candidates. When a candidate is rejected, the Reuser that sent the nomination is to be notified automatically via e-mail. This functionality has not been implemented. Figure 7.7 shows the screen form the Librarian uses to review, approve and reject candidates for inclusion in the stoplist.

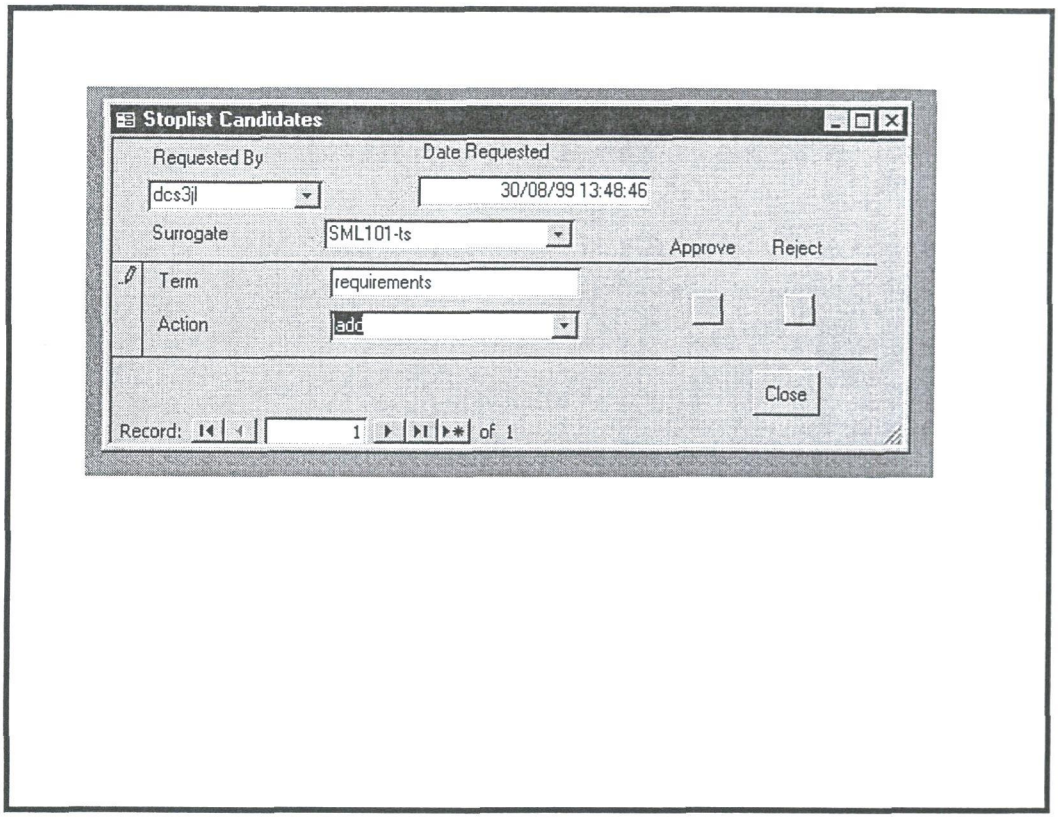

Figure 7.7 Stoplist Candidate Form

7.2.4 Maintaining the Thesaurus

It is intended that the thesaurus be a multi-faceted thesaurus holding domain specific knowledge. It is also intended that terms contained in the design assets be used to populate the thesaurus. Population of the thesaurus requires not only the definition of the terms but also the definition of any relationships between the terms. An Access table is used to store the terms, their definitions and their relationships with other terms held in the thesaurus. The population of the thesaurus cannot be automated to any useful extent. Direct intervention by a Maintainer is required for data to be entered or modified in the thesaurus. Records contain the data pertaining to a single unique preferred term. Index terms that have been reclassified by a Reuser as thesaurus candidate terms are stored in a thesaurus candidate table. The Maintainer reviews the thesaurus candidates and selects the data for entry into the thesaurus. The Maintainer can modify the data in the thesaurus to enhance the data held in the thesaurus candidate records. For instance, a Maintainer can define an additional relationship. The Maintainer can reject the candidate. If the thesaurus candidate is rejected the Reuser that

generated the thesaurus candidate record should be notified of the rejection via e-mail. The automating of the rejection e-mail has not been implemented. The definition of the fields that make-up a thesaurus record is presented in Figure 7.8.

Field	Definition
Preferred Term	A unique term that is accepted as an industry standard by the reuser community.
Definition	A text only definition of the preferred term.
Scope Note	A text area intended to hold additional definitions of the preferred term, extra information on the terms in the thesaurus record or the source of information in the record.
Broader Terms	Terms that have a broader definition than the preferred term and are in the same classification facet.
Narrower Terms	Terms that have a narrower and more precise definition than the preferred term and are in the same classification facet.
Related Term	Terms that are related to the preferred term in the context of the domain but are not suitable for inclusion in any other field in the record.
Scunthorpe Term	The term that has the same meaning as the preferred term but are unique to the Scunthorpe mill.
Teeside Term	The term that has the same meaning as the preferred term but are unique to the Teeside mill.
Top Term	The term that has the broadest definition within the classification facet that the preferred term is in.
Bottom Term	The term that has the narrowest definition within the classification facet that the preferred term is in.
Date Created	The date the record was created.
Created By	The unique id for the maintainer who created the record.
Date Modified	The dates the record was modified.
Modified By	The unique id for the maintainers who modified the record.
In Use	A Boolean with a value of yes or no. If the field contains yes, then the preferred term is currently in use within the domain. If the field contains no, the preferred term is not currently in use within the domain but is a preferred term in historical assets and is therefore still available for use when searching for reusable assets.

Figure 7.8 Thesaurus Record Definition

Figure 7.9 presents a single record from the Thesaurus in ReST. Notice that not all fields are completed. The records will become complete as understanding in the domain matures.

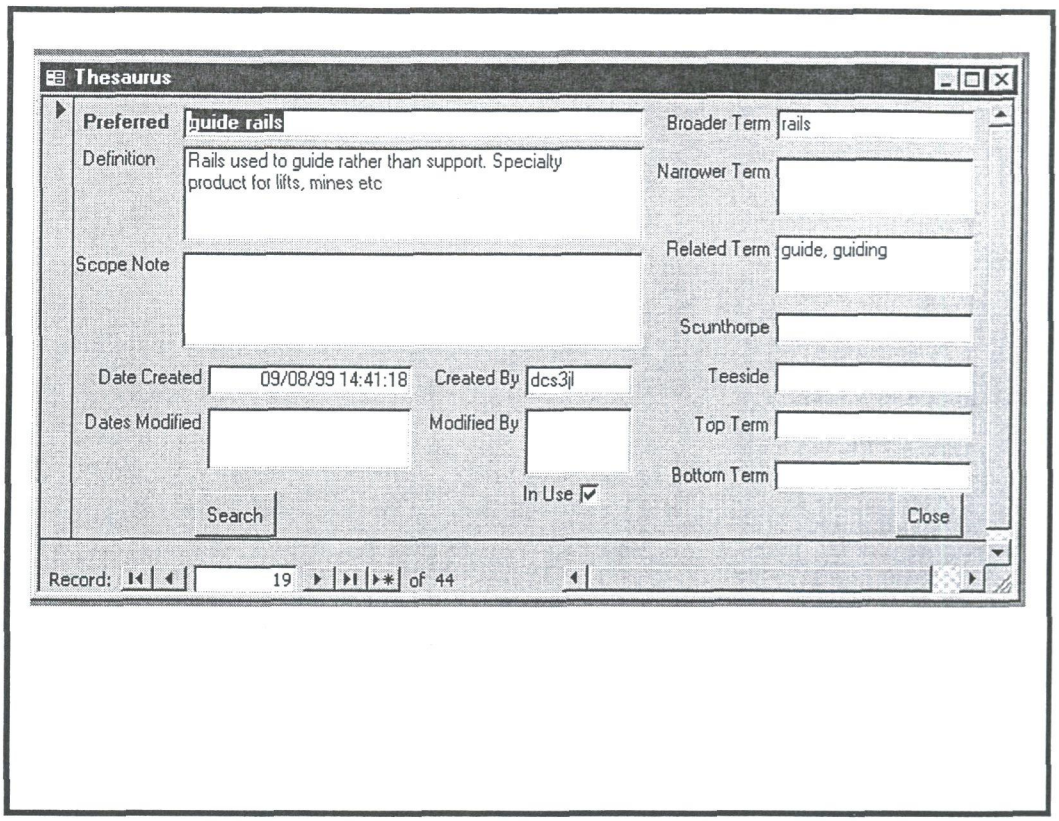

Figure 7.9 A Thesaurus Record

7.2.5 Retrieving Reusable Assets

ReST was developed using only two small text documents to represent design assets. It was not therefore necessary to construct and file separate surrogates for each "asset". The concept of an asset's surrogates is represented in ReST as an Access table structure. The table contains records consisting of three fields: the name of the assets the surrogate is for, the search term, and the frequency count for the search term. The search terms contain only terms that have been defined in the thesaurus as preferred terms. Therefore, search queries contain only terms that are preferred terms. A query will result in the presentation of a search table record, which provides the asset's unique identifier. At this time there is no automated link to the actual asset.

7.3 Validation of Implementation against Requirements

This section provides the validation of the implementation presented in this chapter. The implementation is shown to satisfy the functional requirements presented in Section 5.5.1.1 of Chapter 5 of this document. Listed below are each

of the functional requirements and the number of the implementation section that contains the implementation details relating to the requirement.

Requirement	Implementation Section Number
• Index a design asset providing a list of unique terms and the frequency with which those terms occur in the asset.	7.2.1
• Classify each term contained in the index as either a preferred term, or a term defined in the thesaurus, or a term not in the thesaurus, or an internal stoplist term, or a stoplist candidate term, or a candidate term for inclusion in the thesaurus, or a search term, that is included in an asset's surrogate.	7.2.2
• Construct surrogates using preferred terms only.	7.2.2
• Construct search queries using preferred terms only.	7.2.5
• Populate the thesaurus with terms found in the assets.	7.2.4
• Maintain the thesaurus and the stoplist.	7.2.3 and 7.2.4

7.4 Summary

This chapter contained the implementation details of the final prototype of ReST. The final prototype has been implemented as a Microsoft Access 97 database. The implementation meets the requirements stated in Chapter 5 and was developed from the design presented in Chapter 6.

Chapter 8 Testing and Evaluation of ReST

8.1 Objectives of the Chapter

This chapter provides the details surrounding the testing and evaluation of the final prototype of ReST (Reuse Support Tool). The functionality of ReST is to be tested against the tasks listed in the scenario in Figure 8.1 to check that ReST performs as expected. The scenario will be used again in a formal demonstration of ReST. The formal demonstration allows ReST to be evaluated against the expectations and needs of the potential user community. The results of the testing and evaluation are presented in this chapter. Each activity in the scenario in Figure 8.1 is evaluated separately.

Section 8.2 provides the status of the prototype prior to testing. Section 8.3 provides an overview of the testing procedure that includes how the prototype has been tested and used in a formal demonstration. Section 8.4 provides the details of the test results including the discussion generated during the formal demonstration of the prototype. The result for each activity in the scenario is discussed separately. Section 8.5 provides the summary of this chapter.

8.2 Prototype Status

This section provides details of the status of the final prototype of ReST prior to testing. This includes the details on the stoplist, the thesaurus, the sample data used in the development and testing of the prototype, and the search functionality.

8.2.1 Stoplist

The stoplist used during indexing of assets contains three hundred and thirty-three terms. The stoplist was developed during the development of the prototype. The terms contained in the stoplist can be found in the two sample data files discussed in section 8.2.3. The stoplist was compiled without the help of a domain expert. Commonly understood knowledge about the natural language was used to compile the stoplist.

8.2.2 Thesaurus

The thesaurus contains forty-four preferred terms. Each preferred term has a definition and defined relationships to other terms. The preferred terms are not necessarily related to other preferred terms. They are sometimes related to terms that have no explicit definition in the thesaurus. The content of the thesaurus was developed for demonstration purposes only and has not been reviewed by a domain expert. The terms contained in the thesaurus were selected during the development of the prototype. The preferred terms in the thesaurus can be found in the two sample data files discussed in section 8.2.3. The term selection was for the most part random. However, any term defined in the thesaurus as a preferred term can be found in the "Glossary of Roll Design Terms" [BS97] (glossary). The preferred terms' definitions were taken from the glossary. There is one exception, the preferred term "depth", which was intentionally added to the thesaurus to demonstrate the equivalence relationship unique to the domain, that of the relationship between mill specific terms. The definition for the term "depth" was not found in the glossary.

8.2.3 Sample Data Files

There are two sample data files. Both files are text only files, were used in the development of the initial and final prototype of ReST, and can be found in Appendix B. The first sample data file contained excerpts from the high-level design document "Notes on Designing Primary Rolls with One Beam Shape Forming Pass" [ORD99]. The excerpts comprised a sample data file that was four pages in length, contained one hundred and ninety-one lines of text and one thousand four hundred and thirty-six words. The second sample data file contained excerpts from the high-level design document "Expert Roll Design" [SML98]. The excerpts comprised a sample data file that was one page in length, contained fifty-four lines of text and three hundred and seven words. The second file was used in the demonstration of the initial prototype of ReST and in the demonstration and testing of the final prototype of ReST.

8.2.4 Search Functionality

The search functionality contained in the final prototype of ReST was provided for the formal demonstration. The emphasis in the demonstration was on how to locate potentially reusable assets via their surrogates and how the thesaurus could be used to help reusers define search queries. The search queries were constrained to a single term each. The terms used in the demonstration were terms that are defined as preferred terms in the thesaurus and known to be included in one or more of the test surrogates. There was no domain expertise applied to defining the search queries.

There was an insufficient pool of test documents to enable testing of precision, recall, overlap, or relevance to the search goal. The premise that restricting the contents of the surrogate to preferred terms would increase recall at the cost of precision was not tested.

8.3 Overview of the Testing Process

The final prototype for ReST was developed as a functioning prototype to be used to demonstrate proposed software tool support that was intended to support the practice of component-based reuse and domain analysis concurrently. In the context of this work, domain analysis is the development of a domain specific thesaurus and the ongoing population and maintenance of the thesaurus. Testing of ReST consisted of enacting the work activities presented in the scenario in Figure 8.1.

Action/Tasks	Result	Comments
Index document excerpt.	Breaks the document into 47 unique terms.	
Perform the initial **quality assessment** of the indexed terms.	5 preferred terms 9 defined terms 33 undefined terms	Automatic classification can only result in these three categories
Reclassify the terms manually.	0 preferred terms 0 defined terms 31 undefined terms 9 internal stoplist terms 1 stoplist candidate 1 thesaurus candidate 3 search terms (these become the Surrogate)	In actual use undefined terms should equal 0. Note that because search terms are uplifted to preferred terms the overall number of terms is reduced
Submit the quality report.	Now able to view document history.	Could be used to assess the institutionalising of the standard terminology and the usefulness of the thesaurus
Review Stoplist Candidate and reject proposed addition.	Term is deleted. No addition to the stoplist.	Automatic e-mail messaging should be implemented.
Review Thesaurus. Search for preferred term "depth"	The term, with definition and various relationships are displayed.	
Search surrogates containing the preferred term for "diagonal"	The thesaurus is used to find the preferred term "diagonal rolling" which is then used to search the surrogates. The document id "Primary Roll Design" is found.	Good recall but may be at the cost of precision.
Search surrogates for the term "finishing pass"	Two document ids, "SML101-ts" and "Primary Roll Design" are found.	

Figure 8.1 Scenario of Work Activities Designed to Test Prototype of ReST

The scenario is an ordered list of simple tasks encompassing the activities that need to be performed to complete a body of work. The scenario presented in Figure 8.1 includes activities for all the external entities discovered in the design of the prototype: the Reuser, the Librarian, and the Maintainer. The scenario was performed using a sample data file that included excerpts taken from the high-level design document "Expert Roll Design" [SML98]. A copy of the excerpt is contained in Appendix B. The first column of the scenario contains the list of tasks to be performed. The result of each activity performed on the text file is presented in the middle column of the scenario. In addition, the scenario was enacted using the final prototype of ReST at a formal demonstration presented to two British Steel roll designers and a group of academics from the CARD and

REMAIN projects. The third column of the scenario has been used for comments intended to generate discussion during the demonstration. The discussion generated during the formal demonstration is included in Section 8.5. The overall time taken for the formal demonstration was under one hour including questions and discussion.

8.4 Test Results

This section contains an analysis of the test results and the discussion generated during the formal demonstration of ReST. The section is broken down to correspond to the activities identified in the scenario presented in Figure 8.1. The final prototype of ReST performed the automated portions of the activities as designed. The results of each activity were correct and as expected. Any additional tests performed using ReST are discussed in the appropriate sub-section of this section.

8.4.1 Index Document Excerpt

The underlying purpose for performing the indexing of a document excerpt was to demonstrate and test the indexing functionality of ReST. The sample data used in the test and demonstration was a file containing excerpts taken from the British Steel high-level design document "Expert Roll Design" [SML98]. The sample file was indexed and produced an index of 47 unique terms and their individual frequency counts. The results of the indexing are contained in Figure 8.2.

Surrogate	Term	Frequency
SML101-ts	bstp	1
SML101-ts	centre line	1
SML101-ts	choke	1
SML101-ts	collars	1
SML101-ts	corner	1
SML101-ts	crown	1
SML101-ts	crown radius	1
SML101-ts	crown surface	1
SML101-ts	diagonal	1
SML101-ts	dimensions	3
SML101-ts	elongation	1
SML101-ts	expansion	1
SML101-ts	fillet	1
SML101-ts	finishing pass	1
SML101-ts	finishing pass profile	1
SML101-ts	foot line	2
SML101-ts	gap	1
SML101-ts	head	1
SML101-ts	hot	1
SML101-ts	hot internal head height	1
SML101-ts	identification	1
SML101-ts	limit	2
SML101-ts	lines	1
SML101-ts	loop	1
SML101-ts	machined	2
SML101-ts	main dimensions	1
SML101-ts	meeting point	1
SML101-ts	mill spring	1
SML101-ts	non line	1
SML101-ts	open flange	2
SML101-ts	parameters	2
SML101-ts	pass	3
SML101-ts	pass centre line	1
SML101-ts	pass profiles	1
SML101-ts	pitch line	3
SML101-ts	pitch line intercepts	1
SML101-ts	point	1
SML101-ts	radius	1
SML101-ts	requirements	1
SML101-ts	roll	4
SML101-ts	series	1
SML101-ts	sharp dimensions	1
SML101-ts	spread	1
SML101-ts	starts	1
SML101-ts	toe line	1
SML101-ts	top roll	2
SML101-ts	undercut	1

Figure 8.2 Resulting Index

The resulting index was as expected. Appendix C contains a copy of the sample file with the index terms underlined. As can be seen in Appendix C, those terms that are underlined are present in the asset's automated index and the frequency counts concur.

Indexing of high-level design documents proved the success of the indexing functionality and the use of the stoplist to segregate terms. However, subsequent testing not included in the demonstration showed that the indexing functionality was not robust enough to handle the HTML design document. An excerpt from an HTML design document was indexed. The excerpt from this document can be found in Appendix B. The indexing process failed and no index was constructed. A term being constructed during indexing became too large for the process to handle. Further review of the document showed that the stoplist was not useful in helping to segregate terms. In the high-level design documents the stoplist is applied to help locate the start and end of terms. In the HTML document it would have been better to use the HTML mark-up language as indicators for the start and end of terms. The investigation of the failure revealed that it is necessary to perform a more in-depth review of not only the asset's format but also the terms contained in the design documents prior to developing a working version of ReST. This review provided the insight necessary to make the correct modifications to the current indexing functionality i.e. exploit the HTML mark-up language to locate terms contained in HTML design documents.

8.4.2 Initial Quality Assessment of Index Terms

The underlying purpose for performing the automated initial quality assessment of the index terms was to demonstrate and test the functionality in ReST that identifies terms contained in an asset that are not as yet defined in the domain knowledge base, the thesaurus. In addition, it was performed to demonstrate that preferred and defined terms in an asset could be automatically identified.

The initial quality assessment of the index terms is performed automatically at the request of a Reuser. ReST uses the thesaurus in conjunction with an asset's

index of terms to separate and list the preferred terms, the defined terms, and the terms not as yet defined in the thesaurus.

The automated initial quality assessment performed as expected and resulted in the identification of five preferred terms, nine defined terms, and thirty-three undefined terms. The five preferred terms are listed in figure 8.3. The nine defined terms are listed in figure 8.4.

Preferred Term	Frequency
elongation	1
finishing pass	1
head	1
open flange	2
spread	1

Figure 8.3 Preferred Index Terms

Defined Term	Frequency
collars	1
diagonal	1
finishing pass profile	1
gap	1
mill spring	1
pass	3
pitch line	3
roll	4
top roll	2

Figure 8.4 Defined Index Terms

8.4.3 Reclassify the Index Terms

The underlying purpose for performing the manual reclassification of the index terms was to demonstrate and test how a Reuser could accomplish domain analysis while performing component-based reuse. Reclassifying the index terms requires the Reuser to select each term and then choose the category for reclassification. The reclassification of terms to search terms has the effect of constructing a surrogate used in component-based reuse. Terms chosen to be included in the surrogate do not necessarily reflect the contents of the asset, as they would do in a true working environment. The chosen terms were selected to

demonstrate and test the functions surrounding the construction of a surrogate. This functionality includes checking to see if a search term is already in the surrogate. If it is, then the term is not copied into the surrogate and the surrogate term's frequency count is increased by the amount held in the search term's frequency count. If not, then the term and frequency count are copied into the surrogate. In addition, search terms that are not preferred terms are uplifted to their preferred term before inclusion in the surrogate. The preferred and defined terms selected to be search terms are contained in Figure 8.5.

Surrogate	Term	Frequency	Preferred Term
SML101-ts	diagonal	1	diagonal rolling
SML101-ts	elongation	1	elongation
SML101-ts	finishing pass	1	finishing pass
SML101-ts	finishing pass profile	1	finishing pass
SML101-ts	pass	3	finishing pass

Figure 8.5 Terms Selected to be Search Terms

ReST performed this reclassification as expected. The surrogate is comprised of three search terms. All three terms are defined as preferred terms in the thesaurus contained in ReST. The search term 'diagonal rolling' has a frequency count of 1. The search term was initially classified as the defined term 'diagonal' with a frequency count of 1. The defined term was automatically uplifted to its related preferred term before inclusion in the surrogate. The search term 'elongation' has a frequency count of 1. The search term was initially classified as a preferred term. The term was not changed prior to its inclusion in the surrogate. The search term 'finishing pass' has a frequency count of five. The term 'finishing pass' was initially classified as a preferred term with a frequency count of 1. The terms 'finishing pass profile' and 'pass' with frequency counts 1 and 3 respectively were initially classified as defined terms. The terms were uplifted to their preferred term 'finishing pass' prior to their inclusion in the surrogate. As the preferred term 'finishing pass' can appear only once in a surrogate the three frequency counts were added together (1+1+3) and a frequency count of 5 is stored as part of the surrogate. The surrogate for the 'asset' used in testing is illustrated in Figure 8.6.

Search Term	Frequency
diagonal rolling	1
elongation	1
finishing pass	5

Figure 8.6 Asset's Surrogate

The remaining preferred and defined terms were reclassified as internal stoplist terms. During the demonstration, the concept of an internal stoplist was difficult for one of the British Steel designers to comprehend. In the discussion generated at this part in the formal demonstration it became apparent that the names "internal stoplist" and "stoplist" were being confused and a more meaningful name for the "internal stoplist" needs to be found, perhaps 'temporary stoplist' or 'disregarded terms'.

The term 'bstp' was initially classified as an undefined term and was reclassified as a stoplist candidate as a means of demonstrating how a Reuser could contribute to the domain analysis and maintenance of the stoplist. The term 'centre line' was initially classified as an undefined term and was reclassified as a thesaurus candidate. The reclassification of a term as a thesaurus candidate required the Reuser to manually enter a proposed definition, which was taken from the glossary.

The manual reclassification of the index terms successfully demonstrated the concepts behind increasing domain understanding by allowing reusers to select the terms, definitions and relationships with which to populate the thesaurus, while establishing the need for domain knowledge before the terms, definitions and relationships are included in the thesaurus.

In the testing of the prototype for ReST as in the formal demonstration the remaining undefined terms were not reclassified. However, it should be noted that in an actual working environment it is expected that all the undefined terms be reclassified thereby increasing the thesaurus population and the size of the stoplist. As no domain expertise was applied when reclassifying the terms, there was little point in demonstrating reclassification any further.

The need for domain expertise to compile the stoplist and populate the thesaurus is an underlying premise of this thesis. Examining the list of undefined terms identified during the formal demonstration can further corroborate this premise. Figure 8.7 contains the listing of the undefined terms and their frequency count.

Undefined Term	Frequency
bstp	1
centre line	1
choke	1
corner	1
crown	1
crown radius	1
crown surface	1
dimensions	3
expansion	1
fillet	1
foot line	2
hot	1
hot internal head height	1
identification	1
limit	2
lines	1
loop	1
machined	2
main dimensions	1
meeting point	1
non line	1
parameters	2
pass centre line	1
pass profiles	1
pitch line intercepts	1
point	1
radius	1
requirements	1
series	1
sharp dimensions	1
starts	1
toe line	1
undercut	1

Figure 8.7 Undefined Terms

Only one term "centre line" is included in the glossary. No other complete term is defined in the glossary. There are terms that are partially defined in the

glossary e.g. "toe" from the undefined term "toe line" and "profiles" from the undefined term "pass profiles". Without domain expertise it is impossible to know whether the undefined terms have meaning in the domain, or are nonsense terms resulting from a flaw in either the indexing function or the stoplist.

8.4.4 Submit the Quality Assessment

At the end of the reclassification of the index terms the terms were classified as follows:

0 preferred terms
0 defined terms
31 undefined terms
9 internal stoplist terms
1 stoplist candidate
1 thesaurus candidate
3 search terms (the surrogate)

The assessment was submitted and a subsequent recovery of the quality report associated with the particular 'asset' (sample data file) showed the classifications, as expected, had remained unchanged.

The filing of the quality assessment provided an opportunity to discuss how the automated portion of the index classification could be used to improve the quality of the assets being developed. The quality assessment could be used to show developers the quality of the terms that are contained in an asset before it is included in the reuse library or even complete. This would allow them the opportunity to improve the quality of the terms by replacing undefined terms with terms contained in the thesaurus or replacing defined terms with their preferred term found in the thesaurus.

8.4.5 Reject Stoplist Candidate

The underlying purpose for reviewing the Stoplist Candidates is to test the automated functionality of the tasks involved and to demonstrate those tasks where direct intervention by a domain expert is needed. The Stoplist Candidates

are brought to the screen and the Librarian decides whether to approve or reject candidates. In the scenario above the candidate was rejected. As expected, the candidate was deleted from the stoplist candidates' table, the number of stoplist candidates was reduced by one and the stoplist remained unchanged. When a stoplist candidate is approved, the number of stoplist candidates is reduced by one and the stoplist is modified to include the approved term. This functionality was tested but not included in the demonstration. The results of the test were as expected.

8.4.6 Accept Thesaurus Candidate

The underlying purpose for reviewing the Thesaurus Candidates is to test the automated functionality of the tasks involved and to demonstrate those areas where direct intervention by a domain expert is necessary. The Thesaurus Candidates are brought to the screen for a Maintainer to review and then decide whether to approve or reject the candidates. In the scenario above (Figure 8.1) the candidate was approved. The thesaurus was brought on to screen and the candidate was added to the thesaurus as a preferred term. The total number of preferred terms held in the thesaurus was increased to forty-five. This modification was performed manually to demonstrate the lack of automation useful in populating and maintaining a thesaurus.

The largest issue raised for discussion during the formal demonstration of the prototype focused on the difficulty in defining a community acceptable standard. Before a term can be defined as a preferred term in the thesaurus it is deemed necessary for the term to be accepted by the reuser community as a 'standard' term. This requires not only the insight of domain experts but also the agreement of the reuser community. The roll designers attending the demonstration were pessimistic about the possibility of achieving a standard. It was not resolved, whether this was due to difficulty in getting the reuser community to empower a group of one or more domain experts to set a standard or the amount of effort it would require to establish a standard or a combination of these two factors.

8.4.7 Review Thesaurus

The underlying purpose for reviewing the Thesaurus was to demonstrate the type of domain knowledge that could be stored in a thesaurus and how that knowledge could be used. The review of the thesaurus during the demonstration also allowed for a discussion of the concepts and rationale behind the use of a thesaurus, including an explanation of the various relationships between the terms contained in the thesaurus.

The equivalence relationship between mill specific terms, a relationship unique to British Steel, was emphasised in the demonstration. A search was initiated to find the term 'depth', which is defined as a Scunthorpe term, as well as a preferred term. As expected the term was located. The thesaurus entry can be seen in Figure 8.8.

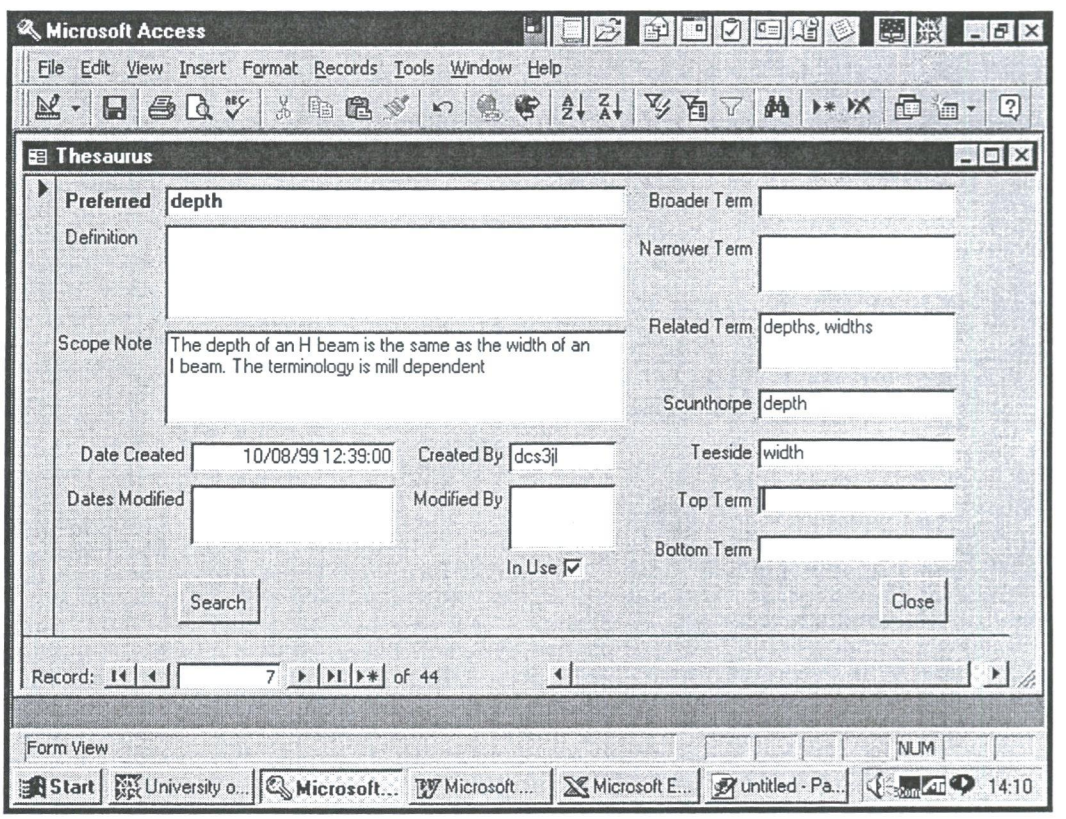

Figure 8.8 A Screen Shot of the Thesaurus in ReST

The Scunthorpe term "depth" was specifically chosen for the demonstration to illustrate how a thesaurus could be used to help overcome specific domain understanding problems, such as the difficulty of understanding the terminology

used in documents written using terminology specific to a single mill. The definition for "depth" was not written by a domain expert and is not found in the glossary.

8.4.8 Search Surrogates for Specific Terms

The remaining two activities were used to demonstrate the search functionality of the prototype. An extra surrogate for an asset named "Primary Roll Design" was constructed solely for the search test. The preferred terms were randomly selected from an index of a sample data file based on the document "Notes on Designing Primary Rolls with One Beam Shape Forming Pass" [ORD99]. The exception to the random selection was the term "finishing pass", which was included in the surrogate to demonstrate the location of more than one surrogate for a single search query.

The demonstration showed how potentially reusable assets could be located via a search of the surrogates. Search queries were comprised of a single preferred term. As surrogates are comprised of preferred terms, it is necessary for the search queries to contain only preferred terms. The preferred term selected for each search query was a term that was defined in the thesaurus as a preferred term and one of the terms contained in at least one surrogate. The search terms were found in one or more surrogates and each surrogate contained the unique identifier of the appropriate asset.

The demonstration included using the thesaurus to locate the preferred term for the non-preferred term "diagonal". This was done to demonstrate how the thesaurus could be used to help reusers define the terms to be included in a search query. This also generated discussion on using the thesaurus to vary the possible searches. For example, using a broader term to widen the search space and potentially increase the number of surrogates found to satisfy the search query.

The results of the activities were as expected. The terms used in the search queries were chosen because they were known to be present in one or more

surrogates. The surrogates located as a result of the searching were correct as regards the search queries defined.

8.5 Summary

This chapter contains the details of the testing and evaluation of the final prototype of ReST. Testing and evaluation was based around the performance of a scenario, a series of tasks to be performed to accomplish a body of work, illustrated in Figure 8.1.

When the scenario was enacted the functionality of the final prototype of ReST performed as expected. One area of concern was the indexing mechanism. Additional testing proved that more analysis of the various design documents' format and content is needed before the indexing functionality could be implemented in a work environment.

The scenario was also enacted during a formal demonstration of the final prototype of ReST. It was demonstrated that ReST has in place fundamental mechanisms that will allow reusers to practice component-based reuse while performing on-going domain analysis. It was shown that ReST provides the support necessary to index text assets; determine the quality of the terms contained in assets; and aid with the selection of terms to populate the thesaurus. Also demonstrated was how a thesaurus can be used to support domain understanding and aid with component-based reuse by helping reusers to define surrogates and search for potentially reusable assets.

One matter arising from the demonstration was the need to establish the means of deciding when a domain term was a standard term and therefore a candidate for inclusion in the thesaurus as a preferred term. Problems arise because of the effort required to establish a standard and the difficulty in getting a sanctioned group to define the standard.

The scenarios proved an adequate means of testing and evaluating ReST. The scenarios provided the means to determine if the prototype performed as expected

and promoted discussion during the formal presentation. However, testing would have been more comprehensive if there had been a larger pool of sample files and more access to domain experts. A larger number of files would have allowed for testing the recall and precision of the search functionality and a more rigorous examination of the indexing functionality. Access to domain experts for the development of realistic test data and assistance determining the expected results would have increased confidence in both the indexing mechanism and the thesaurus. The overall usefulness of ReST in the practice of reuse and ongoing domain analysis would require a full implementation of a working version of ReST and a measured study of several years.

Chapter 9 Conclusions

9.1 Objectives of the Chapter

This chapter provides the summation and evaluation of the research and work undertaken for this thesis. Section 9.2 provides a summary of the previous chapters of the thesis. Section 9.3 discusses the success of the research and the prototype type based on the criteria for success outlined in Section 1.3 of Chapter 1. Section 9.4 provides an evaluation of the thesis as a whole. Section 9.5 provides a general discussion of possible further work on the prototype ReST and the research area in general. Section 9.6 provides the summary for this chapter.

9.2 Synopsis of Work

This work began with an investigation into the general research areas of this thesis: software reuse and domain analysis as it pertains to software reuse within software engineering. Chapter 2 contains the results of a literature survey on reuse within software engineering. This included an examination of the literature relating to component-based reuse, generative reuse and the effects of domain analysis on both types of reuse. The focus of the research was then narrowed to an investigation into the effects of a changing domain on the evolution of support for component-based reuse and domain analysis and on the application of software reuse support to another engineering discipline. Chapter 3 contains the results of the more focused literature survey and includes an examination of the literature relating to component-based reuse library, and the development and maintenance of a thesaurus as a mean to capture and share the domain terminology.

British Steel's move from a decentralised roll design environment to a centralised design environment provided the domain in which to investigate the potential of applying software engineering reuse and domain analysis techniques to another engineering discipline's design process. Chapter 4 contains an analysis of British Steel's roll design domain. This analysis showed that British Steel's roll design community's need to reuse roll design artefacts, and capture and share an

understanding of the domain terminology was comparable to software engineering concepts of component-based reuse within an evolving domain.

A prototype, ReST (Reuse Support Tool) was developed to demonstrate how software engineering reuse and domain analysis techniques could be used in British Steel's roll design process to improve the following:

- the roll design process,
- the quality of the design documents, and
- the roll design community's ability to capture and share domain knowledge.

The prototype was developed to demonstrate the basis for evolving the software tool support for a component-based reuse library and a thesaurus to hold and display the associated domain terminology.

The requirements specification for ReST is contained in Chapter 5. An initial analysis of the roll design domain led to the development of an initial set of requirements and the development of the first pass of the prototype. An evaluation of the prototype and further analysis of the roll design domain led to a final set of requirements for ReST. These requirements provided the foundation for all further work on the prototype.

The final prototype of ReST was designed using dataflow diagrams and entity-relationship diagrams. Chapter 6 contains the design details of ReST. Dataflow diagrams were used to discover and show the entities, data, and processing that comprise ReST. As the prototype was used as a simple functioning model of a potentially more complex implementation, the use of dataflow diagrams proved sufficient for locating the major entities, the key processes, and the elementary data for input to and output from ReST. Entity-relationships diagrams were used to show the key relationships between the entities that comprise ReST. To clarify the relationships and make the diagrams easier to interpret an entity-relationship diagram was provided for each of the three external entities illustrated in the dataflow diagrams i.e. for the Reuser, the Librarian and the Maintainer. For

completeness a single entity-relationship containing all entities and all relationships was also included. The entity-relationship diagrams proved to be a useful means to demonstrate the different tasks that can be performed by users of ReST and to introduce the concept that the level of domain expertise required varied in accordance with the task to be performed.

The implementation details of ReST are contained in Chapter 7. The final implementation was a functioning prototype that satisfied the final requirements specification detailed in Chapter 5 and was developed on the foundations established in the design detailed in Chapter 6.

Chapter 8 contains the details and results of the evaluation of ReST. The prototype, ReST, was evaluated using scenarios. The scenarios provided a list of tasks that a reuser could perform while using ReST. The scenarios were designed to demonstrate the prototype's competence in performing reuse support while accommodating increased domain knowledge through changes to the knowledge about the domain terminology. The scenarios proved an adequate mean of testing and evaluating ReST. The scenarios were used to show that the prototype performed as expected and fulfilled the requirements stated in Chapter 5. During the formal presentation the prototype proved to be a useful device for encouraging a discussion on the underlying concepts of the thesaurus, particularly the need for the application of domain expertise when developing and maintaining a thesaurus.

9.3 Criteria for Success

This research was primarily an examination of the proposal that a thesaurus developed as part of a reuse support environment to define domain terms and their relationships can evolve as knowledge of the domain expands through reuse, and that increased understand of the domain will reveal more opportunities for reuse. In addition, this research aimed to demonstrate that specific software reuse techniques can be applied to support reuse in another engineering discipline, specifically, British Steel's roll design process.

There were three specific criteria for success for this body of work. Each of these criteria will be restated and discussed in section 9.3.1 to 9.3.3.

9.3.1 Criterion One

An investigation into software reuse and domain analysis as it applies to software reuse.

This criterion is satisfied by the literature survey contained in Chapter 2. The literature survey covers both software reuse and domain analysis as it pertains to software reuse. The investigation of software reuse focuses on component-based reuse. Assets and the means to realise them are also discussed. In addition, the survey includes an examination of generative reuse and of the proposition that the practise of component-based reuse will help reusers achieve sufficient domain understanding to move, over time, from component-based reuse to the more advantageous generative reuse.

9.3.2 Criterion Two

An investigation into software tool support for software reuse and domain analysis, which will support the evolution of the domain that must be reflected in software reuse. The focus will be on supporting the evolution of a component-based reuse library and the associated domain terminology.

This criterion is satisfied by the literature survey contained in Chapter 3. The literature survey concentrated on the support environment necessary for successful component-based reuse. It gives a detailed examination of the reuse library; and includes descriptions of the methods used to store, search, and retrieve potentially reusable components. Also included, are the results of a detailed examination of the concepts behind the development and maintenance of a thesaurus; and a description of how a thesaurus can be used as part of a software support environment to aid with both domain knowledge acquisition and sharing, and software reuse.

9.3.3 Criterion Three

Development of a prototype of a reuse environment that will support component-based reuse and will include a thesaurus that will evolve as the domain understanding is increased. The prototype will be developed for the roll design community at British Steel.

This criterion is satisfied by the development of the prototype, an evolving reuse support environment, entitled Reuse Support Tool or ReST. Even though reuse is a concept familiar in software engineering the use of ReST is evaluated in terms of its applicability to another engineering domain namely the British Steel's roll design community. The progression of the development and evaluation of ReST is contained in Chapters 4 through 8. The prototype ReST does provide the mechanisms necessary to support component-based reuse of roll design assets and concurrent domain analysis of the roll design domain terminology.

The prototype of ReST is used to demonstrate the functionality needed to support roll designers at British Steel in their aim to improve: the roll design process; the quality of the roll design assets; their understanding of the domain; and their sharing of domain knowledge. This thesis proposes that the way to improve the roll design process is to provide the means for designers to reuse design assets. The prototype for ReST provides the required mechanisms to support: the defining of an asset's surrogate; storage of surrogates in a reuse library; and searching of the surrogates of potentially reusable assets.

The prototype for ReST also provides the mechanisms necessary to aid with improving the quality of roll design assets. ReST provides an automated process that indexes an asset, producing a list of the unique terms contained in the asset. Using the thesaurus contained in ReST each index term is then automatically given a quality category. A quality report is presented to reusers that identifies not only the number of terms an asset contains but also the number of standard terms, non-standard terms, and undefined terms used in the asset. This allows the roll designers to judge the quality of the terms used in the asset. For completeness

ReST also allows a roll designer to review all the terms in an asset that fall into the specific categories.

The prototype for ReST provides the mechanisms necessary for recording reusers' increased understanding of the domain terminology that occurs as the reuser practises component-based reuse using ReST. This mechanism is a thesaurus that supplies the means to standardise the domain terminology, define the domain terms, and define the relationships between the terms. The thesaurus is comprised of only those terms that have been found during the indexing of the assets. As each asset is indexed, the terms categorised as undefined are identified by reusers and entered into the thesaurus by domain experts. The understanding of domain terminology increases as the numbers of potentially reusable assets' surrogates are added to the reuse library.

9.4 Evaluation

The prototype for ReST supports the results of the literature survey into software reuse presented in this thesis. Specifically it supports the concepts pertaining to component-based reuse libraries and the use of a thesaurus for capturing and sharing the understanding of specific domain terminology. ReST was developed for British Steel's roll design community to illustrate that software reuse support is applicable to engineering disciplines other than software engineering. ReST was successfully evaluated in this domain using scenarios based on the tasks roll designers would perform to achieve reuse of roll design assets. The evaluation clearly demonstrates that the roll design community could use ReST when performing reuse of roll design assets. The evaluation clearly demonstrated that domain experts must contribute to the development and maintenance of a domain specific thesaurus.

Discussions during the formal presentation demonstrating ReST identified a problem area not covered in this thesis. The problem is the difficulty in establishing which of the domain terms are industry standard terms and therefore candidates for inclusion in the thesaurus as preferred terms. The problem occurs because in real-world situations immense effort is required to establish a standard

terminology and the complexity of getting a group to define the standard that has been sanctioned by the community to do so.

9.5 Further Work

This section contains an overview of possible further work on the support environment ReST and in the research areas explored in this thesis.

9.5.1 Further Work on ReST

The most obvious piece of further work on ReST would be to construct a fuller implementation of ReST and measure its use in a real-world working environment.

A fuller implementation of ReST would require the following:

- An increase in the robustness of the indexing functionality already available in the prototype for ReST;
- The extension of the search functionality; and
- The means to support domain experts responsible for the maintenance of the thesaurus.

To increase the robustness of the indexing functionality in ReST there needs to be further analysis of both the content and format of the roll design assets. This analysis would provide the domain knowledge necessary to ensure the indexing function provided indices for all roll design assets that contained natural language.

The search functionality in ReST needs to be extended to include functionality that would allow multiple term searches with Boolean delimiters, and to provide the means to exploit the structure of the thesaurus. For example, extending a search area by automating the means to use broader terms instead of those already contained in the search query. It may prove useful to provide ReST with the functionality necessary to automatically uplift search terms to their preferred term value in much the same way the defined terms are uplifted to preferred terms during surrogate definition.

To ensure the continuing relevance and use of the thesaurus metrics need to be developed that will measure the use of terms when defining surrogates and search queries. There needs to be an automated way to track and report on the use of terms to assist domain experts in maintaining the thesaurus.

A more comprehensive evaluation of ReST is needed. A much larger pool of sample files and greater access to domain experts are necessary to provide a more informed judgement of ReST.

A larger number of files would allow measuring of the recall and precision of the search functionality and a more rigorous examination of the indexing functionality. It would also be useful to test ReST using full sized assets and surrogates to ensure that it performs in a time that would satisfy the potential users.

Greater access to domain experts would provide the domain knowledge necessary for the development of realistic test data. Domain experts could also be used to assist in determining the expected results, which would increase the level of confidence in the indexing mechanism and the usefulness of the thesaurus as a tool for collecting and sharing domain knowledge.

Prior to measuring an actual implementation of ReST's use in a real-world environment it would be necessary to investigate the metrics that are used in reuse and domain analysis. The research would need to lead to the identification of those metrics that could be used to indicate the success or failure of ReST in the practise of reuse and ongoing domain analysis.

9.5.2 Further Research

In this thesis it has been suggested that the practice of component-based reuse will help achieve sufficient domain knowledge to enable the practice of generative reuse. To be able to judge whether or not this proposition holds would require a much more extensive investigation into generative reuse than provided by this thesis.

This research into generative reuse would need to provide a more in-depth understanding of what constitutes successful generative reuse and details on how the success is measured. There would need to be a detailed examination of how generative reuse is intended to work and of the tools and techniques needed to support the performance of generative reuse.

Also of interest would be an investigation into whether or not the change to generative reuse would have an effect on the domain knowledge and how it is captured and shared.

9.6 Summary

The general research areas of this thesis are software reuse and domain analysis as it pertains to software reuse. The main focus of this thesis is an investigation into the effects of a changing domain on the evolution of support for component-based reuse and domain analysis and on the application of software reuse support to another engineering discipline.

The research in this thesis has been substantiated by the development of the prototype Reuse Support Tool (ReST). ReST is a reuse support environment developed to explore the effects of a changing domain on the evolution of support for component-based reuse and domain analysis. ReST confirmed that it is possible for the support tools for component-based reuse to evolve as the execution of reuse increases the understanding of the domain. The proposal that the techniques and tools of software reuse are applicable to engineering domains other than software engineering has been validated by the successful trial application of ReST with roll designers in British Steel.

Appendix A Indexing Program

An indexing program written in Perl, adapted from Larry Wall's Programming Perl [WAL96], by James Ingham B.Sc. of the University of Durham Department of Computer Science and then by Janet Lavery B.Sc.

```
#!/usr/local/bin/perl

$FileInName=shift(@ARGV);
$FileOutName=shift(@ARGV);

# This has been adapted by James Ingham then Janet Lavery from
Larry Wall's #Programming Perl p39
$/ = ""; # Enable Paragraph mode
$*=1;

# Now read each paragraph and split into words.  Record each
# instance of a word in the %wordcount associative array
open (INFILE, "< $FileInName") || die "Can't open file
$FileInName \n";
open (OUTFILE,"> $FileOutName") || die "Can't open file
$FileOutName \n";
while (<INFILE>){
      s/-\n//g;        # Dehyphenate hyphenations
      tr/A-Z/a-z/;     # Canonicalize to lowercase
      tr/a-zA-a/ /cs; # Change non-alphas to single space
      @words = split(/\W*\s+\W*/, $_);
      foreach $word (@words) {
            $wordcount{$word}++; #increment array entry
      }
}
close INFILE;

# Now print out all the entries in the %wordcount array.
# Get the word and the frequency and print them to FileOutName
# include the column headings needed for Access 97
print OUTFILE "term,frequency\r\n";
foreach $word (sort keys(%wordcount)) {
      $wordsum=$wordcount{$word};
      print OUTFILE "$word,$wordsum\r\n";
};
close OUTFILE;
```

Appendix B Sample Data Files

B.1 Expert Roll Design

Excerpt taken from "Expert Roll Design" [SML98]

File Ref. SML 101-ts
1st April 1998

Expert Roll Design

Summary: BSTP's roll design process can be considered as a series of stages. The process starts with the identification of the section parameters, as specified by the Customer, from these the finishing pass profile can be established using expansion coefficients to modify these section parameters. The pass profiles for the diagonal passes F3 to R4 are then established by a four loop approach. First the main dimensions for each pass are established, then the sharp dimensions (i.e. those dimensions required for the construction lines), then the full dimensions (i.e. the dimensions of the non-line components) and then check mechanism. This check mechanism is based on two criteria elongation and choke.

Etc.

5. Determination of Finishing Pass

5.1 Place hot section in pass.

5.2 Establish Pitch line : Rotate the section through 1.5 (around a point on the
centre-line at 2/3 of the hot internal head height (from crown).

5.3 Extend foot line up by 5/8", this is the limit of the bottom roll and forms an open
flange.

5.4 Extend the toe line across the foot line, this is the limit of the top roll.

NB. These rules governing the meeting point at the open flange are for pass
design only and do not address the requirements for collars.

5.5 Establish where the pitch line intercepts the crown radius, consider this to be the
pass centre-line.

5.6 To allow for a mill spring of 7/32" establish a gap of 1/8" between the top roll and
the pitch line and 3/32" between the bottom roll and the pitch line.

5.7 Use the smallest radius which can be adequately machined to fillet the corner.

NB. The head should be machined with undercut at the crown surface to allow
for any excessive spread.

Etc.

B.2 Notes on Designing Primary Rolls...

Excerpt taken from "Notes on Designing Primary Rolls with One Beam Shape Forming Pass" [ORD99]

NOTES ON DESINGING PRIMARY ROLLS
WITH ONE BEAM SHAPE FORMING PASS

The procedure and reasoning for the calculation of each element in the compilation of a roll profile is given with reference to the diagram in appendix 1.
The most significant fact to appreciate is that the beam shape is not identical to the roll profile, with a single forming pass this holds true whether the initial input is ingot, bloom or slab.
In drafting the web down to an appropriate size both sideways spread and elongation occur, this causes the shape web width to be wider than the roll web width and the shape flange thickness to be thinner and in some cases shorter than the roll flange. The difference between beam shape and the roll shape is referred to as underfilling.
It follows that having calculated the bewn shape required, allowances for underfilling must be made to determine the roll profile dimensions.
Where barrel space for two forming passes is available the first pass can be tailored to suit the second pass. In this event the difference between roll profile and the shape profile is negligible providing the web drafting is light.

This provides the basis for the shape width calculations and for this exercise it is assumed to be known from calculations initiated from the section standard.
Shape web- width (Sa)
The aim is to provide a shape of suitable web width which will fit central and smoothly onto the roughing rolls. If the shape width is too narrow the roughing roll corners will cut into the shape corners pushing material down towards the web causing a lap to form which will subsequently be evident on the finished section. If the shape width is too wide it may not centralise correctly onto the roughing rolls and the internal flange profile can be too wide at the toes.
Nominal section Y4dk4.- Nominal section @i,id&i;
300 & over under 300.

A Primary roll flange length (Ph)
This should be the same as the finished section flange length calculated in the hot condition. Thus the edger work on the flange toes in the beam mill is kept to minimum being confined to controlling the spread generated by drafting the flange thickness in the universal mill.
In the latter stages of drafting down a thin web of a wide beam shape and particularly those with long flanges the elongation of

the web pulls away the flange length. While the shape at 100 web may fill and roll on the flange toes if it is further drafted down to say 60 web the flange shortens. To allow for this the flange length in the roll must be longer than that required on the shape.
All section depths e 1 000 > 700 @ ,,f" Section depths under 700 > 500 with
section widths 300 & above
Section depths under 700 > 500 with section widths under 300
All section depths under 500
Flange underfill (u)
1 Ph= h x 1.01 +5% 1
) Ph= hx 1.01
See earlier comments. The wider the shape web width and the longer the flange the greater will be the value for u.
Section depths 1 000 > 700 Section depths under 700 > 500
Section depths under 500
Primar-v roll web width (Pa)
U=
U= U =
(Sa x 0.02) + (Ph x 0.15) (Sa x 0.02) + (Ph x 0.08) (Sa x 0.02) + (Ph x 0.03)
Having determined the required web width of the shape the underfill can be deducted to obtain the roll web width.
Pa = Sa - 2u
Flange tapers
n
du
0 0
Inside flange
Toe taper
Outside flange
et = 15 degrees - This is a compromise between having a taper great enough to minimise value u and small enough to locate the internal flange toes onto the roughing rolls.
p = same as the edger flange toe taper 3 -j ti 0
,5e 0 re
0 = 8 degrees - This allows recovery of the roll width with dressing while not presenting too great a taper to the U.B.M vertical rolls.
(2)

1
Shape flange thickness (Sf)
Bearing in mind that a beam shape is required to roll several weights with adjustment to the web thickness a compromise web thickness is necessary. Generally the aim is to provide a beam shape with the same flange 1 web ratio as the finished section plus a factor which will ensure slightly greater drafting on the flanges in the U.B.M keeping the thinner web in tension to prevent buckling.
Choose a section weight midway in the required range, use the web thickness to establish a ratio. Add a factor for extra flange drafting and using a nominal shape web of 60 calculate the desired shape flange thickness
Sf 60 x section flange thickness a +3 %
section web thickness a
Roll flange thickness a (Rf
Taking into account the underfilling previously explained
Rf = Sf +u
Corner laybacks

These are intended to produce a smoother corner when the web is drafted by reducing the stepping effect as the web spreads sideways. To some extent it also helps centralise the initial entry of stock.
All sections
Section width 300 & above Section width under 300
k= M= M=
Pa x 0.22 1 0
6
Web 1 layback radii (r4)
k above 50
k 50 and below
r4 = r4 =
500
200
Web 1 flange corner radii (r3)
Corner radii on roughing roll 35 & above Corner radii on roughing roll under 35
r3 = 55 r3 = 45
Web below collars (z)
This determines the minimum web thickness which can be obtained from the rolls, giving due consideration to mill spring and the minimum screw setting available. For special sections requiring only a thick web it may be deeper to improve stock entry into the pass, but this also reduces the roll diameter at the flange toes hence weakening the roll.
Generally z = 15
(3)

Flange toe radii (rl & rZ)
From a roll strength point of view these should be as big as practicable consistent with obtaining sharp corners on the finished section.
Generally:-
Section widths 200 & above section widths under 200
Collar corners
ri & r2 =25 rl & r2 = 20
To reduce the risk of introducing grooves which will develop into laps on the outer flanges a layback is used. It is also useful for guiding the bar into the pass..
Section width 200 & above Section width below 200
All sections All sections All sections
Former pass depth (Fd)
X = X =
y =
r7 = rS =
Distance from collar to flange toe.
All sections
Fd=
10
6
so
25 300
Ph + z

EDGING & SPREADER PASSES
The slab is drafted down in several edging passes the final one being referred to as the spreader.
While the standard requirement in beams from slabs rolling is for two edging passes, and one spreader the 225 thick slab is better

drafted in three edging passes as this affords greater stability and subsequently the slab is less likely to lean during drafting. Exceptionally for smaller beams where too much flange formation can be a problem the two edging passes and one spreader system is used with a 225 thick slab.
The overall roll design can be divided into three types:-
A)
B)
C)
Two edgers and a spreader for all sections rolled from a 250 thick slab.
Three edgers and a spreader for the larger bearns rolled from a 225 thick slab,
Two edgers and a spreader for the smaller beams rolled from a 225 thick slab,
The division of section sizes & roll design types are detailed in below:-
Design type & pass identity
Slab Thickness
Design Type
250
225
225
A
B
c
Section
Depths 1000 > 800
Depths 762 > 200 -
with widths below 320 Depths 762 > 400 -
with widths 254 & over Depths under 400 > 250- with widths under 300 Depths 610 > 250 -
with widths under 300
Pass Identity
PIA P2A P3A
PIB P2B P3B P4B
PIC P2C P3C
For principle pass dimensions see Appendix 3.
Given the design type and pass identity from the table most of the dimensions required are predetermined in Appendix 3.
The only adjustments to the standard pass design being the spreader width the pass depths and corner laybacks.
5)

B.3 LX_DEL_FLANGE_GUIDE

Excerpt taken from "LX_DEL_FLANGE_GUIDE.classes"

Text only HTML mark-up language not included.

TEST
Mining Library: LX_DEL_FLANGE_GUIDE
Generated by: dcs on 4/12/1999 13:07
PARTS
DESIGN_GUIDE_LINES
BOTTOM_ROLL_CL
CRAMP_HEIGHT
GUIDE_NOSE_CENTRE_HOR
GUIDE_NOSE_CENTRE_VER

PASS_CL
ROLL_CL
DESIGN_GUIDE_TEXT
BTM_ROLL_CL_TEXT
CRAMP_HEIGHT_OFFSET_TEXT
DIMENSION
DIMENSION_AB
PASS_CLEARANCE_TEXT
PASS_CL_TEXT
ROLL_CL_TEXT
SMALLEST_ROLL_FLANGE_RAD
FINISHED_GUIDE
FILLET_ABC
FILLET_BCD
FILLET_CDE
FILLET_DEF
FILLET_EFG
FILLET_JKL
GUIDE_NOSE_ARC
LAB_TIP_FILLET
FINISHED_GUIDE_PROFILE
FIN_LINE_AB
FIN_LINE_BC
FIN_LINE_CD
FIN_LINE_DE
FIN_LINE_EF
FIN_LINE_FG
FIN_LINE_JK
FIN_LINE_KL
GUIDE
GUIDE_CONSTRUCTION
ROLL
ROUGH_GUIDE
EXTRUDED_GUIDE
LINE_AB
LINE_BC
LINE_CD
LINE_DE
LINE_EF
LINE_FG
LINE_GH
LINE_HI
LINE_IJ
LINE_JK
LINE_KL
TIP_FILLET

ASSEMBLIES
GUIDES
GUIDE_DESIGN_CALCULATION
GUIDE_DESIGN_SYSTEM

LX_DEL_FLANGE_GUIDE_INDEX

LX_DEL_FLANGE_GUIDE.classes

Text with HTML mark-up language included.

<HTML><HEAD><TITLE>Class hierarchy of library
LX_DEL_FLANGE_GUIDE</TITLE></HEAD><BODY><H2 ALIGN=CENTER>TEST
</H2>
<P ALIGN=LEFT> Mining Library: LX_DEL_FLANGE_GUIDE
Generated by: dcs on 4/12/1999 13:07 </P>
<P><A href="LX_DEL_FLANGE_GUIDE_attributes.html#PARTS
"TARGET="ATTRIBUTES"> PARTS </A> </P>
<DL><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#DESIGN_GUIDE_LINES
"TARGET="ATTRIBUTES"> DESIGN_GUIDE_LINES </A>
<DL><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#BOTTOM_ROLL_CL
"TARGET="ATTRIBUTES"> BOTTOM_ROLL_CL </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#CRAMP_HEIGHT
"TARGET="ATTRIBUTES"> CRAMP_HEIGHT </A>
</DD><DD><A
href="LX_DEL_FLANGE_GUIDE_attributes.html#GUIDE_NOSE_CENTRE_HOR
"TARGET="ATTRIBUTES"> GUIDE_NOSE_CENTRE_HOR </A>
</DD><DD><A
href="LX_DEL_FLANGE_GUIDE_attributes.html#GUIDE_NOSE_CENTRE_VER
"TARGET="ATTRIBUTES"> GUIDE_NOSE_CENTRE_VER </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#PASS_CL
"TARGET="ATTRIBUTES"> PASS_CL </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#ROLL_CL
"TARGET="ATTRIBUTES"> ROLL_CL </A>
</DD></DL></DD><DD><A
href="LX_DEL_FLANGE_GUIDE_attributes.html#DESIGN_GUIDE_TEXT
"TARGET="ATTRIBUTES"> DESIGN_GUIDE_TEXT </A>
<DL><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#BTM_ROLL_CL_TEXT
"TARGET="ATTRIBUTES"> BTM_ROLL_CL_TEXT </A>
</DD><DD><A
href="LX_DEL_FLANGE_GUIDE_attributes.html#CRAMP_HEIGHT_OFFSET_TEXT
"TARGET="ATTRIBUTES"> CRAMP_HEIGHT_OFFSET_TEXT </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#DIMENSION
"TARGET="ATTRIBUTES"> DIMENSION </A>
<DL><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#DIMENSION_AB
"TARGET="ATTRIBUTES"> DIMENSION_AB </A>
</DD></DL></DD><DD><A
href="LX_DEL_FLANGE_GUIDE_attributes.html#PASS_CLEARANCE_TEXT
"TARGET="ATTRIBUTES"> PASS_CLEARANCE_TEXT </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#PASS_CL_TEXT
"TARGET="ATTRIBUTES"> PASS_CL_TEXT </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#ROLL_CL_TEXT
"TARGET="ATTRIBUTES"> ROLL_CL_TEXT </A>
</DD><DD><A
href="LX_DEL_FLANGE_GUIDE_attributes.html#SMALLEST_ROLL_FLANGE_RAD
"TARGET="ATTRIBUTES"> SMALLEST_ROLL_FLANGE_RAD </A>
</DD></DL></DD><DD><A
href="LX_DEL_FLANGE_GUIDE_attributes.html#FINISHED_GUIDE
"TARGET="ATTRIBUTES"> FINISHED_GUIDE </A>
<DL><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#FILLET_ABC
"TARGET="ATTRIBUTES"> FILLET_ABC </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#FILLET_BCD
"TARGET="ATTRIBUTES"> FILLET_BCD </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#FILLET_CDE
"TARGET="ATTRIBUTES"> FILLET_CDE </A>

</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#FILLET_DEF
"TARGET="ATTRIBUTES"> FILLET_DEF </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#FILLET_EFG
"TARGET="ATTRIBUTES"> FILLET_EFG </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#FILLET_JKL
"TARGET="ATTRIBUTES"> FILLET_JKL </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#GUIDE_NOSE_ARC
"TARGET="ATTRIBUTES"> GUIDE_NOSE_ARC </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#LAB_TIP_FILLET
"TARGET="ATTRIBUTES"> LAB_TIP_FILLET </A>
</DD></DL></DD><DD><A
href="LX_DEL_FLANGE_GUIDE_attributes.html#FINISHED_GUIDE_PROFILE
"TARGET="ATTRIBUTES"> FINISHED_GUIDE_PROFILE </A>
<DL><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#FIN_LINE_AB
"TARGET="ATTRIBUTES"> FIN_LINE_AB </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#FIN_LINE_BC
"TARGET="ATTRIBUTES"> FIN_LINE_BC </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#FIN_LINE_CD
"TARGET="ATTRIBUTES"> FIN_LINE_CD </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#FIN_LINE_DE
"TARGET="ATTRIBUTES"> FIN_LINE_DE </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#FIN_LINE_EF
"TARGET="ATTRIBUTES"> FIN_LINE_EF </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#FIN_LINE_FG
"TARGET="ATTRIBUTES"> FIN_LINE_FG </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#FIN_LINE_JK
"TARGET="ATTRIBUTES"> FIN_LINE_JK </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#FIN_LINE_KL
"TARGET="ATTRIBUTES"> FIN_LINE_KL </A>
</DD></DL></DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#GUIDE
"TARGET="ATTRIBUTES"> GUIDE </A>
</DD><DD><A
href="LX_DEL_FLANGE_GUIDE_attributes.html#GUIDE_CONSTRUCTION
"TARGET="ATTRIBUTES"> GUIDE_CONSTRUCTION </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#ROLL
"TARGET="ATTRIBUTES"> ROLL </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#ROUGH_GUIDE
"TARGET="ATTRIBUTES"> ROUGH_GUIDE </A>
<DL><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#EXTRUDED_GUIDE
"TARGET="ATTRIBUTES"> EXTRUDED_GUIDE </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#LINE_AB
"TARGET="ATTRIBUTES"> LINE_AB </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#LINE_BC
"TARGET="ATTRIBUTES"> LINE_BC </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#LINE_CD
"TARGET="ATTRIBUTES"> LINE_CD </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#LINE_DE
"TARGET="ATTRIBUTES"> LINE_DE </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#LINE_EF
"TARGET="ATTRIBUTES"> LINE_EF </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#LINE_FG
"TARGET="ATTRIBUTES"> LINE_FG </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#LINE_GH
"TARGET="ATTRIBUTES"> LINE_GH </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#LINE_HI
"TARGET="ATTRIBUTES"> LINE_HI </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#LINE_IJ
"TARGET="ATTRIBUTES"> LINE_IJ </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#LINE_JK
"TARGET="ATTRIBUTES"> LINE_JK </A>

</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#LINE_KL
"TARGET="ATTRIBUTES"> LINE_KL </A>
</DD><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#TIP_FILLET
"TARGET="ATTRIBUTES"> TIP_FILLET </A>
</DD></DL></DD></DL><HR><P><A
href="LX_DEL_FLANGE_GUIDE_attributes.html#ASSEMBLIES
"TARGET="ATTRIBUTES"> ASSEMBLIES </A> </P>
<DL><DD><A href="LX_DEL_FLANGE_GUIDE_attributes.html#GUIDES
"TARGET="ATTRIBUTES"> GUIDES </A>
</DD><DD><A
href="LX_DEL_FLANGE_GUIDE_attributes.html#GUIDE_DESIGN_CALCULATION
"TARGET="ATTRIBUTES"> GUIDE_DESIGN_CALCULATION </A>
</DD><DD><A
href="LX_DEL_FLANGE_GUIDE_attributes.html#GUIDE_DESIGN_SYSTEM
"TARGET="ATTRIBUTES"> GUIDE_DESIGN_SYSTEM </A>
</DD></DL><HR><P><A
href="LX_DEL_FLANGE_GUIDE_attributes.html#LX_DEL_FLANGE_GUIDE_INDEX
"TARGET="ATTRIBUTES"> LX_DEL_FLANGE_GUIDE_INDEX </A> </P>
<HR></BODY> </HTML>

Appendix C Comparison Test Data

Excerpt taken from "Expert Roll Design" [SML98]
Phrases underlined in this copy of the excerpt were manually located and are to be used in the evaluation of the indexing functionality of the prototype for ReST.

File Ref. SML 101-ts
1st April 1998

Expert Roll Design

Summary: BSTP's roll design process can be considered as a series of stages. The process starts with the identification of the section parameters, as specified by the Customer, from these the finishing pass profile can be established using expansion coefficients to modify these section parameters. The pass profiles for the diagonal passes F3 to R4 are then established by a four loop approach. First the main dimensions for each pass are established, then the sharp dimensions (i.e. those dimensions required for the construction lines), then the full dimensions (i.e. the dimensions of the non-line components) and then check mechanism. This check mechanism is based on two criteria elongation and choke.

Etc.

5. Determination of Finishing Pass

5.1 Place hot section in pass.

5.2 Establish Pitch line : Rotate the section through 1.5 (around a point on the centre-line at 2/3 of the hot internal head height (from crown).

5.3 Extend foot line up by 5/8", this is the limit of the bottom roll and forms an open flange.

5.4 Extend the toe line across the foot line, this is the limit of the top roll.

NB. These rules governing the meeting point at the open flange are for pass design only and do not address the requirements for collars.

5.5 Establish where the pitch line intercepts the crown radius, consider this to be the pass centre-line.

5.6 To allow for a mill spring of 7/32" establish a gap of 1/8" between the top roll and the pitch line and 3/32" between the bottom roll and the pitch line.

5.7 Use the smallest radius which can be adequately machined to fillet the corner.

NB. The head should be machined with undercut at the crown surface to allow for any excessive spread.
Etc.

References

AIT72 Jean Aitchison and Alan Gilchrist, Thesaurus Construction A Practical Manual, Aslib, 1972.

ARA91 Guillermo Arango and Rubén Prieto-Díaz, Domain Analysis Concepts and Research Directions. In Domain Analysis and Software Systems Modelling, IEEE Computer Society Press, 1991.

BAS97 Paul G. Bassett, Framing Software Reuse: Lessons from the Real World, Prentice Hall PTR, 1997.

BIG98 Ted J. Biggerstaff, A perspective of generative reuse. In Annals of Software Engineering, Vol. 5, pp. 349-414, September 1998.

BRO95 Frederick P. Brooks, JR., The Mythical Man-Month Essays on Software Engineering, Anniversary Edition, Addison-Wesley Publishing Company, 1995.

BS97 Glossary of Roll Design Terms, provided by British Steel. Dated 29 January 1997

CHA99 B. Chandrasekaran, John R. Josephson and V. Richard Benjamins, What Are Ontologies, and Why Do We Need Them?. In IEEE Intelligent Systems, Vol. 14, No. 1, January/February 1999.

DOW99 Downtime, Imperial measures strike back. In Computer Weekly, 7 October 1999.

FRA94 William B. Frakes and Thomas P. Pole, An Empirical Study of Representation Methods for Reusable Software components. In IEEE Transactions on Software Engineering, Vol. 20, No. 8, August 1994.

GEN97 W. Morven Gentleman, Effective Use of COTS (Commercial-Off-the-Shelf) Software Components in Long Lived Systems. In Proceedings of the 1997 International Conference on Software Engineering, IEEE Computer Society Press, 1997.

HAL91 Patrick Hall and Cornelia Boldyreff, Chapter 41 of Software Engineer's Reference Book, edited by John A. McDermit, Butterworth - Heinemann Ltd., 1991

HEN94 Scott Henninger, Using Iterative Refinement to Find Reusable Software. In IEEE Software, September 1994.

ISO2788 International Standard ISO 2788 - Documentation - Guidelines for the establishment and development of monolingual thesauri, Second edition, 1986-11-15.

JAR95 Stan Jarzabek, From reuse library experiences to application generation architecture. In ACM SIGSOFT Software Engineering Notes Special Issue Proceedings of the Symposium on Software Reusability, eds. Mansur Samadzadeh and Mansour Zand, August 1995.

KEL97 F. Kelledy and A.F. Smeaton, Automatic Phrase Recognition and Extraction from Text. In Information Retrieval Research Proceedings of the 19th Annual BCS-IRSG Colloquim on IR Research, April 1997.

KIE98 D. Kiely, Are Components the Future of Software? In Computer, February 1998.

KRU93 I. Kruzela, Successful Management Structures for Reuse. In P. Walton and N. Maiden, editors, Integrated Software Reuse: Management and Techniques, Ashgate Publishing Limited, 1993.

MCP93 Dr. P. McParland, Application Templates: Reusable Design. In P. Walton and N. Maiden, editors, Integrated Software Reuse: Management and Techniques, Ashgate Publishing Limited, 1993

MIL94 Hafedh Mile, et al., Practitioner and SoftClass: A Comparative Study of Two Software Reuse Research Projects. In Journal of Systems Software, No. 25, pp. 147 - 170, 1994.

MIL97 Hafedh Mili, Estell Ah-Ki, Robert Godin and Hamid Mcheick, Another nail to the coffin of faceted controlled-vocabulary component classification and retrieval. In Software Engineering Notes, Vol. 22, No. 3, Proceedings of the ACM SIGSOFT 1997 Symposium on Software Reusability. ed. Mehdi T. Harandi, May 1997.

MIL98 A. Mili, R. Mili, and R.T. Mittermeir, A survey of software reuse libraries. In Annals of Software Engineering, Vol. 5, pp. 349-414, September 1998.

NEI89 James M. Neighbors, DRACO: A Method for Engineering Reusable Software Systems. In Software Reusability, Volume 1 Concepts and Models, eds. Ted J. Biggerstaff and Alan J. Perlis, ACM Press, 1989.

ORD99 D. Ord, Notes on Designing Primary Rolls with One Beam Shape Forming Pass. British Steel, Internal Document, 17th June 1999.

ORN93 Stephen B. Ornburn and Richard J. LeBlanc, JR., Building, Modifying, and Using Component Generators. In Proceedings 15th International Conference On Software Engineering, May 17 - 21, 1993, IEEE Computer Society Press, 1993.

POU97 J.S. Poulin, Measuring Software Reuse Principles, Practices and Economic Models, Addison Wesley Longman, Inc., 1997.

PRE97 Roger S. Pressman, Software Engineering A Practitioner's Approach, Forth Edition, Adapted by Darrel Ince, MaGraw-Hill, 1997.

PRI89 R. Prieto-Díaz, Classification of Reusable Modules. In Software Reusability Volume 1 Concepts and Models, eds. Ted J. Biggerstaff and Alan J. Perlis, ACM Press, 1989.

PRI91 R. Prieto-Díaz, Domain Analysis For Reusability. In Domain Analysis and Software Systems Modelling, IEEE Computer Society Press, 1991.

RAD90 Roy Rada, Maintaining Thesauri and Metathesauri. In Int. Classif. 17, No. 3/4, 1990.

REI97 Donald J. Reifer, Practical Software Reuse Strategies for Introducing Reuse Concepts in Your Organization, John Wiley & Sons, Inc., 1997.

SML98 SML101-ts, Expert Roll Design. British Steel Internal Document, 1st April 1998.

SNE98 Harry M. Sneed, Measuring Reusability of Legacy Software Systems. In Software Process-Improvement and Practice Vol. 4, pp. 43 - 48, 1998.

SOM96 Ian Sommerville, Software Engineering, Fifth Edition, Addison-Wesley Publishing Company Inc., 1996.

SWA99 William Swartout and Austin Tate, Ontologies. In IEEE Intelligent Systems, Vol. 14, No. 1, January/February 1999.

THI97 Scott Thibault and Charles Consel, A Framework for Application Generator Design. In Software Engineering Notes, Vol. 22, No. 3, Proceedings of the ACM SIGSOFT 1997 Symposium on Software Reusability. ed. Mehdi T. Harandi, May 1997.

VAL99 Andre Valente, Thomas Russ, Robert MacGregor, and William Swartout, Building and (Re)Using an Ontology of Air Campaign Planning. In IEEE Intelligent Systems, Vol. 14, No. 1, January/February 1999.

WAL96 Larry Wall, Tom Christinasen, and Randal L. Schwartz with Stephen Potter, Programming Perl, Second Edition. O'Reilly & Associates Inc., 1996.

WAS95 Micheal Wasmund, The Spin-Off Illusion: Reuse Is Not a By-product. In ACM SIGSOFT Software Engineering Notes Special Issue Proceedings of the Symposium on Software Reusability, eds. Mansur Samadzadeh and Mansour Zand, August 1995.

ZAN97 Zand Mansour, Reuse R&D: Is it on the right Track, Introduction and organisational issues. In Software Engineering Notes, Vol. 22, No. 3, Proceedings of the ACM SIGSOFT 1997 Symposium on Software Reusability. ed. Mehdi T. Harandi, May 1997.