Supporting Software Reuse with Configuration Management and the World Wide Web

by

Christopher Marlowe Barchak

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Computer Science and Engineering and

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1995

Author .....
Department of Electrical Engineering and Computer Science
May 18, 1995

Approved by .....
Anne Clough
Technical Supervisor, Charles Stark Draper Laboratory

Certified by .....
Barbara Liskov
Thesis Supervisor

Accepted by .....
F. R. Morgenthaler
Chairman, Department Committee on Graduate Students

Supporting Software Reuse with Configuration Management and the World Wide Web

by

Christopher Marlowe Barchak

Submitted to the Department of Electrical Engineering and Computer Science

May 18, 1995

In Partial Fulfillment of the Requirements for the degrees of
Bachelor of Science in Computer Science and Engineering and
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The increasing size and complexity of software systems strain the traditional ad hoc software development process. Rising development costs coupled with a substantial investment in maintaining legacy software constitute a Software Crisis. Software reuse attempts to leverage past workproducts for integration in new ones and potentially provides more robust systems at less cost. Software configuration management (SCM) systems are important to this reuse effort; such systems are necessary to increase confidence in code stability, which in turn facilitates reuse. Once the number of reusable artifacts in a collection reaches a critical point, a software reuse library becomes a necessary component of reuse, for it provides browsable access to collections of reusable components that would be too large for any one person to fully utilize. This thesis describes the implementation of a process-based SCM system, and the design and implementation of a low-cost prototype of a World Wide Web-based software reuse library constructed primarily from freely available open systems.

Technical Supervisor: Anne Clough
Title: Principal Member of the Technical Staff, Charles Stark Draper Laboratory

Thesis Supervisor: Barbara Liskov
Title: Professor of Computer Science and Engineering

Acknowledgements

May 18, 1995

It goes without saying that this thesis would not have been possible without the support of numerous people. I would like to thank Anne Clough for agreeing to supervise my thesis at the last minute after my previous advisor had left the Laboratory. Your helpful criticism and enthusiastic support helped me pull through and finish. My thanks also go to Barbara Liskov, for her insightful comments and rapid return of my drafts. I would also like to thank Stuart Roseman, the original supervisor of this work, for giving me an opportunity to learn so many new skills and making work a fun place to be.

My M.I.T. experience was a truly rewarding one, in large part due to the friends I made over the years. I have too many great friends at Beta Theta Pi to name names without leaving anybody out, but the friendships I made there had a profound effect on my college experience. I also want to thank Alicia for her love and support over these past two years. Finally, and most importantly, I wish to thank my brother Rich and my parents, Len and Oili. Your constant love and encouragement were a source of strength during my years at school, and I owe you a debt I shall never be able to repay.

This thesis was prepared at The Charles Stark Draper Laboratory, Inc. under Internal Research and Development Project #13083, "Software Process Improvement."

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Software Crisis

The cost of software development as a percentage of total system costs in engineering projects is widely considered to be rising more quickly than necessary. Further, the size and complexity of these systems, many of which handle critical tasks, make code quality of increasing concern to software developers and users. Many observers maintain that the rising costs of development combined with increasingly risk-prone critical systems constitute a Software Crisis.

1.1.1 Economics

According to several established analyses, the software portion of many systems accounts for the majority of its cost [1]. The U.S. Department of Defense alone is projected to now spend over $30 billion per year on software development. In addition to direct and maintenance costs, there are the indirect costs of faulty and delayed software in lost profits or productivity [2].

These large costs may be attributed to the requirements of increasingly ambitious systems, rising salaries from an increased demand for qualified software professionals, and a lack of productivity improvement despite the introduction of new software development tools and methodologies [2]. While not much can be done about the first two factors, there is considerable interest in improving the tools and methods of software development in order to seek out the elusive productivity gains which are seemingly a matter of course with our hardware counterparts.

1.1.2 Installed Base

Some of the greatest cost in software systems lies not in creating newer, more complex systems, but rather in maintaining investments in older, legacy systems. These older sys-

tems were created before modern advances like object-oriented programming, formal design methodologies, and CASE tools. According to recent studies by Dun and Bradstreet (1991) and Ardak Corp. (1993), the number of lines of code in functioning systems exceeds 100 billion lines worldwide. Further, of the five million computer programmers currently employed, four million are performing software maintenance rather than new development [3]. The astounding size of the intellectual investment in legacy software systems demands that responses to the software crisis must include the ability to recover value from legacy systems as well as to simplify the design of newer systems.

1.1.3 Software Reuse as a Solution to the Software Crisis

One technology effort which attempts to address the above concerns is software reuse, defined as "the use of engineering knowledge or artifacts from existing systems to build new ones." While small groups have historically practiced reuse in an ad hoc manner, software development organizations are attempting to achieve systematic, domain-based, repeatable reuse [4]. By reusing well-understood, trusted, and available artifacts, the cost of development can be held down while increasing confidence in correctness.

1.2 Project Motivation

1.2.1 Draper Software Environment

A wide variety of software projects are in development within Draper Laboratory, many of which exist within the Software Engineering Directorate that is the context for this thesis. While individual projects had long adopted simple configuration management tools like RCS and SCCS, and reuse existed within projects on a limited basis, there was little consistency among development methodologies nor knowledge sharing between projects. Individual projects developed their own methods of ensuring stability and quality, while some operated in a decidedly ad hoc manner. While there were not any real catastrophes with this anarchic environment, the potential for one was present.

An independent research and development project devoted to the cause of Software Process Improvement was instituted in order to improve the quality and efficiency of all software engineering at Draper, especially within the Software Engineering Directorate. As part of this effort, a Tools and Methods group was formed, which attempted to improve the software process with the aid of both commercial and custom software tools and explicit design of a software engineering process methodology.

1.2.2 Capability Maturity Model

Draper's position as a primarily government contractor further added to the need for improving the software process, due to expected mandatory standards for software development for government contractors. After spending large sums of money on software development with unrepeatable and inconsistent results, the federal government has defined as a major priority the understanding of how to manage software processes, as the key to actually benefitting from the promise of new development methodologies and technologies [5].

The federal government has contracted the Software Engineering Institute (SEI) at Carnegie-Mellon University to develop software process improvement methods to be used nationwide for contractors. SEI has developed a software process model called the Capability Maturity Model (CMM) and a process rating system by which the maturity of contractors' development process will be measured. The rating system ranges from 1 to 5, with 1 designated the starting level and 5 being the most advanced level [5].

The government is expected to mandate that all software contractors achieve a certain level of process maturity within the next few years or face loss of contracts. While the CMM system is not without its critics, it is at least an initial effort by the government to insist on some minimum level of professionalism in software development methodology. Similar standards for professionals in law, medicine, and engineering have long been in

place, but their necessity in the computer science community is still the subject of vigorous debate.

1.3 Software Reuse

1.3.1 Approaches to Software Reuse

While most agree on the need to achieve systematic reuse, there are many different vehicles to attempt to achieve it. Broadly speaking, all these attempts can be classified as either composition-based or generation-based systems. Real systems are frequently mixtures of the two, but the distinction between the two extremes is useful for the sake of analysis [6].

In composition systems, reused components are generally atomic and are ideally not modified at all in the reuse process. While in practice such components must actually undergo some processing, the ideal case is one of "passive elements operated upon by an external agent." Such elements might be code skeletons, subroutines, functions, programs, or objects. The efficient composition of new programs from the elements requires well-defined composition principles, such as those in the UNIX pipe mechanism, which allows easy connection of a program's inputs and outputs [6]. Object-oriented programming languages like Smalltalk and C++ provide another method of simplifying module interconnections by implementing a generalized function call mechanism to encapsulate code and data to hide the actual implementation from the caller.

The components reused by generation-based systems are less concrete and self-contained. They are often expressed as patterns in the generator program, and may not resemble the final artifacts of the process [6]. There are three major subclasses of generation-based systems: language-based systems, application generators, and transformational-based systems. Language-based systems use a well-defined specification language for a particular domain, but hide the implementation details from the user. The user programs at

the domain level, and a type of "compiler" turns the specification into an implementation. Transformational-based systems focus on generating new target programs by applying transformations to high-level specifications to produce an equivalently functioning system on several architectures. These systems provide promise for researchers looking to easily port programs to new architectures with less effort [7]. Application generators are employed to build applications in a specific domain, such as database or financial systems. They have very specific domain knowledge and a very precise specification language. These tight constraints limit their general applicability, but allow naive users to quickly produce functional and useful systems [2].

1.3.2 Challenges to Successful Software Reuse

As in any engineering system, design of a software reuse system necessarily entails trade-offs between conflicting needs. No single technological aid will solve all software engineering challenges - there is no "silver bullet." Therefore, each individual contribution needs to be analyzed to determine its true potential benefit.

There are a number of general conflicts inherent in software reuse technologies. One of these is the Generality versus Payoff conflict. Those methods with extremely broad reach, such as formal methods and high-level languages, tend to have modest payoff in increased productivity over base-level technology. Methods with extremely limited focus, such as database application generators, offer great power in their domain, but offer no improvement in other areas [6]. A great challenge of software reuse research is the search for methods that are general, yet powerful.

Another challenge is the conflict between component size and its reusability. Larger components, such as whole subsystems or applications, greatly increase the immediate benefit of reuse when compared to subroutines or sections of code. Unfortunately, larger

artifacts are typically more specific to a particular domain, and thus harder to reuse and entail more modifications [6].

Finally, there is the dilemma of conflicting long-term and short-term goals, especially with reuse libraries. It is likely to be more expensive to design and develop code that exhibits reusability and generality. On the other hand, if components are not designed with reuse in mind, they may be of little actual use to others. Before significant return on investment is seen, much expenditure of effort, time, and money is required. Since the long-term payoff is expected but not guaranteed, organizations may be hesitant to make the investment. Further, most organizations are made of teams with specific project goals, with no budgetary incentive to work further to generalize their results for use by others, even within the organization. Populating a library adequately is necessary to achieve its viability and utility, but short-term monetary concerns may override such benefits in many organizations [6].

A composition-based reuse system must support four major tasks: finding, understanding, modifying, and composing components. The finding portion is optimally used to locate not just exact matches to search criteria, but also similar components, since some may need to be only partially modified to be reused. The understanding portion is most necessary in cases where some modification is required, since the reuser requires guidance in order to understand the system. Hypertext offers some promise in this area, as related documents like specifications, requirements, and diagrams can be conveniently co-located with the actual code. While modification is necessary for more general applicability of components, at this point the process involves mostly manual work. Since the act of composition requires an understanding of the interfaces established by the connected components, the software support of component composition is heavily intertwined with the specification representation. Without an established system of formal specification,com-

position remains a human endeavor. There also remain unsolved problems in functional composition, as software systems have both global and local effects, and are not just simple mathematical compositions [6].

1.4 Thesis Scope

Before the start of this thesis project, there was little awareness or systematic support for software reuse at Draper Laboratory. While some reuse was practiced informally among colleagues with close working relationships, there was little cross-fertilization between separate groups.

The major focus of the Software Process Improvement group had been increasing quality in new individual projects, with less regard for recouping the investments in legacy systems or potentially adding value through allowing projects access to the work of other groups in the Laboratory. One of the major thrusts of the thesis is the necessity of providing both software tools and system-wide development methodologies to support a more systematic approach to reuse.

On a more practical level, the thesis describes the implementation of a software configuration management system and the design and implementation of a low-cost prototype software reuse library constructed primarily from freely available open systems. The necessity of adopting a configuration management system had already been established, but no actual implementation was in place. The author, as well as two colleagues in the Tools and Methods group, was responsible for the system implementation. The responsibility for design and implementation of the reuse library lay solely with the author.

1.5 A note on Internet-influenced research

This project, like many other contemporary research efforts, has drawn widely on many Internet resources, from World Wide Web home pages, to Usenet newsgroups, ftp sites,

and worldwide researchers. While the resources of the Internet are vast and timely, they can be troublesome to reference in a scholarly work, since the documents they reference are not fixed in form, content, or location, and may not be available just a few years after a document's completion. Wherever possible, this thesis references published references, even though the original insights may have come from on-line systems. A current but necessarily soon to be outdated list of useful Internet resources for software reuse and reuse library issues is included as Appendix A.

Chapter 2

Configuration Management

2.1 SCM Systems

One of the keys to improving the software process past the initial level in the Capability Maturity Model is adoption of a software configuration management (SCM) system. The purpose of SCM is to "establish and maintain the integrity of the products of the software project throughout the project's software life cycle." [5] SCM allows controlled changes to software artifacts, including not just code, but design documents, test cases, interfaces, and so forth. Such a system is capable of enforcing standards to ensure a stable development environment in which teams of programmers and managers interact in order to meet customer needs.

Effective reuse is facilitated by such a configuration management system, since two of the most fundamental impediments to effective reuse are access and confidence [8]. When all lifecycle objects are versioned automatically, there are universal and unique identifiers, so the reuser can be confident that the artifact that was retrieved from the database is the correct one. Further, unexpected changes do not occur during builds because of potentially harmful unannounced changes to objects. A mandatory version history is maintained with changes in features explicitly described, so reusers can choose the version most appropriate to their needs. An effective SCM system can ensure confidence in reuse candidates by enforcing criteria for code that is to be reused. Confidence in the reused code is increased because artifacts need to meet certain criteria in order to be candidates for reuse. These criteria include testing information, such as percentage of source lines tested and actual test cases. By excluding those artifacts which fail to meet such criteria from reuse, only those artifacts shown to be reasonably trustworthy will be shared.

2.2 Draper CM Process

2.2.1 Purpose

While most software engineering researchers acknowledge the value of an explicit software development process, some discussion of Draper's particular purpose for the CM process is in order. One of the major goals is compartmentalization of the different portions of the life cycle, so that no single team member can "corrupt the configuration in any significant or even inconvenient way." [8] Each action of the process will be reversible, so that any previous versions can be recovered. Process actions will be tracked automatically to generate metrics for determining how long products spend in each stage of the software lifecycle. Team members will have full authority and responsibility in their particular roles. Members with multiple roles can separate their duties such that they only need concentrate on one action at a time. Finally, the CM process itself will be subject to change requests and enhancements from developer feedback, just as any other project.

2.2.2 Requirements

In addition to meeting the goals above, Draper's CM tool needed to meet certain process requirements, stipulated from institutional experience with software development. Unique and automatic identifiers were required for every version of every document, file, and assembly, so that they could be identified unambiguously. The revision control portion had to have the ability to recreate any previous version of all components and assemblies. The process required event driven actions, including the potential for email notification. The tool required access from a wide range of platforms, and had to be extensible to allow accommodation of new tools. Finally, the tool should require minimal administrative oversight [8].

2.2.3 Overview

The CM system can best be described using a state transition diagram. The configured components in the process are in a particular "state," and are acted upon by engineers per-

forming "actions" while playing certain "roles." Each state is in the control of an engineer acting in some specified role, and components in a particular state can only be promoted into the next state by the engineer in the controlling role. Roles are assigned according to user identifications. The engineer in the controlling role has the authority to promote items in the lifecycle to the next state, where an engineer playing another role usually has control of the item. Each state has its own specified promotion criteria, which the engineer in the controlling role has the responsibility of judging against when deciding about promotion. The use of a role based system allows an easy mechanism for explicitly authorizing team members to take action. When there are fewer team members than roles, members may be assigned more than one role [8].

While the actual complete state diagram is more complicated, a simplified version of the Draper CM process model is illustrated in Figure 2.1 (adapted from McLachlan's Quick Guide [9]). The basic states shown are:

- Start
- Create a Task
- Write Code
- Integrate
- Test
- Release
- PCR/ECR

Since the CM process is continuous, the Start state provides an entry point, but the process transitions quickly to Create a Task as a first step in accomplishing some goal. This goal might be a bug fix, an approved requested enhancement, or an original task. Next the project transitions to Write Code, although "code" here means any editing of documentation or supporting files as well. After successful compilation and unit testing, the process moves on to Integrate, which ensures that the new code integrates with the rest of the system. If it does, the product can move on to Test, where formal testing procedures are implemented. If these too are successful, the process transitions to Release, where the

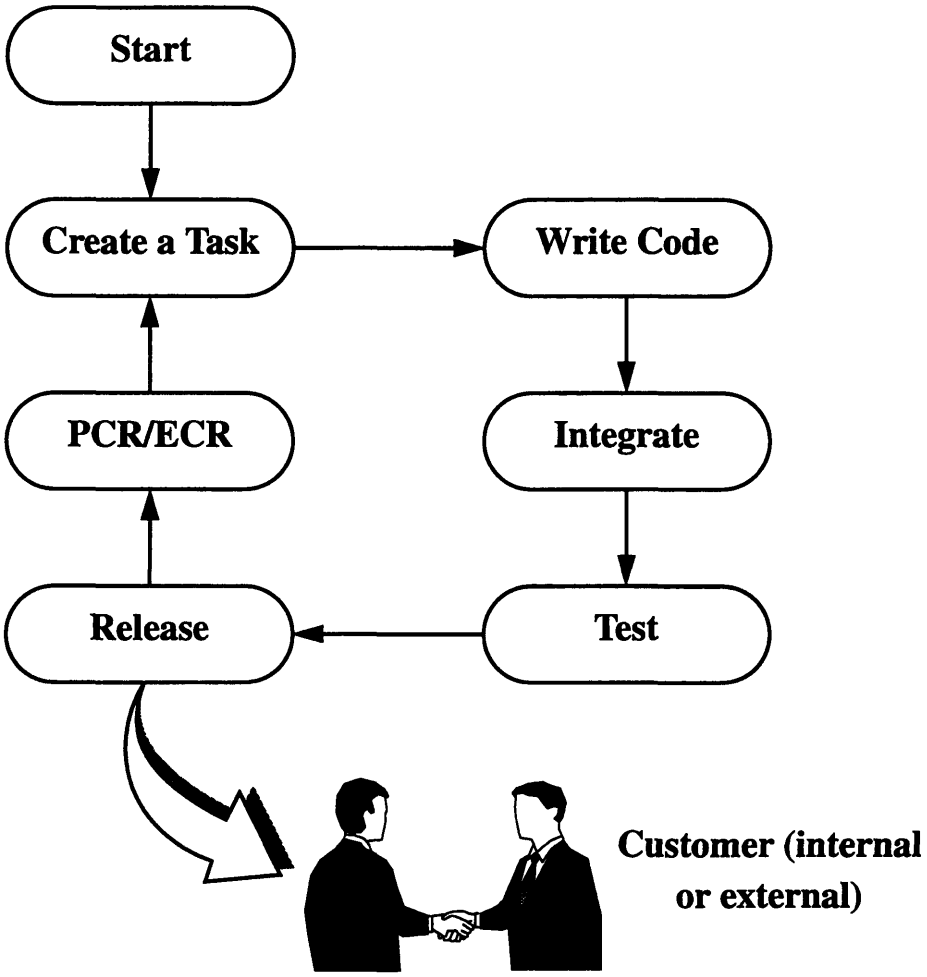

Figure 2.1: Simplified Draper Process Model

item is released either to an external or internal customer. Later, when a bug is found or enhancement is requested, the process transitions to PCR/ECR (Problem Change Request/ Engineering Change Request) to make a change request, and then transitions back to Create a Task [9]. The model is continuous so that a Released component always has the potential to incorporate new changes for future versions.

2.2.4 Configured Components

Code is emphatically not the only type of component managed by the CM system. Anything related to the software development process is managed and versioned. Such documents include:

- Proposals
- Task Agreements
- Requirements Documents
- Interface Documents
- Design Documents
- Code
- Build Files
- Include Files
- Man/help Pages
- Test Documents
- Documentation [8]

In addition to configuration managing the above elements, the CM tool itself, the operating system, and all build and development tools (e.g. compilers, linkers) should be configuration managed to have a totally consistent and re-creatable build environment.

2.3 Draper CM System

Draper has chosen the tool Continuus/CM [10] to support its software process model. While many competitive products exist, Continuus/CM has built-in support for roles and states, and thus required less customization to meet Draper requirements.

2.3.1 System Structure

A diagram of the CM hardware system is given in Figure 2.2. A database server on a single Sun workstation provides database engines for general access. Such engines can be distributed throughout the organization to serve users locally on most standard Unix workstations. Engines can also be accessed remotely through either an X-windows or command-line interface, thus effectively opening up the tool's use to all platforms. The configuration management database is stored on a central file server accessible to all users of the CM tool. The file server is a Network Appliance FAServer [11], a dedicated NFS file server with near-local access time. Another dedicated Sun workstation provides administrative capability. Both Suns and the FAServer reside on a high-speed fiber-optic FDDI ring. The ring is connected to the Draper Ethernet via a CISCO bidirectional FDDI-

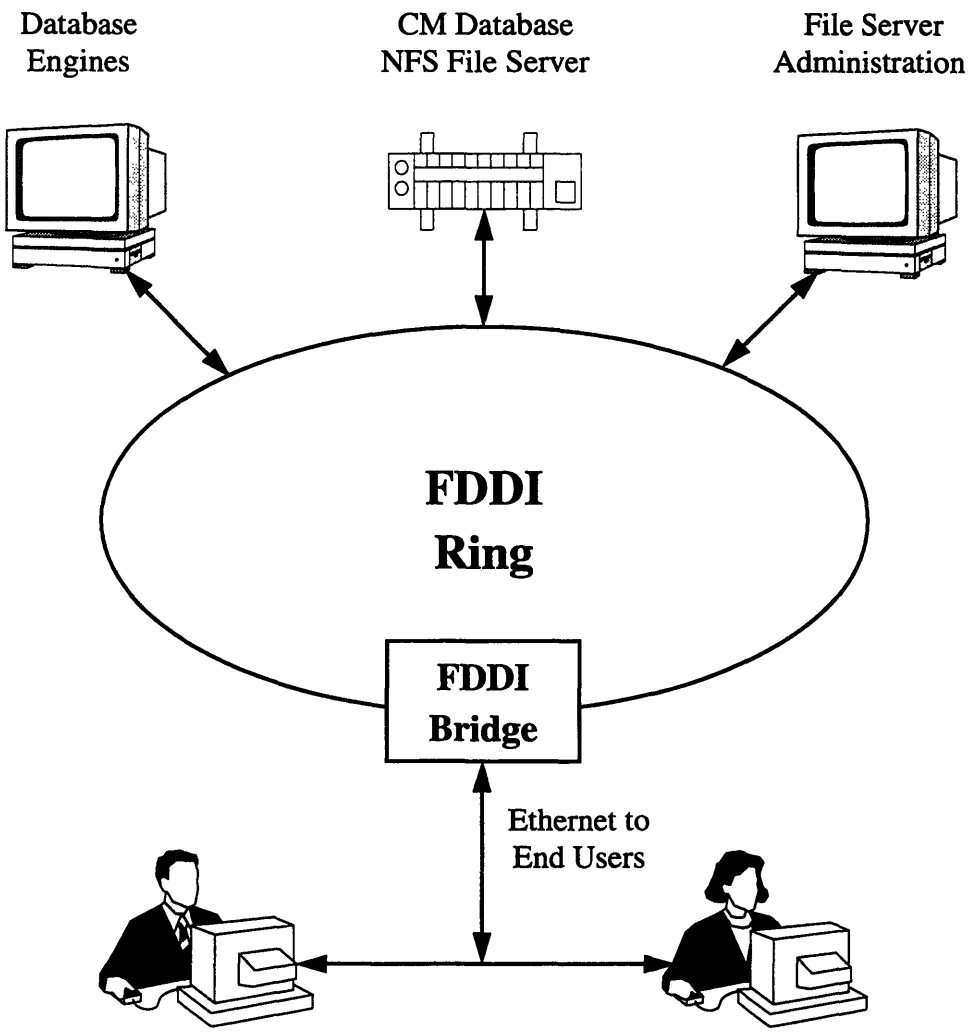

Figure 2.2: CM Hardware System

to-Ethernet bridge. The FAServer is a RAID [12] array, which is an array of disk drives containing some amount of redundant information in order to improve availability in the event of failures. The FAServer is a level 1 RAID array, in which an extra parity disk is maintained to recover from failure of any single disk. Through its proprietary file system, the FAServer makes a "snapshot" or on-line backup every 10 seconds to make the file system very reliable. Further, the system is backed up using the Epoch file system which maintains regular external backups to guard against a local failure [9].

Each version of each object will be stored only once in the database object pool. Multiple projects can link to and use a component without replicating it in the file system. This design enhances portability to multiple platforms. Creating new binary code versions from source code is as simple as NFS mounting the database archive and specifying a target machine and compiler for the General Make Facility [8]. Multiple projects with multiple target platforms, operating systems, and compilers can all compile from a single source, thus eliminating redundant and conflicting code.

2.3.2 General Make Facility

The General Make Facility, or genmake, is a collection of Unix shell[1] scripts which automatically determine which set of compiling tools to use and which host machine (i.e. target) to run them on to make a build for a collection of portable code. Genmake is used by the CM system to actually build a product. Genmake was originally the conception of Kyle McDonald, then a student intern at Draper, and the functionality and correctness of the package were improved upon by the author and Stu Roseman, the original technical supervisor of this work.

Genmake uses language-specific rules and regular expressions to parse source code and automatically generate dependencies from "include" references and entry points. By default, genmake defines the target to be the host type of the invoking user's system, but this can be overridden to make the target any arbitrary (but supported) host type. Support has been added for dynamically choosing which host machine to use for a particular build, based on the current load of available hosts, but, as of this writing, that feature was not complete.

---

1. The General Make Facility scripts are written for the Bourne Again Shell (bash), an sh-compatible shell written by the Free Software Foundation.

2.4 CM tools as Reuse Libraries

While the SCM system is an excellent base with which to facilitate reuse, with its version control, stability, and confidence, it is not a fully-functioning reuse library. With very small libraries, it is plausible for developers to know the contents of the library, but, after its holdings grow beyond some critical point, users will not be able to be wholly familiar with all collected components. At this point, a powerful search and browsing ability becomes necessary, which is not included in our chosen SCM tool. Additionally, new users of the system will be unfamiliar with the database's contents. One of the major challenges, then, is choosing an appropriate user interface to allow easy search and browsing of the database. One such solution, a World Wide Web-based reuse library, is described in the next chapter.

Chapter 3

Repository Construction

3.1 System Requirements

The requirements for the design of the Draper Reuse Library included both technical and resource-related requirements. While Draper is a research and development institution, its primary focus is not in software engineering research, but rather in applying knowledge toward real-world technical problems like satellite tracking and navigation systems. However, like any industry software practitioner, it needs to be able to harness the potential of advanced software engineering techniques to do its job more efficiently and reliably. Since the author was the sole person assigned to the project of the reuse library, and since a limited time frame was allotted, any solution had to be as simple as possible, and make use of as many freely available open systems as possible. Since investing in a software reuse tool program is a long-term investment with potentially little short-term gain, many companies at the beginning of reuse programs will have substantially similar economic requirements.

It is important to note here that there is really no widely-available, established standard software reuse library application, even in the commercial sector. There are still many debated representation issues, classification standardization attempts, and general disagreements in the reuse community about how best to support reuse with library tools. Some tools only add value for object-oriented systems, while still others are proprietary internal tools not available to academic researchers. Several companies have or are attempting to provide valuable and general reuse library applications, but as of this writing, no clearly successful effort has emerged.

3.1.1 General

From a technical standpoint, it would be desirable to be able to separate the individual components of a reuse library system into modular units that could be replaced as needs change and as new technologies are introduced. Thus, the library itself should be relatively independent of the user interface, and should be compatible with different CM tools. Because Draper is such a heterogeneous computing environment, any solution should not only be capable of supporting a wide variety of platforms, but also make it relatively easy to propagate changes to all the platforms. The system ought to be able to handle a wide variety of document formats, including multimedia formats, and be able to anticipate future file formats. Finally, especially until reuse has been adopted as a major managerial commitment, there will likely be little administration of the system, so its administration should be as automatic and invisible as possible.

3.1.2 Front End

The presentation of the user interface has traditionally been a major portion of the programming effort, while diverting attention from the main goals of the project and rarely breaking new ground in effective user-interface design. By spending much effort on the interface, it is quite possible for the developers to lose sight of the real purpose of the system, while failing to provide additional functionality or efficiency.

In order to reduce user-interface development cost, there are now at least three major interface standards for multi-platform applications. The simplest is to provide a text-only interface, over a telnet dialup, or something similar. The ASSET reuse library system, a federal government experimental reuse library[1], uses such an interface. In conjunction with a retrieval method like ftp, nearly all networked computers can make use of such a system. It has low computational overhead and serves a wide variety of platforms, but has

1. While retrieval of components requires membership on a telnet-accessible server, component information and general information on ASSET (including membership guidelines) is available on their Web server at http://source.asset.com/asset.html.

extremely limited user-interface possibilities, and is not very appealing from a human factors standpoint.

Another possibility would be an X-windows GUI running as an X-client on a central compute server and displayed on individual users' X-servers. However, the computational demand on a large centralized system would be great, considering the potential number of users in a large institution or network. The response time for users would not likely be reasonable. More importantly, the complexity of writing bug-free X-windows code is high, which makes it difficult to adhere to the requirement of cost-effectiveness. Further, there remains the security issue, as users would need to grant access to their display to the X-client, a prospect more unnerving on the Internet than in a closed commercial environment. At least one study has found that the intricacies of X-windows programming in reuse library projects is usually the cause of most system faults, rather than the stringent demands placed on the reuse library itself [13]. On the positive side, however, an X-windows interface provides a much richer collection of user-interface components and could potentially be fairly portable if used in conjunction with a multi-platform build tool like imake. It would, however, require separate builds for all the target platforms.

The third, and most promising, possibility, is a World Wide Web interface supporting the enhanced user interface known as "forms." The features of Web interfaces are explored more fully in Section 3.2. This solution addresses most of the fundamental requirements for our front end, with little additional work. It can handle a wide variety of documents, including multimedia. It runs on a wide range of PCs and workstations, and much of the functionality can even be reproduced in text mode with a browser called Lynx. User interface basics like text entry fields, checklists, and selection boxes are built in. The Web also provides a "gateway" interface so that information providers can run scripts on their servers to provide arbitrary content, including software components, text,

or images. By writing such scripts on the server side in a language like Perl [14], the database can be queried, search results parsed and formatted, and hypertext links to the references dynamically created. More importantly, the Web interface enforces a sharp distinction between the front and back ends of the application, making it easier to change the functionality without having to rework the user-interface. Difficult X-Windows programming can be replaced with a simple annotation language called HTML to quickly test and experiment with the interface.

3.1.3 Back End

There are many potential ways of organizing, browsing, and accessing the material in a repository. Four of the major options are a flat file system, Artificial Intelligence (AI) system, database management system (DBMS), or information retrieval (IR) system [15]. The flat file system is the simplest of all - the files are stored in essentially the same format in which they will be used, perhaps with compression. No particular browsing or searching capability is provided automatically, but some basic functionality could be established with standard pattern search utilities like grep, sed, awk, and Perl. For that matter, perhaps no pattern search would be necessary, so long as there were a full-time repository librarian responsible for categorizing the material and providing links in the user interface to the material. In small reuse systems, this sort of manual intervention with a single knowledgeable individual might be plausible.

AI systems are frequently knowledge-based systems that draw inferences based on logical statements and relationships, and use structures like frames and semantic nets to locate appropriate artifacts. LASSIE [16] is an example of a knowledge-based software reuse tool. While AI systems show promise, at this point in time they have not proven practical. The amount of work in semantically representing huge collections of documents is prohibitive [15].

DBMSs are well suited for highly structured data like records with attribute-value pairs of limited length, but are less suited for full text. The search is deterministic, in that fields either match or do not match the records contained in the database. With knowledge of required values for a few fields, relevant documents can be determined conclusively. DBMSs are especially suited for changing data, and don't require as much daily maintenance as IR or AI systems. However, response time can be long depending on the size of the database and the processing power of the server.

IR systems are distinguishable from DBMSs in that they retrieve documents probabilistically [15]. By using pre-computed indexes to aid in retrieval, the computational demands grow slowly with growing databases. Unfortunately, the overhead of the indexes is frequently comparable to the size of the original data in storage cost. They are well suited for finding keywords in huge collections of full-text documents with very fast response time. However, the retrieval may result in omissions as well as mistaken inclusions, since heuristic methods are used to determine the relevance of a document. IR systems can handle large document collections, much like DBMSs.

A hope of many researchers is to eventually combine the strengths of AI, DBMS, and IR systems in a practical hybrid system. To some extent, that wish has been fulfilled with freely available software packages like freeWAIS-sf, a member of the WAIS family of IR systems. This system will be described more fully in section 3.3.4. By combining the field nature of DBMSs for more accurate search with the fast text-search ability of IR systems, freeWAIS-sf provides a viable back end storage and retrieval system today. While freeWAIS-sf combines the strengths of two of the three major repository approaches, perhaps future systems may also incoporate the strengths of AI systems to create even more powerful tools.

Another aspect of reuse library research that has been of substantial interest in the reuse community is the choice of representation methods for classifying components. Some researchers hope to gain power from traditional library classification systems like faceted classification, as advanced in Ruben-Prieto Diaz's seminal Ph.D. thesis [17]. However, little experimental work has been done comparing the effectiveness of different representations. The one recent study that did include an empirical comparison of representation techniques found no statistical differences in search effectiveness in recall and precision among four representation techniques. The techniques tested were attribute-value, enumerated, faceted, and keyword [18]. Given that there is no proof at this point that any one works better than the others, it seems most prudent to use what is most straight-forward to implement, namely keyword and attribute-value. The results of the above study should, of course, be replicated by other researchers and in other conditions to validate their results, but that is not the intention of this project.

3.2 System Element: World Wide Web

The World Wide Web is one of the fastest growing information services on the Internet, with its simple and consistent interface, linking capabilities, flexibility, and intuitive graphical interface. It was developed at the CERN particle physics laboratory in Switzerland under the direction of Tim Berners-Lee, as a method for scientific researchers to share multimedia information over networks. Initial development focused on defining the HTTP client/server protocol, creation of a prototype server, and creation of the wwwlib programming library. After the developers released their code to the public in 1992, numerous organizations, but most notably the National Center for Supercomputer Applications (NCSA) at the University of Illinois in Urbana-Champaign, innovated on CERN's original designs [19]. NCSA's Mosaic Web browser quickly became an international standard, unchallenged for about a year until serious challenges from other parties surfaced

this year.

A World Wide Web browser presents a scrollable graphical interface to the user. The browser may display formatted text, images, and hypertext links. Clicking on the hypertext links with a pointing device (usually a mouse) can load a new screen ("page") or launch an external application to view a graphical image or animation, play a audio file, or display specially formatted text like Postscript documents. One of the chief advantages of using the World Wide Web for an application interface is that a new client for each platform does not need to be developed so long as a compliant Web browser already exists for the platform. Browsers exist for all major UNIX flavors as well as most popular microcomputers, including Windows PCs and Macintoshes. The information provider simply needs to program the server side, and the operation and presentation will all be substantially the same, although the browsers do have some flexibility on the specifics of the presentation, especially with logical styles.

Another major advantage of the Web is the ability to flexibly handle many different document formats. Each document returned by a Web server has a Multimedia Internet Mail Extensions (MIME) type associated with it, e.g. text/html or graphics/jpeg. Certain images can be displayed in-line by the browser, but the real power lies in the MIME typing mechanism. The browser associates these types with external applications that allow the browser to launch dedicated handlers for audio, video, graphics, and special text. Because the software development process frequently produces a wide variety of documents, from code and journal articles to graphical calling diagrams and voice annotations, it is useful to use a tool that can flexibly handle new formats as the tools used by the practitioners change.

3.2.1 HTML

Hypertext Markup Language (HTML) is the lingua franca of the Web, similar to Structured Query Language (SQL) in the database world. While the features of the language have expanded over the past few years, the core of the language is understood by all browsers. HTML is much simpler than TeX, Donald Knuth's document typesetting language, but is similar in that it has some features of a language but is more of a formatting method than a general-purpose language. Pairs of tags are placed around text elements to denote them as headings, bullet lists, logical and absolute character formats, and hypertext links. Certain images and even input elements can be included in-line, depending on the browser. The hypertext links are what interconnect the individual Web pages to each other. By clicking on a link on a page, the user may be transported to a different part of the page, a different page on the server, or a different server altogether, without regard to the actual physical location.

3.2.2 Web Gateways

The World Wide Web provides a "gateway" interface, which provides a method for Web clients to pass information to scripts residing on the Web server host computer. By calling these scripts instead of standard HTML pages, servers can provide much more diverse information to clients. While standard HTML lets the user choose from a variety of formatting options and media, it is still a static entity that typically needs to be manually updated. Sometimes, dynamically generated information is what needs to be presented. Further, while Web browsers can access a variety of services directly, all services are not supported by all browsers. Thankfully, with gateways, Web servers can be set up to run application programs, and return dynamic text or even dynamic HTML back to the client. Furthermore, by including a gateway, the browsers themselves don't have to be updated and recompiled to include support for new types of services.

Communications between the gateway and the server is accomplished via the Common Gateway Interface (CGI) protocol. The user's client inputs information through some sort of entry field (usually a form), and the client sends this input to the server via either environment variables or standard input. The gateway processes the input in an arbitrary way according to the particular application, and then takes some action, either changing the server's file systems in some way, firing off a non-interactive application, returning HTML output, or frequently some combination of all of them. The gateway program itself can be any arbitrary executable program or script. Scripts are most frequently written in C/ C++, Perl, tcl, C shell, or Bourne shell, but can be written in any arbitrary language [19]. Perl is a frequent choice because it is interpreted rather than compiled so it is easy to change on the fly, and it contains more powerful constructs than most of the other scripting languages. Also, many popular public domain packages for Web scripts are written in Perl, and many of these have become the basis of derivative works.

3.2.3 Forms

"Forms" refers to the more diverse types of user-interface elements that can be displayed on a Web client with just a few lines of HTML. Forms are a recent addition to the HTML standard, part of version 3.0, but they have been implemented already in most current browsers. They offer more varied input elements than just links, and frequently present the user several fields to fill with a single submit button, rather than just a single text-entry field. Supported input elements include text entry fields, password fields, scrollable text areas, checkboxes, radioboxes, pop-up menus, scrollable lists and push buttons. They are implemented in most browsers with the standard Motif widget set. Most sophisticated Web sites have a forms-based interface connected to a gateway on the server to offer dynamically generated content. Forms can also be useful for sending email and comments back to Web maintainers.

3.3 System Element: WAIS

The Wide Area Information Server (WAIS[1]) is a commonly-used full-text information retrieval system appropriate for time-efficient queries of large databases of documents of multiple formats. The searching functions over standard TCP/IP networks with the ANSI standard Z39.50 protocol [19]. While widely used for internet searching, WAIS can be a powerful tool in local search as well.

3.3.1 History

WAIS originated with the supercomputer company Thinking Machines, makers of the Connection Machine series. It was originally developed for use as an information service in collaboration with Dow Jones, Apple Computer, and KPMG Peat Marwick, as a "proof of concept" that commercial services could be built reliably on contemporary network technologies. WAIS consists of an indexer, which creates index files from arbitrary input files, and a server which takes queries from clients and using the indices, searches for keywords or expressions in the original input documents. WAIS also has the potential to incorporate something called relevance feedback, in which users can request additional documents similar to one which has already been retrieved [19].

There are at least three major strains of WAIS as of this writing: WAIS, freeWAIS, and freeWAIS-sf, each with their own attendant peculiarities. WAIS Inc.'s WAIS, developed partially by the original development team at Thinking Machines, is a commercial product while both freeWAIS varieties are freely available. FreeWAIS is a descendant of the original Thinking Machines WAIS and is supported by the Clearinghouse for Networked Information Discovery and Retrieval (CNIDR). FreeWAIS-sf is a variant of freeWAIS that was developed by Ulrich Pfeifer and colleagues at the University of Dortmund. All of the packages are fairly similar but freeWAIS-sf alone has the particularly useful ability to han-

1. pronounced "ways"

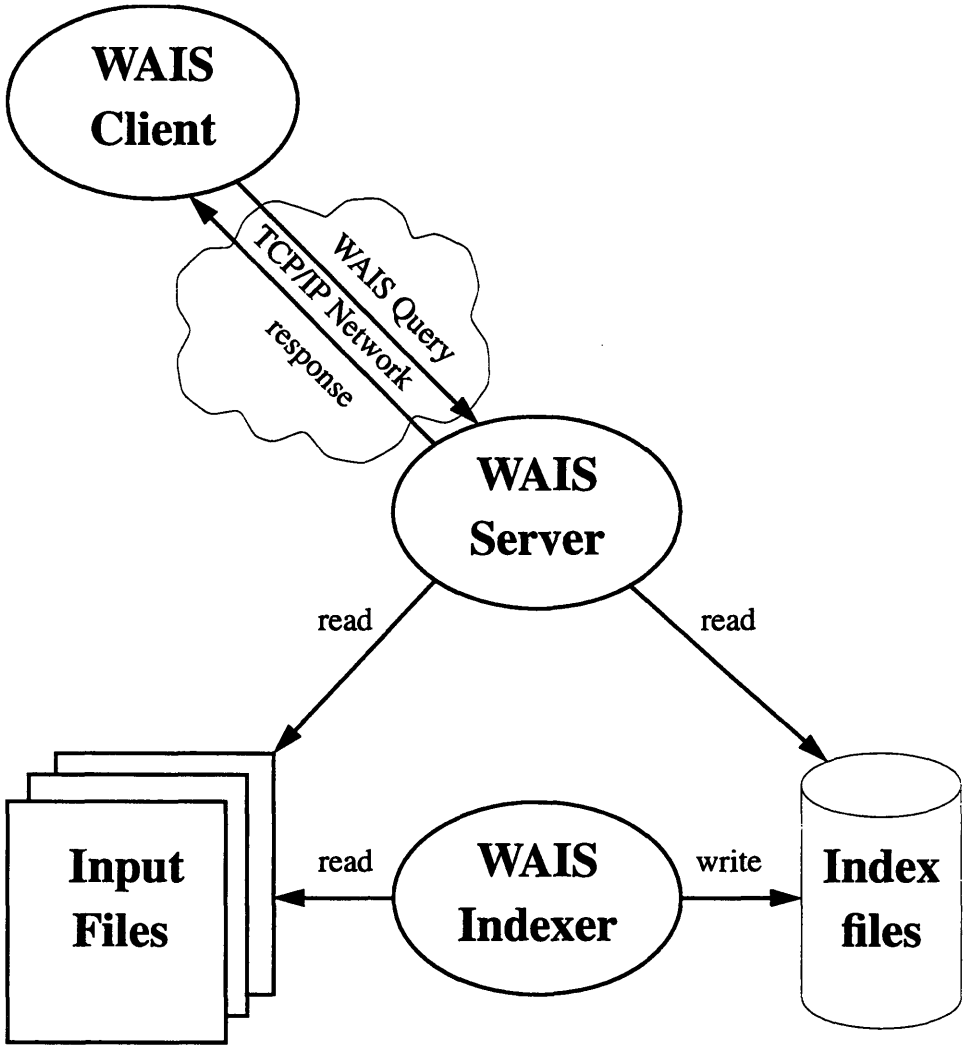

Figure 3.1: WAIS system structure

dle structured fields (explained further in section 3.3.4). The freeWAIS-sf package is rapidly gaining acceptance among Internet users, and, for many, is the package of choice.

3.3.2 WAIS System Structure

Although there are three common versions of WAIS, the code and structure of the systems are remarkably similar, and they can interoperate to a great extent. There are three main components in a WAIS system: indexer, server, and client. The basic system setup is illustrated in Figure 3.1, adapted from Liu [19].

The indexer reads a number of files of different formats and creates a variety of different index and support files or tables for each document, including a list of all words in the

documents, pointers to their locations, and so forth. The server reads the index files which refer to a particular database of documents. It then uses these index files to tell whether the keywords supplied by the client exist in the WAIS database and to determine which original documents contain the references. The server also includes a boolean logic parser to split the arbitrarily complex query into individual search requests, and it then combines the search results using boolean operators. The client may take a variety of incarnations, from a Perl script to a command-line interface to a graphical interface. In any case, the client composes a query in the ANSI Z39.50 format, passes it to the server, displays search results, and allows direct display of the original documents. The main client can somewhat intelligently display multiple document types since each document has an associated WAIS type, and each WAIS type can be connected to a separate default viewing client. Several versions of web browsers now include built-in WAIS client code, but this feature is not universal [19].

3.3.3 Index Files

The reason that WAIS can search so time-efficiently is that the index files are generated upfront. The indexing process is time-consuming, but not necessary to repeat with each query, unlike most typical word-processing search functions. The indexer parses the file differently depending on the type that the operator defines for the input file. All WAIS indexers allow the user to determine delimiters between documents in individual files, specify the part of the document which forms the "headline," or short description of the returned document, and decide whether to index the contents of the document. As an example, binary graphic images are typically indexed just by name and not by contents [19].

FreeWAIS-sf's ability to use structured fields allows for more complicated indexing decisions, detailed in section 3.3.4, but all forms share the same basic set of seven index

files, only two of which are human-readable. Despite the number of index files, they all contain only information about the original files, and not the full content itself. The original files must still be available to be returned to clients on retrieval. The system might perhaps have been designed to reconstruct the documents on the fly, but the time efficiency of WAIS is its chief advantage.

The seven common index files are a dictionary file, an inverted file, a document table, a filename table, a headline table, a catalog, and a source description. The dictionary file is an alphabetically sorted list of all words appearing in all documents anywhere in the database, with pointers into a position in the inverted file. When the server checks sources for a word, it looks in the dictionary file first to see if it exists in the database, and finds the appropriate position in the inverted file from which to search [19].

The inverted file contains pointers from each dictionary word to every document the word appears in, as well as the potentially multiple locations within the document where actual references occur. Further, each occurrence of a word has a *weight* associated with the reference. The weighting is based on factors such as where the word appears and how many times it appears compared to the overall size of the document. The server uses the weights from the inverted file to determine how close of a match has been found, in order to facilitate ranking [19].

The document table has a list of all the documents in a source, and also three pieces of associated information: a pointer into the filename table to determine the database file associated with the document; the document's place in the database file; a pointer to the headline file. The filename table has a list of all filenames used in making the source. The headline table has a comprehensive list of headlines. The catalog file is simply a list of all headlines along with document IDs. It is kept on hand for searches that fail to match any documents. This allows the operator to browse the list of potential documents in order to

provide prompts for more appropriate search keys. The source description contains general information about the source file and the server and was partially designed to permit future commercial use of WAIS software [19].

3.3.4 FreeWAIS-sf

FreeWAIS-sf, the version of WAIS developed by Ulrich Pfeifer and others at the University of Dortmund, Germany, is rapidly gaining popularity among information providers on the Internet. Its powerful enhancements to standard WAIS features, including structured field support, make it a good choice for reuse libraries like the one at Draper.

Standard WAIS systems are optimized for keyword search over the full-text of a document. In software reuse searches, more information about the needs of the reuser is typically available. This might include information about the needed language for code, code for a particular project, and so forth. Without any structure in the database to handle this additional information, there is much more potential for false matches and less probability of finding matches, because of the imperfect nature of the probabilistic weighting algorithm.

By adding the ability to search well-defined fields for keywords in addition to standard free-text search, freeWAIS-sf databases are much more likely to find appropriate matches in reuse libraries. This sort of field information might also be encoded in a standard relational database, but freeWAIS-sf offers the equivalent field search power of a database and moreover its precomputed indexes give the added ability to find individual keywords in thousands of full-text descriptions extremely quickly. Moreover, it is available free of charge and the source is publicly available so it can be understood and customized for particular needs.

Structured fields can be text, date, or numeric key-value pairs, and queries can be made using any or all fields and combined with complex boolean expressions, handled by

a parser in the server. Each individual field can independently turn on stemming and phonetic coding. Stemming allows many different words with the same stem (e.g. compute, computer, computing) to match the query, while phonetic coding allows "soundalike" words to be matched. These additional search options provide more tools to the user for finding appropriate components even with limited information. Instead of the C programming required to introduce new document formats in freeWAIS, the librarian must configure two filter format files using a simple regular expression specification language. These changes are all limited to the indexer and server, and so existing clients can query freeWAIS-sf databases as normal freeWAIS databases, or they can use the additional features.

3.3.5 SFGate

SFgate is a Perl CGI script also authored by Ulrich Pfeifer. While especially suited for freeWAIS-sf servers, it can connect directly to any freeWAIS server. The script is callable from a Mosaic forms interface, and can connect to an arbitrary number of databases, potentially on different servers, each of which can be picked by Mosaic forms checkboxes.

Using HTML, the reuse library maintainer can create one entry field on a searching page to match each field in the freeWAIS-sf database. A general keyword search field also exists to search over all fields of any structured indices as well as indices without structured fields. Potential reusers fill any or all of the fields with information with which they wish to restrict their search and then press a submit button. The information in the fields is sent via the CGI to the SFGate script. The script connects directly to the freeWAIS-sf server using the forms data to form a query using the structured fields. Typically, the data in all the completed forms is connected by an all AND (all fields must match) or all OR (at least one field must match) connection, but a separate text entry field can even support arbitrary boolean queries. Work is already underway on a graphical method of flexibly

connecting the fields in boolean queries. At the time of the Draper Reuse Library's creation, the script still had errors but was mostly functional. Through bug fixes from this author and other Internet contributors, the current product is fairly stable, and is undergoing more enhancements.

When the query has been completed, SFgate writes dynamic HTML to return the headlines that matched the query in ranked relevance order, with hypertext links to those original documents. The user can select a file to view by simply clicking on the link. The headline step can also be skipped by merging all the documents together and "directly getting" the document immediately following the query. Further, the search results can be connected to filters to convert the output. Since the entire script is written in Perl, the source is visible and customizable. Unfortunately, relevance feedback, the feature which would allow users to request documents similar to one that has been retrieved, has not yet been incorporated in the script.

3.3.6 Integration with the Web

WAIS can be integrated with the Web in a variety of ways, each its own trade-offs. Some browsers, like Mosaic 2.0 for X-Windows, include WAIS client code to allow the browser to interact directly with a remote WAIS server. While this might be a useful feature if all users could be guaranteed to use the same browser, not all browsers support this feature. Since one of the advantages of the Web is the ability to provide a single service to multiple platforms, taking advantage of this feature means either dictating which browser users may employ, or handling each kind specially [19].

There are a few publicly accessible Web-to-WAIS gateways, including one run by CERN, and another by NCSA. These gateways are Web servers that function as WAIS clients, and return the results of the WAIS query in HTML for the original Web client. Unfortunately, these gateways are only accessible to machines directly on the Internet, or

with special proxy servers. Commercial institutions, especially those like Draper with serious security demands, typically have firewalls that prevent company machines interacting with external ones except through a carefully controlled gateway, making this approach unusable.

There is also a commercial gateway between the Web and WAIS run by WAIS, Inc. called WAISgate. WAISgate is accessed by users with forms-capable Web browsers and WAIS databases to do fielded search with a variety of appealing features. Unfortunately, as a commercial entity, the service costs money, and its performance depends on your network connection and many other features that may hard to control.

The most effective solution, for the time being, is to use a CGI script on your own Web server to take forms input, convert it to a query, and convert the result to dynamic HTML. While numerous people have recently worked on similar systems, the SFGate script by Ulrich Pfeiffer meets most basic needs for Web-WAIS interaction. It contains most of the desirable features of the commercial WAISgate, including ability to connect to multiple remote and local databases and fielded search. Further, the source is public, and can be customized for particular uses, such as the Draper Reuse Library. The one major drawback to this approach applies only to UNIX users with non-superuser access who wish to run such systems. Since network-accessible servers require access to TCP/IP ports to provide services and secure ports are only accessible to superusers, such library maintainers may have to sacrifice some security to run such this sort of system. The only risk is that someone may listen to the WAIS port and capture the documents being retrieved. However, the system is not accessible to users outside of Draper due to the Lab's firewall protection, and in any case the library is intended to be a Lab-wide public system with freely accessible materials.

3.4 System Description

3.4.1 Overview

The first issue that needed to be addressed is which data should be migrated from the CM database to the WAIS database. While Draper work includes a number of classified projects, none of them will be stored on the CM system due to their stringent security requirements.[1] Therefore, the decision needed to be made whether to include access to all products in the Released state in the CM database, or only select components of exceptional reusability. In order to meet the requirement of little system administration, it was at first tempting to simply repackage the Released products in the reuse library since all Released products could be extracted with a script that was automatically executed at some regular time interval or event-driven basis (such as promotion to Released). The CM system would ensure the Released code is stable and of high quality; however, without standards for demanding design generality, the utility of components in the library may become substandard. The administrative savings of such an automatic system would be pointless if the reuse library holdings were not truly reusable for such components would merely discourage developers from using the system. Because of the existence of Released components that are not truly reusable, the current prototype system relies on a library administrator to determine which components to include in the library.

This judgement of reusability requires a separate human certification process as there are no automatic tools that can make sound judgments on software reusability. Since Draper already plans to establish a Software Change Control Board (SCCB) to judge the worthiness of change requests, perhaps a Software Reusability Board (SRB) could be established to judge a component's reusability. Another state in the CM process, called Reusable, might be established to succeed Released. The Reusable state would denote

1. Classified projects are, of course, still free to draw from the library for their work.

components that not only meet customer requirements for their particular application, but also are general enough to be useful to others. In this way, components could be added to the reuse library using an automatic script and less library system administration would be possible, as all Reusable components would be certified as such and could be reasonably trusted to be so.

The WAIS server uses its indices to check for matches against queries, but it still requires direct access to any files to be served to the client. Therefore, any components in the library system need to be migrated from the CM database into the ordinary file system, in order to present them to the client. This requires a duplication of some of the holdings of the CM tool, but only of those components in the database designated as Reusable.

Another major decision was how to associate related files in a single unit for presentation to the client. Although WAIS has a "multi" feature that allows a single match of a document to return other associated files, there is a major limitation. All associated files must have the same stem name and differ in filename only with respect to the extension, so files like project.doc, project.jpeg, and project.html could easily be retrieved as a group. While this is sufficient for information providers who simply want to explicitly link related information, the artifacts of software development ought to have descriptive names during the development process and cannot be limited in this way. One way around this would be to put all the source code and compiled library archives in a single ".tar" archive, while requiring all the documentation artifacts within the library to be given the project name with appropriate suffixes. You would have a project.tar, project.spec, project.design, etc. When the search criteria listed matched any of the supporting documentation, all the files would be retrieved. This was the solution ultimately adopted.

The final decision was which files of a product need to have their contents indexed. Since most source code is going to consist primarily of internal identifiers and language-

specific reserved words, there is little likelihood that indexing the full source code will be useful. The only part of the source code likely to be useful to searchers will be the name of the function. Many projects at Draper have adopted a convention in which the name of the file has been standardized as the name of the function or data abstraction. Therefore, by indexing only the name of the file, we additionally have the name of the function or data abstraction, the portion of the source code most likely to be useful for queries.

Each file type has an associated ".fde" and ".fmt" format filter to parse the source documents into the structured fields during the indexing process. Using freeWAIS-sf's regular expression-based description language, a separate filter was created for each type by the librarian. Special field information that would be useful to reusers was added to the CM database model in contracted customization work on the configuration management tool. A Perl script was created which queried the CM tool with regard to the additional fields of a particular project which wrote the data to a special extra file for input to the WAIS indexer.

The basic structure of the reuse library system is illustrated in Figure 3.2., which is adapted from a similar figure in Liu [19]. Using the Web browser, the reuser fills in a form on a "search page" with any desired field information, as well as the general keyword search field. After the user presses the submit button, this information is passed to the Web server through the forms interface. The server in turn submits this information to the SFgate via the CGI interface. SFgate uses this data to formulate a WAIS query and passes it to the WAIS server. The WAIS server returns references which match the search criteria back to the SFgate script. The script dynamically writes an HTML page with hypertext links to the original documents returned by the WAIS query, and returns the HTML to the Web server, which passes it back to the browser.

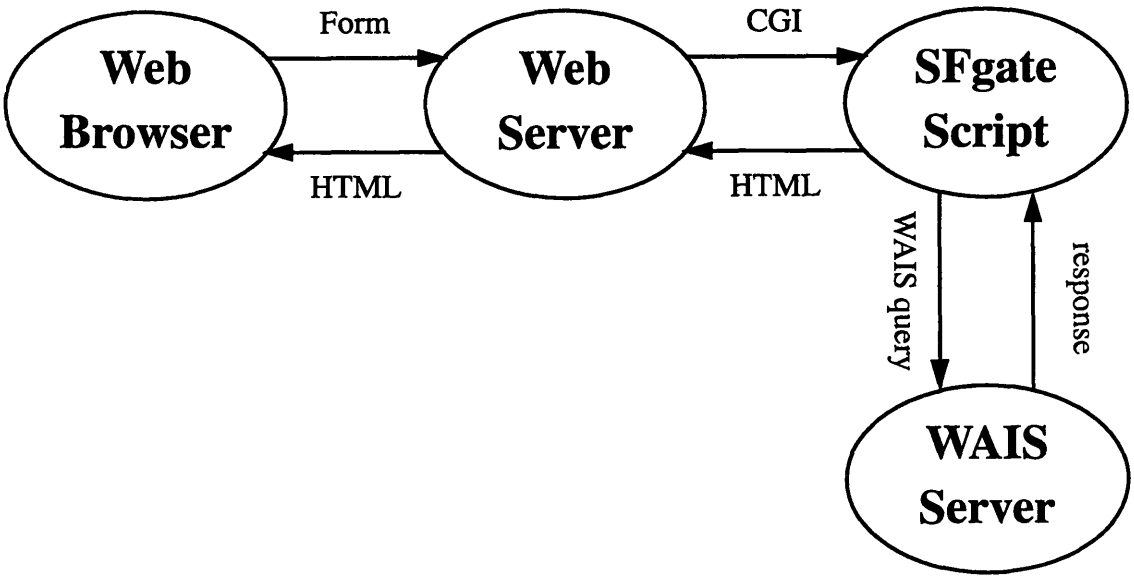

Figure 3.2: Reuse Library System

For users with an interest in retrieving components that might not yet be certified, a separate Web page was established to make CM database queries. Users can employ the page to get leads on components which might be reusable, but have not yet been certified by the librarian. This method only gives location information ("metadata"), and requires the user to extract the project from the CM database. In this way, users can be confident that artifacts extracted from the reuse library meet some minimum standards for reusability, but more advanced users can still effectively browse other library holdings.

3.4.2 Integration with CM System

Some of the field information required for the reuse library already existed as fields in the standard CM database, but some fields were added specifically to support the functions of the reuse library. Those fields are summarized in Table 3.1.

Attribute	Value
Author name	Primary author
Author email	Email address
Maintainer name	Person supporting component
Maintainer email	Email address
Source language(s)	e.g. C, Ada, FORTRAN
Keywords	From standard list
Abstract	Paragraph about purpose of package
Reliability	Either Excellent, Good, OK, or Weak
Age/Stability	Either Stable, Old, Research, Experimental, or Superseded
Size	source lines or kilobytes
Platforms supported	e.g. Ultrix, SunOS, Solaris, MS/Windows
URL	http://
Standards Supported	e.g. ANSI-C, POSIX.1, TCP/IP

Table 3.1: Fields added to CM database

Some of these adopted fields were inspired by the classification scheme of the Netlib repository[1].

3.4.3 Library composition

While sharing of truly reusable code is the obvious goal of a reuse library, there are problems with this approach in practice. In some cases, either the portion of a system which can be implemented with reused code is small, or specifications are hard to satisfy, or performance is suboptimal. Many cases where effective reuse exist are in narrow, well-

1. See Appendix A for information on locating Netlib and more detail about the full classification scheme.

understood domains like numerical computation [6]. Despite the drawbacks, libraries can remain useful since reuse of design is still possible in cases were exact implementation details are incompatible. In fact, design reuse has substantially fewer difficulties than code reuse. Much more of the information can typically be freely reused without the technical incompatibilities of source and object code interconnection.

Since the reuse library contains links to all the software development artifacts, including design documents and specifications (and not just code), the reuse library can help a developer to browse previous efforts with a higher potential of understanding the components. Those code parts which are still compatible can be copied, while those less so can still be instructive through design documentation. The hypertext nature of the Web in particular allows these related but separate documents to remain a logical whole.

Chapter 4

Summary and Future Work

4.1 Assessment and Limitations

The adoption of a stable software configuration management system and development of a prototype software reuse library tool are fundamental first efforts toward effective reuse and efficient software development. However, there is still much to be done before real benefits can be claimed.

It is fundamental to realize that effective systematic reuse is a multi-faceted endeavor, involving organizational issues, effective measurement (discussed in section 4.2), technological issues, and economic considerations. Because of the costs involved in establishing reuse programs, individuals are not likely to be able to achieve systematic reuse without support from management.

While establishment of a reuse library is a necessary condition of a reuse program, it is not a sufficient one. One of the dangers in establishment of a reuse library is that developing and maintaining the library consumes all the resources of the reuse program [20]. A successful reuse program requires more than just a common repository for old work. It requires design for reuse, transition to more maintainable and readable coding practices, adequate testing, domain understanding, and adequate documentation.

One of the major limitations of reuse libraries is the fact that they must usually go through a progression of phases before they actually contain useful artifacts. Frequently, the first stage is one of very few parts, a "library without books." At this point, there is very little incentive for potential reusers to even access the holdings. The next stage can often be one of many parts, but of low quality. By the use of SCM, Draper hopes to largely skip over this stage. The next stage is often one of many parts with little use [20]. This is a

more fundamental problem, for making the library effective ultimately has more to do with designing for reusability and proper documentation than it does with technical aspects of the library.

4.2 Quantifying Reuse

Anecdotal evidence indicates that software reuse might be a truly valuable practice in terms of increased confidence, economic savings, and improved productivity. However, hard evidence is still lacking. The amount of empirical data on actual costs and benefits of software reuse is insufficient [4]. As in all engineering research, measurement is paramount, for that which cannot be measured cannot be adequately controlled. For Draper's reuse program to be deemed a success, a quantification of the program is necessary, including generation and analysis of metrics, computation of economic benefits, and establishment of incentive systems.

4.2.1 Metrics

The most telling measures of reuse are taken from actual completed projects, rather than just library statistics. One of the fundamental metrics of reuse is the reuse level, the ratio of reused to total components in a system. Reused components include those both external and internal to the organization. This level is affected by managerial, legal, economic, and technical factors [4].

One of the real difficulties in analyzing commercial reuse metrics is that companies frequently only report successes - they become research papers - while failures are usually swept under the rug. One company, Hewlett-Packard, has publicly released its findings based on metrics of reuse benefits for two particular programs. They claim defect reductions of 51% and 24% as measured in defects per thousand non-comment source statements (defects per KNCSS). They claim productivity increases of 57% and 40% as measured by KNCSS per month when compared to "similar projects." [21] Such statistics

must be looked at with some measure of suspicion. To start, the measurement of similarity in project type is quite subjective, and further, different personnel with different levels of productivity are involved. Moreover, there is the danger that in reusing code, large blocks of partially unused code are inadvertently included in the count. Finally, H.P. reports time to market reduction of 42% and "not available." [21] Again, the measure of reduced time to market is fairly subjective, since the project is only done once, and the statistic for comparison is "estimated production time without reuse." This estimate of the time it would take without reuse is essentially computed from the equally suspect productivity increase statistic, and is therefore not very independent.

Other important indicators are what extra cost goes into designing software to be reusable and what the cost of integrating reusable products is as a percentage of what it would cost to create a new one. These are largely affected by the domain. For the cost of design, figures ranged from an estimate of nearly five times the ordinary cost in a study of a menu- and forms-management system, to an 11% increase in a graphics firmware project. For the relative cost to reuse, the figures ranged from 10 to 63% [21]. The decision to design for reusability or to reuse code should take into account the figures of its own domain.

While the metrics compiled by H.P. should be viewed with some caution, they are a step in the right direction - an attempt to measure that which is claimed to be beneficial. While comparing single projects to other single projects (real or hypothetical) can be rather subjective, a historical record of projects should in the long run provide a statistical basis to give more confidence in our assertions about reuse benefits. Perhaps if metrics were generated at the module or subsystem rather than at the project level, comparable classes of products would be easier to determine. The best metrics are those which are independent of other indicators and which involve the fewest value judgments, such as defect density. Reuse level is important to quantify, but since reuse usually involves some

form of modification, it is important to classify and track the amount of modification to determine whether extensive modifications are required when incorporating components that are claimed to be sufficiently general. The number of lines added, deleted, and modified might be useful metrics to determine the amount of work involved in adapting reused components. Any scientific assertion of actual benefits from reuse is on shaky ground without metrics.

4.2.2 Economic Considerations

Software reuse is as much a business decision as a technical decision. Organizations considering implementing systematic reuse programs would be wise to consider the expected economic return on investment (ROI) before inflating hopes that reuse will single-handedly ensure the productivity of their software development teams. While in the long run it is hoped that reuse programs will ultimately save money, organizations should be prepared for several years when reuse is not cost-effective before seeing any return. Smaller companies with less resources may be better served with smaller capital investments in reuse programs.

Hewlett Packard also attempted to measure the economic return on investment of its two studied divisions. They used the net-present-value method, which "takes the estimated value of reuse benefits and subtracts from it associated costs, taking into account the time value of money." Considering all start-up and maintenance costs, H.P. claims to have reaped R.O.I.'s of 410% and 216% in terms of savings divided by cost. They claim to have broken even on the startup costs in the second and sixth years [21].

4.2.3 Incentive Systems

In order for a reuse library or any reuse program to be successful, widespread participation by organization members is required. While the intrinsic simplification of programming work may provide some incentive, organizations may also wish to examine incentive systems to help populate reuse libraries with useful components. Many research-

ers have noted the importance of rewarding both sides of the reuse relationship: suppliers and consumers [22].

These incentives for successful reusers or producers may include formal recognition, dinners for two, more powerful computer equipment, cash prizes, or even a regular system of compensation for reuse. A few caveats for incentive systems designers must be recognized. While reuse is a desirable activity, immediate schedule needs should be the highest priority and an incentive system can misdirect motivation. Also, the external nature of such incentives may cause the reuse to cease when the incentives inevitably do. Finally, those rewarded with incentives may have already been reusing software effectively before the system, and there is no guarantee that modest incentives will change the behavior of less skilled reusers [20].

While incentive system designers should be careful, the incentives can nevertheless play an important role in the transition to a reuse-centered development methodology. In the necessary phase of library population, incentives can be useful to provide motivation for producers to jump start the library and consumers to draw from it. After reuse is a firmly established tenet of the organization's design philosophy and becomes an expected and not unusual practice, such incentives will probably have lost their usefulness. At this point, it may simply be more useful for the organization to consistently recognize and promote the best reusers and not similarly reward software developers that produce unreusable (and therefore lower value) components.

4.3 Conclusions

Software reuse attempts to leverage past workproducts for integration in new projects thereby to provide more robust systems at less cost. Effective software configuration management systems become important to this reuse effort by increasing confidence in code

stability. And, once the number of reusable artifacts in a collection reaches a critical point, a software reuse library becomes necessary as well for it provides browsable access to large collections of reusable components that would be too large for any one person to fully utilize.

By using freely available open systems, a Draper prototype which allows a basis from which to foster reuse was developed at extremely low cost using commonly available tools. While source code libraries have well-known limitations, providing access to full design documentation makes it possible to reuse design and still recover value even when the software component is incompatible with the system being built.

However, establishment of a reuse library is only a preliminary step on the road to achieving systematic reuse. A multi-faceted program with management support is necessary to truly achieve added value from the practice of reuse. Further, without quantitative measurement, claims of improvement and productivity are merely speculative. The elusive productivity gains sought by software reuse are attainable, but that can only be fully realized with a concerted multidisciplinary effort and a greater focus on satisfying longer-term goals.

Appendix A

Useful Internet Sites

A.1 WAIS

- WAIS, Inc.

  `http://www.wais.com`

- WAIS for UNIX, Technical Description

  `http://wais.com/company/Tech_TOC.html`

- WAIS and HTTP Integration

  `http://wintermute.ncsa.uiuc.edu:8080/wais-tutorial/wais-and-http.html`

- FreeWAIS-sf

  `http://ls6-www.informatik.uni-dortmund.de/freeWAIS-sf/README-sf`

- SFgate

  `http://ls6-www.informatik.uni-dortmund.de/SFgate/SFgate`

- CNIDR Home Page

  `http://cnidr.org/welcome.html`

- Mosaic And WAIS Tutorial

  `http://wintermute.ncsa.uiuc.edu:8080/wais-tutorial/wais.html`

- Wais and WWW pointers

  `http://www.cs.vu.nl/~anne007/waissearch/pointers.html`

A.2 Software Engineering

- Software Reengineering Web Home Page

  `http://www.erg.abdn.ac.uk/users/brant/sre/`

index.html

- WWW Virtual Library - Software Engineering

  http://ricis.cl.uh.edu/virt-lib/soft-eng.html

- Software Engineering Institute Information Server

  http://www.sei.cmu.edu/

- Model-Based Software Engineering

  http://www.sei.cmu.edu/SEI/topics/model_base_sw/overview/MBSE.html

- Computers:Software:Software Engineering

  http://akebono.stanford.edu/yahoo/Computers/Software/Software_Engineering/

- Software Engineering Laboratory

  http://fdd.gsfc.nasa.gov/seltext.html

A.3 Software Reuse Libraries

- ARPA STARS Program

  http://www.stars.ballston.paramax.com/index.html

- RBSE Program

  http://rbse.jsc.nasa.gov/eichmann/rbse.html

- Electronic Library Services and Applications (ELSA) Lobby

  http://rbse.mountain.net/ELSA/elsa_lob.html

- ASSET - Asset Source for Software Engineering Technology

  http://source.asset.com/asset.html

- Netlib Repository at UTK/ORNL

  http://www.epm.ornl.gov/msr/

- Netlib Index File Format

  http://www.ncsa.uiuc.edu/SDG/Software/XMosaic/

- HPCC National Software Exchange (NSE)

  `http://http://www.netlib.org/nse/home.html`

- Software Repositories

  `http://http://www.netlib.org/nse/software_rep.html`

- MORE: A Case Study in Reengineering a Database Application for the Web

  `http://rbse.jsc.nasa.gov/eichmann/www-f94/reeng.html`

A.4 World Wide Web

- Tools for WWW providers

  `http://info.cern.ch/hypertext/WWW/Tools/Overview.html`

- World-Wide Web server software

  `http://info.cern.ch/hypertext/WWW/Daemon/Overview.html`

- The Common Gateway Interface

  `http://hoohoo.ncsa.uiuc.edu/cgi/overview.html`

- A CGI Programmer's Reference

  `http://www.halcyon.com/hedlund/cgi-faq/`

- NCSA httpd Overview

  `http://hoohoo.ncsa.uiuc.edu/docs/Overview.html`

- SCG Script Archive

  `http://iamwww.unibe.ch/~scg/Src/`

- NCSA Home Page

  `http://www.ncsa.uiuc.edu/`

- Mosaic for X version 2.0 Fill-Out Form Support

  `http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/fill-out-forms/overview.html`

- Letting Users Search Your Web Pages

```
http://www-rlg.stanford.edu/home/jpl/
websearch.html
```

A.5 Perl

- UF/NA Perl Archive

```
http://www.cis.ufl.edu:80/perl/
```

- Index of Perl/HTML archives

```
http://www.seas.upenn.edu/~mengwong/perlhtml.html
```

- Perl Database scripts at ftp.cis.ufl.edu

```
ftp://ftp.cis.ufl.edu/pub/perl/scripts/db
```

- Forms in Perl

```
http://www.seas.upenn.edu/~mengwong/forms/
```

- CGI Form Handling in Perl

```
http://www.bio.cam.ac.uk/web/form.html
```

A.6 Harvest

- The Harvest[1] Information Discovery and Access System

```
http://harvest.cs.colorado.edu/
```

- Glimpse[2] Working Group - University of Arizona, CS Dept.

```
http://glimpse.cs.arizona.edu:1994/
```

---

1. Harvest is an integrated set of tools to gather, extract, organize, search, cache, and replicate relevant information across networks. While still somewhat experimental, it is a promising competitor to WAIS and claims more flexible searching capability with smaller indices.
2. Glimpse is the main search engine in Harvest.

Appendix B

Acronyms

AI - Artificial Intelligence

ANSI - American National Standards Insitute

CASE - Computer-aided Software Engineering

CERN - European Laboratory for Particle Physics

CGI - Common Gateway Interface

CM - Configuration Management

CMM - Capability Maturity Model

CNIDR - Center for Networked Information Discovery and Retrieval

DBMS - Database Management System

FDDI - Fiber Distributed Data Interface

FTP - File Transfer Protocol

GUI - Graphical User Interface

HTML - Hypertext Markup Language

HTTP - Hypertext Transport Protocol

IR - Information Retrieval

KNCSS - Thousand Non-comment Source Statements

MIME - Multimedia Internet Mail Extensions

NCSA - National Center for Supercomputing Applications

NFS - Network File System

PCR/ECR - Problem Change Request/Engineering Change Request

RAID - Redundant Array of Independent Disks

RCS - Revision Control System

ROI - Return on Investment

SCCS - Source Code Control System

SCM - Software Configuration Management

SEI - Software Engineering Institute

SQL - Structured Query Language

TCP/IP - Transmission Control Protocol/Internet Protocol

URL - Uniform Resource Locator

WAIS - Wide Area Information Server

References

[1] B. Boehm. *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs, N.J., 1981.

[2] E. Horowitz and J. B. Munson. "An Expansive View of Reusable Software." In T. J. Biggerstaff and A. J. Perlis, editors, *Software Reusability: Volume I*, chapter 2. ACM Press, New York, 1989.

[3] D. Kaspersen. "For Reuse, Process and Product Both Count." *IEEE Software*, September 1994.

[4] W. B. Frakes and S. Isoda. "Success Factors of Systematic Reuse." *IEEE Software*, September 1994.

[5] M. C. Paulk et al. "Capability Maturity Model for Software, version 1.1." Technical Report CMU/SEI-93-TR-24, Software Engineering Institute, Carnegie Mellon University, 1993.

[6] T. J. Biggerstaff and C. Richter. "Reusability Framework, Assessment, and Directions." In T. J. Biggerstaff and A. J. Perlis, editors, *Software Reusability: Volume I*, chapter 1. ACM Press, New York, 1989.

[7] T. J. Biggerstaff and A. J. Perlis, editors. *Software Reusability: Volume I and II.* ACM Press, New York, 1989.

[8] S. Roseman and R. Medeiros. "Software Configuration Management Process Definition." Draper Internal Document ESD-94-468, 1994.

[9] J. McLachlan. "Quick Guide to Draper Lab Standard Configuration Process." Draper Internal Document, 1995.

[10] Continuus Software Corporation, Irvine, California. *Command Reference*, 1994.

[11] D. Hitz. "An NFS file server appliance." Technical Report TR01, Network Appliance Coporation, 1993.

[12] D. Patterson et al. "A Case for Redundant Arrays of Inexpensive Disks (RAID)." In *ACM SIGMOD 88*, Chicago, June 1988.

[13] D. Eichmann et al. "Integrating Structured Databases into the Web: The MORE System." First International Conference of the World Wide Web. http:// rbse.jsc.nasa.gov/eichmann/www94/MORE/MORE.html.

[14] L. Wall and R. L. Schwartz. *Programming Perl.* O'Reilly and Associates, Sebastopol, California, 1990.

[15] W. B. Frakes and R. Baeza-Yates, editors. *Information Retrieval: Data Structures and Algorithms*. Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[16] P. Devanbu et al. "Lassie: A Knowledge-based Software Information System." *Communications of the ACM*, May 1991.

[17] R. Prieto-Diaz. *A Software Classification Scheme*. PhD thesis, University of California, Irvine, 1985.

[18] W. B. Frakes and T. P. Pole. "An Empirical Study of Representation Methods for Reusable Software Components." *IEEE Transactions on Software Engineering*, August 1994.

[19] C. Liu et al. *Managing Internet Information Services*. O'Reilly and Associates, Sebastopol, California, 1994.

[20] J. S. Poulin. "Populating Software Repositories: Incentives and Domain-specific Software." *The Journal of Systems and Software*, September 1995. To Appear.

[21] W. C. Lim. "Effects of Reuse on Quality, Productivity, and Economics." *IEEE Software*, September 1994.

[22] B. H. Barnes and T. B. Bollinger. "Making Reuse Cost-effective." *IEEE Software*, January 1991.