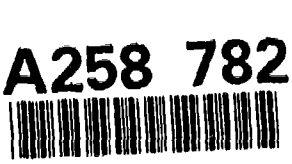


NPS-AS-93-006

NAVAL POSTGRADUATE SCHOOL

Monterey, California

A Clearinghouse for Software Reuse: Lessons Learned from the RAPID/DSRS Initiatives

Tung Bui
James C. Emery
Gerald Harms, Tina VanHook
Myung Suh

October 1992

Approved for public release; distribution is unlimited.

Prepared for: Director of Information, OASI (C3I),
Washington, D.C. 20301-3040

NAVAL POSTGRADUATE SCHOOL
Monterey, California

RADM R. W. West, Jr.
Superintendent

Harrison Shull
Provost

The report was prepared for the Director of Defense Information, OASI (C3I), Washington, D.C.. This research was funded by DoD Washington Headquarters Services, IAD, The Pentagon, Washington, D.C.

This report was prepared by:

Tung X. Bui
Dept of Admin. Sciences

Reviewed by:

David R. Whipple, Chairman
Department of Administrative Sciences

Released by:

Paul J. Marto, Dean of Research

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 1992	Technical Report, 1992

4. TITLE AND SUBTITLE
A Clearinghouse for Software Reuse: Lessons Learned from the RAPID/DSRS Initiatives

5. FUNDING NUMBERS
MIPR DXAM 20001

6. AUTHOR(S)
Tung X. Bui, James C. Emery, Gerald Harms, Myung Suh Tina VanHook

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES)
Administrative Sciences Department
Naval Postgraduate School
Monterey, Ca 93943

8. PERFORMING ORGANIZATION REPORT NUMBER
NPS-AS-93-006

9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Director of Information
OASI (C3I)
Washington, D.C. 20301-3040

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Software Reuse, Information Engineering, Domain Analysis

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This study reports the lessons learned from the recent establishment of the Defense Software Repository Service (DSRS) to provede government agencies and authorized contractors. with reusable software components (RSCs). DSRS is based on a pilot operation initiated by the Army's Reusable Ada Product for Information System Development (RAPID). The study concludes that a clearinghouse is a necessary condition for software reuse in DoD. To become a software productivity multiplier, the clearninghouse must populate its repository with quality RSCs, and incentives for reuse must originate from high-level DoD management.

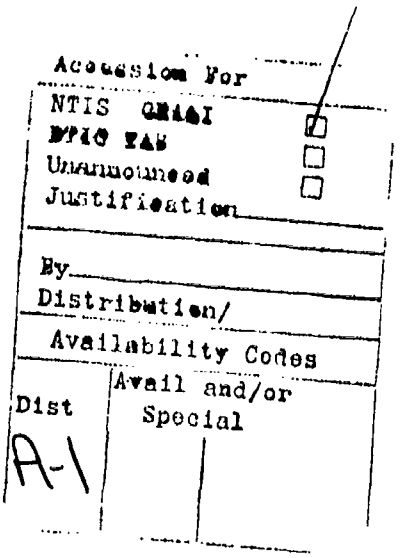
14. SUBJECT TERMS	15. NUMBER OF PAGES
	48
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

CASE STUDY SERIES
ON IMPLEMENTATION PRACTICES OF
MANAGEMENT INFORMATION SYSTEMS
IN THE DEPARTMENT OF DEFENSE

A Clearinghouse for Software Reuse: Lessons Learned from the RAPID/DSRS Initiatives

Tung X. Bui
James C. Emery
Gerald Harms
Myung Suh
Tina VanHook



Department of Administrative Sciences
Information Technology Management Curriculum
Monterey, California

October 1992

Acknowledgments

The authors would like to thank Mr. Paul Strassmann, Director of Defense Information, OASD (C3I), for sponsoring the Case Study Series on implementation practices of information systems in DoD. The case study on software reuse could not have been realized without the support of the Army Reuse Center and the DISA Center for Software Reuse Operations. We appreciate the support of Ms. Ginny Parsons at the Center for Software Reuse Operations.

We would also like to thank the faculty members and students at the Naval Postgraduate School who participated in the study and gave much support. Many thanks go to Professor Dani Zweig whose expertise in software reuse was very much needed. Also our thanks to LCDR Gilliam Duvall, USN; LT Cheryl Blake, USN; Professor Daniel R. Dolk, LT Greg Hayes, USN; Professor Sterling Sessions, and especially LT Christine Donohue and LT MaryJo Elliott who spent much of their time assisting in the format. We appreciate the support from all who helped us in this effort.

Table of Contents

I. Executive Summary

To be successful in meeting the challenges of today's rapidly changing world, Information Technology managers must become proficient at doing more with less. As information resources become strategic assets for many organizations, IT managers are asked to deliver more software, more quickly and with lower fault tolerances than ever before. As the demand for new software steadily increases, so does the demand for maintaining and improving the systems in operation.

DoD program managers have long recognized the benefits of software reuse to face the software challenge. In 1991, the Office of the Director of Defense Information set the following goals for software reuse:[1]

- 100% reusable data with an infinite life for data definitions
- More than 80% reusable code with more than a 20 year life on software elements
- 80/20 development/maintenance ratio
- Technology asset life two to three times larger than the technology innovation cycle.

The goal of implementing a DoD-wide clearinghouse of reusable components has culminated in the recent establishment of the Defense Software Repository Service (DSRS). Under the administration of the Center for Software Reuse Operations (CSRO), DSRS currently provides automated access to more than 1550 government or commercially owned/developed reusable software components. This repository is available to all DoD, other government agencies, and authorized contractors.

[1] Strassmann, 1991

The design and implementation of DSRS is based on a pilot operation initiated by the Army's Reusable Ada Product for Information System Development (RAPID). RAPID's goal was to establish a library of RSCs for application developers. To achieve this goal, RAPID has focused on defining reusable domains, searching for RSC donors, certifying RSCs, and creating a user-friendly library.

Though software reuse practices in DoD are still in their early stages of growth, the lessons learned from the RAPID and DSRS experiences provide some valuable insights:

- *A clearinghouse is a necessary condition for software reuse in DoD:* Given that RSC donors and recipients are separated by organizational and geographical boundaries, there must be a marketplace, such as DSRS, to facilitate the exchange of RSCs. Without a clearinghouse bridging the gap between donors and recipients, the software reuse effort would fall apart.

- *A clearinghouse is a productivity multiplier:* As the DSRS initiative has started to show signs of success, the demand and supply of RSCs are expected to increase. With a well-populated repository, the clearinghouse can be expected to multiply software development productivity.

- *The clearinghouse must populate its repository with quality RSCs:* The clearinghouse must ensure that RSC quality is not compromised. RSC users are reluctant to utilize components of uncertain quality developed by unknown sources; they want nothing but "error-free" RSCs. The clearinghouse must carefully weigh the benefits of accelerating the population growth of RSCs against the potential damage that might be caused if users experience problems with RSCs.

- *Domain analysis is the foundation for a successful software reuse program*: Domain analysis enables the clearinghouse to identilfy components that are commonly used within a given application domain and which therefore provide a higher payoff as a standardized RSC.

- *High-level DoD management must provide incentives for reuse:* A fully functioning repository populated with useful components is a necessary but not sufficient condition to promote a successful reuse program; some inducement to reuse software is required as well. To carry sufficient weight and visibility, these incentives must be promulgated from the highest authority within DoD.

II. Software Reuse: Doing More With Less

Reuse of prior work is certainly not a new concept. When electrical engineers set out to develop a new circuit board, they often reuse pre-fabricated integrated circuits, made available to them through technical catalogs, to accelerate the development process. Likewise, corporate executives rely heavily on standard text documents to prepare legal and business documents. In the software environment, with billions of lines of codes and thousands of implemented information systems, existing software modules could be recycled to avoid the exorbitant costs of "re-inventing the wheel."

Software reuse can be defined as the application of one or more previously developed software component(s) to a new system or to an expansion of an existing system. Software developers for large organizations are just beginning to appreciate the potential benefits of reuse and are now seriously integrating reuse practices into the software development process.

A. Productivity Gains with Reuse

Software reusability is viewed widely as a major opportunity for improving software productivity.[2] With software costs estimated to have reached $125 billion in 1991 for the U.S. alone, even a modest productivity gain through reuse translates into a tremendous savings. At the industry's present rate of growth, a 20% improvement in productivity is projected to result in a savings of $45 billion in 1995 for the U.S. alone.[3] Reuse can contribute to productivity gains in the following ways:

[2] Biggerstaff and Richer, 1987; Banker and Kauffmann, 1991

[3] Boehm, 1987

- *Achieve shorter development time*: When a developer is able to reuse previously generated software in a new application, he/she frees up assets that can be devoted instead to development of the system's unique modules. The savings in time and resources creates a development environment that is more responsive to the requirements of a dynamic world.

- *Increase software reliability*: Preexisting software, if it has been employed to any extent, has already been field tested and fine tuned. This offers the opportunity of deploying modules whose expected error rate is significantly lower than that of a newly developed module.

- *Ensure enhanced maintainability*: Reuse of well-structured and well-documented software will also lead to improved maintainability and portability in that alterations will be required only on the source component located in the central repository. With studies indicating a significant number of companies spending between 60 and 80 percent of their software dollars on maintenance, [4] this benefit may well generate the most significant dollar savings.

B. Beyond Code Reuse

To date, efforts to reuse software have been primarily focused at the code level. *Code reuse* is the simplest form of reuse. A reusable code component consists of functions, procedures, or packages. Once developed, a component is tested, certified, and stored in a repository so that it can be utilized by programmers for new software development projects.

Savings associated with code reuse can be realized in two ways: (a) each time a portion of code is reused, resources otherwise devoted to coding can be saved, and (b)

[4] Biggerstaff and Lubars, 1991

further savings are also realized by forgoing the testing of this pre-tested code component.

Reuse granularity is an important issue in setting reuse procedures. In general, the larger the size of the chunk of reuse code, the higher its "payoff." However, as the size of the code segment increases, it may become more difficult for the developer to identify a good match with specified functions. A large segment of code usually offers many functionalities. The greater the functionalities embodied in the segment, the more difficult it is for the developer to succinctly describe the segment's functional specifications. Larger components also tend to be more specialized or idiosyncratic, and are therefore less likely to be compatible with a given set of requirements specifications.

Reuse at the analysis and design level allows the developer to ignore coding details and focus on the computational intent of larger chunks of code that compose the system. Instead of looking at coding style, the developer can deal with functional specifications. Reusable components at higher levels include logical data models, functional descriptions, and diagrams (e.g., data flow diagrams or entity relationship diagrams). Although component reuse at the code level is better understood and by far the most prevalent form of reuse, there appears to be an acceleration in the reuse of the other types of software components.

For a given organization, there tends to be a family of application software that shares some common design characteristics. If these similar software design components could be reused, the developer would be free to concentrate on implementation of the unique functionalities of the software. Since design does not yet contain detailed decisions for implementation, a design component's potential for reuse is greater.[5] Reusable design should provide pointers to the appropriate pieces of reusable code, reusable test cases, and documentation. As a result, design reusability tends to provide much higher leverage than simply reusing code.[6]

---

[5] Lenz, 1987

[6] Biggerstaff and Lubars, 1991

Technically, existing software can be reused in a variety of ways. The spectrum includes sharing of code, algorithms, routines in application families, and subsystems.[7] Reuse can be applied to all phases of the software development life-cycle, including:

- Requirements specifications
- High-level design
- Detailed design
- Coding and unit testing
- Integrating testing
- Documentation
- Maintenance

RSCs can be either developed from scratch or extracted from public domain software, commercial off-the-shelf software, contractors, and government sources. A desirable reusable software component should include the following characteristics:

- *Flexibility*: The developer should be able to adapt/modify an RSC to fit in with the overall architecture of the software to be built. The smaller the component in terms of functionality, the more flexible it is but the less functionality it provides.
- *Expendability*: The developer should be able to tailor RSCs to specific requirements that might surface well after initial requirements were defined.
- *Portability*: The RSCs should be able to operate under multiple operating environments (physical hardware, operating systems, and runtime environments)
- *Language Independence*: The components above the implementation level should be programming language independent so that they are reusable in any programming language environment.

[7] Lenz, 1987

C. Domain Analysis: The Foundation for Reuse

The investment for an RSC is justified only if the expected benefit outweighs the cost. A dominant factor that determines the expected benefit of an RSC is how often the component is likely to be reused. If a particular software component is never to be reused, any effort to make the component reusable would be a wasted investment. In constast, a software component that would be reused frequently can yeild worthwhile benefits. The frequency with which a software component can be reused depends on whether the component's functionality is unique or common across a range of applications. For example, a reusable component providing date/calendar functions is likely be reused over and over again because this function is commonly needed in a wide variety of applications. A successful reuse program must therefore be preceded by an analysis phase that identifies application domains to be supported and the functionalities commonly required in these selected areas. This analysis is often referred to as *domain analysis*.

The role played by domain analysis is illustrated in Figures 1 and 2. Figure 1 shows the way applications are typically developed in the absence of domain analysis and reuse program. All applications go through their own requirement analysis, design, and coding phases independently, although they share certain common requirements. The lack of a mechanism exists to exploit such commonality will almost certainly lead to little reuse of sharable design and code components. In contrast, Figure 2 suggests how domain analysis enables a developer to identify reusable components that can be assembled as part of an applications design or code.

Research has shown that the adaptability or reuse potential of a software component depends a great deal on the degree to which analysts understand their application domain.[8] Since domain analyses are application specific, people who know best about the application domain should perform this analysis. The analysis can be performed by applying a process modeling method (e.g., IDEF) or an object-oriented

[8] Tracz, 1987

analysis method. The former helps identify the essential processes or functionalities that are commonly used in the domain's applications, while the latter attempts to identify the essential "objects" that compose the domain. An object is an encapsulation of data structure and a multitude of functions that operate on this common data structure. For example, in the application domain of inventory management, one may find such objects as parts, requisitions, customers, and warehouses. Since all applications in the inventory management domain will need these objects, the reusability effort should be concentrated on these objects.

There are a number of issues that need to be addressed:

- An assumption underlying domain analysis is that a sharable component is free from *semantic inconsistencies*. For example, in comparing the Army and Navy inventory management systems, both would contain an object called "part" that in turn would possess the attribute "part_id" at the domain model level. In the Army system, however, a 13-digit alpha numeric field is used to define inventory, while Navy conventions dictate a 9-digit number. Therefore, domain analysis must be supplemented with mechanisms that can resolve such semantic inconsistencies.[9]

[9] Identifying such semantic differences can sometimes be helpful in moving to greater standardization across the services, but software reuse does not force such standardization.

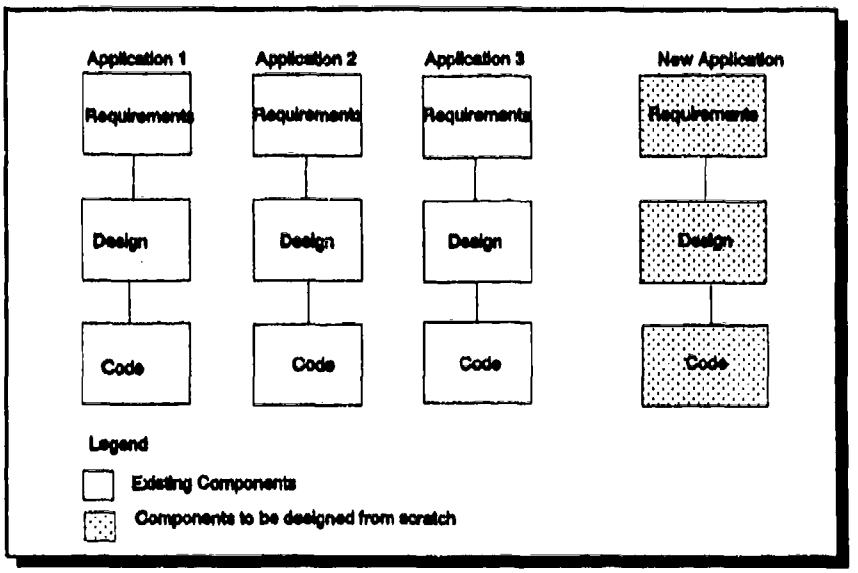


Figure 1. "Stovepipe" Systems Development of similar applications within a domain

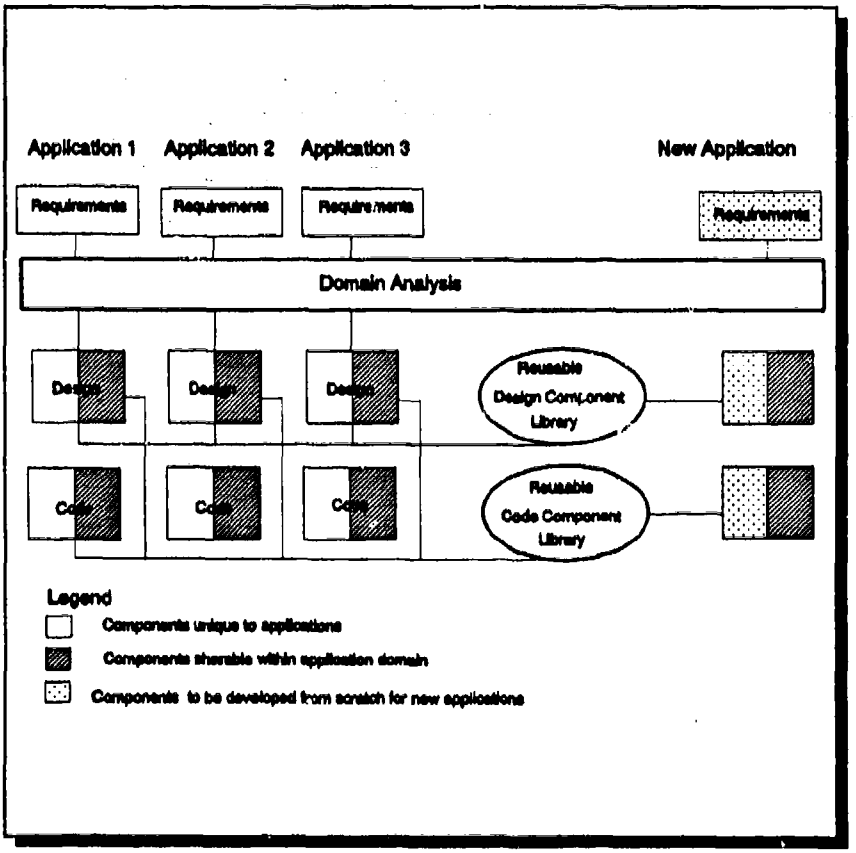


Figure 2. Systems Development using Domain Analysis to identify RSCs from existing applications

- Another issue is how to define the *scope* of a domain. Differences in the way organizations conduct business point to the need for domains that are more narrowly defined. For example, both the Air Force and Navy have procedures in place to ensure that repair part inventories are maintained at certain minimum levels. However, the Navy model defining reorder levels for parts supporting submarine launched ballistic missiles is driven by functionally different business practices and standards than the Air Force model addressing ICBM support. Thus, in this instance, the "DOD Logistics" domain should be broken into more narrowly defined domains such as "Navy Logistics" and "Air Force Logistics".

- It is important to note that, as a requisite for reuse, a domain candidate must be well understood, fully operational, and stable in nature. Once established, a domain model should *not* be subject to change. A modification to the domain implies a fundamental shift of the way business is conducted, making it virtually impossible to identify sharable components. Thus, performing analysis on domains that are unstable is not productive.

Domain analysis, by assuring that essential building blocks are identified and developed into RSCs, promotes an environment in which the major portion of a new application can be completed by simply selecting and putting together existing building blocks.

III. A Clearinghouse for Software Reuse

A. Toward the Concept of a Clearinghouse for Reuse

To effectively reuse software, RSC donors and recipients must incur non-trivial technical and administrative overheads. They need a central organization or clearinghouse to relieve them of these overhead costs. It is only through the clearinghouse that interactions between the users and donors are likely to be sustained.

1. The RSC Donor-Recipient Cycle

In many instances, an RSC will come from existing systems or systems currently under development. For an RSC to be reusable beyond its development site, it is important to distinguish the point of view of the donor from that of the recipient. A *donor* is one who identifies reusability of an RSC, develops the RSC that satisfies reusability specifications, and makes the RSC available in a library or repository. A *recipient* is a developer who reviews RSC available in the library and selects those that most closely support his requirements. Mechanisms for implementation as well as issues involving incentives or technical problems will vary depending on the orientation as donor or recipient.

a. Mechanism for Creating RSCs — the Donor's Perspective

When a program manager participates in the process of contributing software, he assumes a donor's perspective to reuse. He needs to identify and describe the

functionalities embedded in candidate RSCs. He also needs to carry out testing and documentation for those RSCs.

A willing donor is one who is able to look beyond the immediate costs both in time and resources and commit to the long-range reuse strategy. To alleviate this effort, help from external source(s) is usually required unless the donor himself is in the business of software production and can reap the benefits of donated RSCs.

b. Mechanism for Retrieving and Using an RSC — the User's Perspective

In contrast to the donor's orientation where benefits from reuse can only be anticipated at some point in the future, RSC users realize an immediate advantage. To maximize the benefits of reuse, the user must possess skills and experience to:

- Find RSCs (cataloging, search, and retrieval mechanisms)
- Understand RSCs characteristics
- Adapt RSCs to his functional requirements

From the user's perspective, reuse will be attractive only if the overall effort to reuse a piece of software is less than the effort required to create it from scratch. If a user has to invest an unacceptable amount of time in searching and evaluating repository components, he will likely opt to develop the component himself instead. Thus, it is critical that an RSC library be made readily assessable to the users. This repository has to contain RSCs that correspond to the user's requirements and needs.

2. Motivations for a Clearinghouse

Although reuse is an appealing concept, its implementation requires a sustained economic and managerial effort that goes beyond that of an individual participating software developer. The creation of a clearinghouse for reuse is required to direct much of this effort, perform centralized services, provide broader visibility of useful software,

and promote standardization. It is expected that RSC users will yield greater savings of maintenance dollars.

a. Economic Incentive

Before software reuse can begin to pay off, an up-front investment is required. There will generally be costs involved in the preparation of the software prior to its induction into the library. Whether software is written for future reuse or taken from somewhere else, it requires formatting, testing, and documentation. Providing untested and undocumented RSCs is counterproductive. If the user needs to perform significant modifications to adapt the component, the benefit of reuse may be lost.

In the short run, management of organizations that desire to donate software cannot be expected to support up-front costs for making RSCs. The fear that reusability will lead to reduction in their budget and staff is also a source of resistance among managers and programmers.[10] Programmers are often penalized for taking the extra time to make software reusable.

It is thus important that a clearinghouse for reuse be built to absorb the cost of establishing RSCs and — more importantly — maintaining a library of reusable software components. The cost of a library depends on its size and the tools used to populate, organize, access, and maintain RSCs.

A clearinghouse for reusable software has significant advantages. With its resources dedicated to reuse, the clearinghouse can assume the essential and unique role of networking multiple libraries owned by various participating organizations developing similar software, thus yielding economies of scale. It can also develop its own RSCs if they cannot be found anywhere else.

It is expected that the cost savings of reusing thoroughly tested and documented software will be even more significant in the long-run. However, the long-term benefits

---

[10] Wong, 1987

of reuse are often hampered by short-sightedness of many RSC donors and users. It is the mission of the clearinghouse to correct this short-term perspective, and play an active role in reconciling conflicting interests between individual RSC donors and recipients.[11]

b. Managerial Incentive

Researchers tend to agree that the lack of a clear reuse strategy has been one of the major factors inhibiting widespread software reusability.[12] The absence of an appropriate high-level reuse strategy reduces the motivations of project managers and programmers to reuse software. Reuse will not happen by itself; it needs to be promoted with incentives.

Top management of participating organizations must understand management's critical role in addressing non-technical issues such as legal and proprietary rights, compensation for RSC developers, and internal cost apportionment methods for purchasing reusable components associated with software reuse.[13] Contract issues concerning ownership and rights to the developed software arise. Contracts must be tailored to meet the needs of both donors and users in order to provide an incentive for reuse.

Management that invests in the development of a meaningful incentives program will enhance their organization's chances of implementing a successful reuse environment. For example, the National Aeronautics and Space Administration (NASA),

---

[11] Who pays the added development cost involved in designing the software for reuse? Ideally, the organization that profits from the reuse should pay for it, but it does not always work this way. For example, a contractor might be asked to make his software reusable with little recognition of the added cost required. As a consequence, he earns a lower profit. Furthermore, the software component could then be given to a competitor to be reused, thus allowing the competitor to capitalize on the initial contractor's efforts. It can work in the contractor's favor as well: the customer might pay an added cost for highly reusable software without realizing it, so that the contractor can reuse the software in his own future programs.

[12] Biggerstaff and Richer, 1987

[13] Banker and Kauffmann, 1990

in a move to encourage development of higher quality software by contractors, has instituted financial rewards for certain types of library resident routines. Developers are compensated for extracting software from the library instead of being paid solely by the quantity of new code they create. This provides the contractor with incentive to utilize the methodologies of reusable code.[14]

GTE provides another example. GTE Data Services places major emphasis on incentives and has introduced a program that rewards authors, project managers, and reusers. Programmers receive both cash bonuses (when an asset was accepted into the repository) and royalties each time an asset is reused in a new application. Budget increases and promotions for project managers are directly linked to high percentage reuse in deliverables under their cognizance. GTE considers the incentive program a key factor in their reuse success, which translates into an estimated savings of $1.5 million during the program's first year of operation and a projection of $10 million by the end of the fifth year.[15]

B. Activities of the Clearinghouse

The following are the main activities that the clearinghouse should perform on behalf on the reuse community:

1. Defining Reusable Domains

Since software comes from a variety of domains, there is a need to identify software components that share basic functionalities. This can be achieved by a process known as *domain analysis*. The purpose of domain analysis is to determine commonalities within the application domain, focusing on areas with the greatest potential for reuse and in greatest demand by future software developers. It is an iterative process

---

[14] Cashin, 1991

[15] Prieto-Diaz, 1987

involving an intense examination of the domain of interest.[16] Without a well-performed domain analysis, RSCs cannot properly be identified.

2. Searching for RSCs

The search for RSCs is a non-trivial effort. Potential sources (e.g., existing governmental systems, public domain software, or commercial off-the-shelf software) have to be identified. Of particular concern during this component search is a donor's reputation or track record for producing quality software.

3. Certifying RSCs

Certification refers to the process designed to solve and eliminate concerns programmers and managers have about using RSCs that originate from outside their own work. Such concerns include quality, maintainability, liability for defects, and testability. A certified RSC must function as it is intended to function. Evaluation is a very important part of the certification process. It begins as candidate RSCs are identified and continues through the remainder of the life cycle. Evaluation can eliminate unsatisfactory components, identify re-engineering needs, produce documentation, and initiate essential metrics.[17]

4. Creating a User-Friendly Library

A library system is needed for users to identify, retrieve, and use RSCs. Software selected for incorporation in the library must be integrated with other repository

---

[16] Softech, RAPID Center Reusable Software Component Procedures, June 1990

[17] Metrics assist managers in measuring various things and allow engineers to apply predictive algorithms. Metrics include size, productivity, efficiency, and quality characteristics such as portability, maintainability, and reusability. Metrics also provide a prediction of problem areas and alternative solutions.

components via an identification scheme. The cataloging system implemented should be easy for the user to understand, and should provide alternative search patterns for the user to search for the perfect component. Lastly, the indexing system should be adaptable in the event that future components are not easily tailored to existing categories.

5. Supporting RSC Users

A productive reuse environment cannot be maintained without the confidence of the user. This confidence is achieved in several ways. Most notably, the RSC certification process contributes to this end by ensuring both the quality and standardization of each component in the repository. Additionally, RSC users' concerns and needs should be periodically surveyed to ensure efforts are continually focused on the needs of the customer.

In summary, the implementation of a clearinghouse offers several advantages:

- The clearinghouse can perform domain analysis over a broad range of applications, which will likely increase the accuracy of the domain model and its relevance for future application development.
- With its central position, the clearinghouse has the authority and expertise to enforce a strict reuse discipline. Certification procedures can be put in place to ensure uniform quality of RSCs. This directly influences user confidence in the clearinghouse.
- As the clearinghouse imposes a standardized reuse procedure, it offers the opportunity to serve a larger community.

IV. DSRS – A Clearinghouse for Software Reuse

A. CIM and Reuse

Launched in October 1989, the DoD Corporate Information Management (CIM) initiative seeks to improve DoD business processes and the management of information resources. To achieve this goal, CIM is calling for functional interoperability between systems, standards compliance, and efficiency in software development through reliance on reusable software components, commercial off-the-shelf products, and computerized application development aids.

CIM promotes two types of repositories: software reuse repository and hardware reuse repository.[18] The objectives of software reuse repository are to:

- Develop a central DoD-wide RSC clearinghouse
- Establish a data dictionary for DoD
- Build an integrated repository for C3I software

The hardware reuse repository seeks to:

- Shorten acquisition cycle by leasing
- Introduce a standard bus architecture for scalable processors
- Provide for central technology renovation
- Rationalize capacity and security management practices
- Distribute capacity by means of survivable networks

[18] Strassmann, 1991

Software reuse is considered to be one of the key strategies intended to help DoD in responding to variable threats in a rapidly changing environment with less resources. The Office of the Director of Defense Information sets the following goals for software reuse:[19]

- 100% reusable data with an infinite life for data definitions
- More than 80% reusable code with more than a 20 year life on software elements
- 80/20 development/maintenance ratio
- Technology asset life two to three times larger than the technology innovation cycle.

The goal of implementing a DoD-wide repository of reusable components has culminated in the recent establishment of the Defense Software Repository Service (DSRS). Under the administration of the Center for Software Reuse Operations (CSRO),[20] the DSRS provides automated access to more than 1550 government or commercially owned/developed RSCs. This repository is available to all DoD, other government agencies, and authorized contractors.

The design and implementation of DSRS is based on a pilot operation initiated by the Army's Reusable Ada Product for Information System Development (RAPID). This chapter reports some of the experiences gained by RAPID, and subsequently DSRS.

---

[19] Strassmann, 1991

[20] CSRO has been set up within the Defense Information System Agency's Center for Information Management.

B. RAPID - The Army's Software Broker

1. History and Mission

The Army initiated the RAPID project in 1987 in recognition of the tremendous potential of software reuse. The army established a software reuse clearinghouse offering Army/DoD users a centralized repository of reuse components.[21]

With Ada as the programming language mandated by Congress, RAPID sought to promote the reuse of Ada software to reduce the cost of system development and maintenance through the use of previously developed, tested, and implemented components. Ada is a programming language designed to facilitate reusability because its reusability guidelines are structured to include design for reuse, parameterization, and domain analysis.[22] Ada code is portable in that code written anywhere is potentially reusable for another system. It is well suited to the integration of system components from multiple sources.

RAPID defines its missions as follows:

- Achieve the Department of Defense (DoD) initiative of reusable, maintainable, and reliable software
- Develop, maintain, and administer a comprehensive reuse program
- Lower software lifecycle costs by increasing productivity and quality.

Initially intended for management information systems (e.g., financial, logistics, and personnel applications), RAPID expanded its domain to additional application areas (e.g., telecommunications). In 1991, it had more than 960 reusable components stored in a central repository available to all Army/DoD units. RAPID issues software to both DoD users and contractors working on DoD projects as government-furnished equipment (GFE).

---

[21] RAPID was located at the U.S. Army Information Systems Software Development Center, Washington (SDC-W).

[22] Banker and Kauffmann, 1990

2. RAPID Implementation Plan

RAPID was initiated at SDC-W as a pilot prototype reuse program in July 1987 when the Phase I contract was awarded. This initial phase produced the foundation

RAPID PROJECT PHASES

PHASE I: Design and Development (July 1987 - April 1989)

- RAPID Center Concepts and Organization
- Reuse Policies and Procedures
- RAPID Center Library System

PHASE II: Pilot Operation (May 1989 - December 1990)

- Operate Active RAPID Center
- Support SAC-W Customers
- Policy and Procedure Refinement
- Library Population
- Training Program
- Domain Analysis

PHASE III: Implementation (January 1991 - September 1991)

- Expand to all ISEC Development Centers
- Expand to Other Organizations
- Continue Library Population and Enhancements

PHASE IV: CIM Operation (October 1991 - October 1996)

- Individual Service focused Support
- Expand Domain Analysis Customers
- Continue Library Population

Table 1. RAPID Project Phases

needed to provide a reusability program within SDC-W. Table 1 depicts the chronology of the phases of the RAPID project.

3. RAPID Staff Organization

Under the supervision of a center manager, the RAPID staff is composed of technical consultants, systems analysts, software engineers, configuration management specialists, quality assurance personnel, administrative assistants, and librarians. The staff's mission is to encourage design methods and architectures that build from reusable components. Systems analysts and software engineers provide vital support for RAPID's role as a software clearinghouse while other positions, such as RAPID's administrative assistants, are more heavily weighted towards user assistance or RSC cultivation.

Support personnel for the program consisted of 16 government employees and eight contractor personnel. The following describes some specific positions of these staff members:

a. *The RAPID Manager*: Continually monitors the success of the RAPID program and the results of specific operations. He keeps extensive reports that help evaluate the costs of the program as well as the savings to developers.

b. *Technical Consultants*: Perform the domain analysis, attend design reviews, stay abreast of projects, advise project staffs, and assist the developer in identifying potential areas of reuse. They help search for RSCs, and provide guidance, support, and documentation to programmers.

c. *System Analysts and Software Engineers*: Identify high-value RSCs that are to be added to the library. They evaluate, test, and document all RSCs before being added to the library. They also provide maintenance and enhancements to the RCL (Rapid Center Library) software system.

d. *Configuration Management Specialists*: Ensure that all configuration activities are performed for each library component, including problem report tracking, controlling changes, and releasing new versions or enhancements.

e. *Quality Assurance Specialists*: Ensure that all RSCs are of high quality through frequent reviews. They develop and administer testing as well as establish and enforce metrics.

f. *Administrative Assistants*: Prepare all RAPID Center Library System reports and perform follow-up interviews with the users.

g. *The Librarian*: Maintains the RSC data base and performs normal operator functions.

4. RAPID Activities as a Clearinghouse

a. Defining Reusable Domains

For a clearinghouse to operate effectively, domain analysis must be performed. As a continuing process, it not only identifies components that may be reused, but also directs developers to areas where reuse emphasis should be placed so that new components can be found during post-deployment support.

As part of the initial steps to establish RAPID, a high-level domain analysis was done in 1987 that covered management information systems. During the pilot operation, RAPID realized the need to support multiple domains. Therefore, policies, procedures, and guidelines were revised to extend their potential applicability beyond MIS.

Standardized object-oriented methods were adopted for domain analysis. As a standardized method, they proved to be effective in identifying reuse opportunities, and for grouping RSCs according to the level of abstraction or functional category.

b. Searching for RSC Donors

Once the reusable types of software components are identified through domain analysis, RAPID engineers must determine where those components will come from. These personnel review a variety of sources to identify potential candidates, including:

- COTS Software
- Government-owned Software
- Public Domain Software

Of the 960 RSCs currently in the repository, more than 780 have been developed commercially. Of the 170 government-owned components, less than 30 were developed by the RAPID in-house engineers.

COTS software tends to be a more fruitful source for several reasons. It is well-tested before release, and is often accompanied by substantial documentation. Typically, the software has been in use for a substantial period of time before it is identified as a candidate RSC. It is thus likely to be highly reliable.

Roughly 85% of the RSCs in the Army repository are code. The remaining component types include such things as design components, documentation components, and functional specifications. Although originally conceived to house Ada code, the repository does contain RSCs written in other languages, such as COBOL, C, and FORTRAN.

c. *Certifying RSCs*

The certification process begins with a quality evaluation of the candidate RSC once it is identified. The evaluation process not only determines basic attributes such as the lines of code or number of packages but also estimates the reuse potential for the component and the level of re-engineering needed to ensure a quality standard.

The certification includes testing on a variety of platforms. Re-engineering is not done for commercial off-the-shelf components whenever code changes may invalidate the license. COTS RSCs are entered into the RAPID Center library after the applicable RAPID Center documentation standards are met.

The RAPID Center assigns a certification level as "the level of confidence" in the quality of the RSC; it ranges from level I to level V, as follows:

Level I: Depository - No formal testing and documentation

Level II: Reviewed - Some testing and documentation

Level III: Tested - Test Suites Validated and some documentation

Level IV: Documented - Fully tested and documented; meets all standards and guidelines

Level V: Secure - Currently not used

Level V certification is reserved for future use to cover secure components in accordance with DoD CSC-STD-001-83, *Trusted Computer System Evaluation Criteria.* Policies and procedures for secure components will be developed whenever such RSC become available.

Ideally, every component in the library should be brought to a Level IV certification. This has not been the case, however, and a significant number of RSCs have been accepted into the library at the lower certification levels. Of the more than 780 COTS components in the repository, only 285 were certified at Level II, while the remaining ones are Level IV. A similar breakout of government-owned components could not be obtained, but it was confirmed that a number of these components are certified at Levels I and II.

d. Creating a User-Friendly Library

RAPID uses a flexible *faceted* classification scheme to store and retrieve RSCs. Using the PC-based, menu-driven system, the user can initiate a search for an RSC by entering parameters or "facets" that describe the RSC. The nine facet classification descriptors listed below allow the user substantial control over the repository search:

- Component Type
- Language
- Unit Type
- Function

- Algorithm
- Environment
- Object
- Data Representation
- Certification Level

To conduct a search, the Function, Language, and Certification Level descriptors must be provided. The use of the other facet terms is optional. To enhance the search capability, the RAPID search mechanism also provides the user a "thesaurus list" that helps identify facet terms (known as "synonym terms") that appears to be comparable to the user's description. The system also gives links (i.e., "Relationships") between RSCs so that the user can browse or extract related components.

RAPID maintains RSC metrics on reusability, maintainability, reliability, portability, actual usage, and outstanding problem reports. Based on these metrics, the users can choose to rank components and select the most appropriate ones.

e. Supporting the User

User support and feedback is an essential aspect of clearinghouse activities. In addition to the assistance from its central office, RAPID offers a remote site program where its personnel spend a period of time at the user's site to assist in establishing a local reuse repository infrastructure (e.g., reuse planning, hardware and software selection, RSC creation and population).

Formal training is also part of RAPID's support of the users. Programs have been developed at three levels: Executive, Management and Software Engineering. This training is available at both the Army Reuse Center and remote sites.

The RAPID librarian solicits the RSC recipient's feedback, approximately 90 days after the RSC extraction. The inquiry is intended to check whether or not the RSC was used or if any problem was encountered. The expectation is that such user feedback

provided on a continuous basis would ultimately result in a more responsive library. This feedback provides some clues in assessing the effectiveness and the costs of reuse.

C. DSRS — Toward a DoD-wide Reuse Program

The DoD surveyed software reuse efforts within the Department and keyed on the Army's RAPID initiative. It was clear that this program, though still in its early stages of development, was built upon a methodology that closely aligned with DoD's vision for the future regarding reuse, and the mechanisms to achieve it. Tapping on this positive learning experience, DoD established in 1991 the Defense Software Repository Service (DSRS).

1. The DoD Reuse Organizational Structure

As described in Figure 1, DSRS is under the management and supervision of the Center for Software Reuse Operations (CSRO). CSRO is a component of the Defense Information Systems Agency (DISA). DSRS is a distributed operation with four remote centers supporting DoD services and the Defense Logistics Agency.[23] DSRS supports reuse efforts at remote centers, and ensures that these efforts are complementary and not duplicative. Predominantly staffed by contractor employees, seven contract personnel fill billets in project management (1), engineering (4), configuration management (1), and librarian (1). Government personnel fill positions in customer service (1) and liaison with other government sites (1).

[23] The reuse remote centers are located at the following sites: Army Reuse Center is located in the Information Systems Command, Falls Church, VA; Navy Reuse Center at the Naval Computers and Telecommunications Station, Washington Navy Yard, D.C.; Air Force Reuse Center at the Standard Systems Center at Gunter Air Force Base, Ala.; Marine Corps Reuse Center at the Development Center, Quantico, Va.

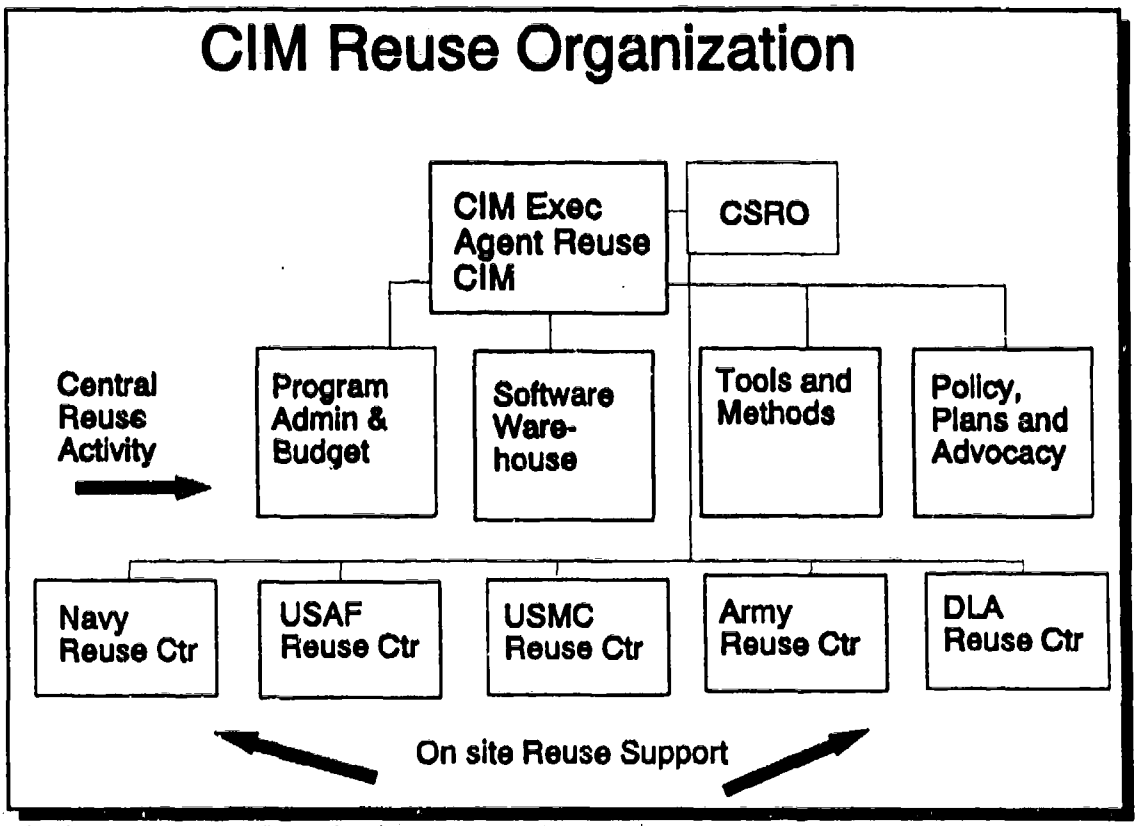


Fig 1: CIM Reuse Organization

2. Current Status of the DSRS Effort

DSRS has adopted as its core not only RAPID's library of RSCs (at the time about 840 components), but its infrastructure as well (i.e., classification/retrieval system, RSC certification methodology, etc.). At the same time, DSRS also accepted RSCs from previously established Navy and Air Force repositories. To date, DSRS more than 1550 components comprising over two million lines of code. RSCs are composed of communications packages, graphics programs, man-machine interfaces, Ada bindings, data structures, and Ada development tools.

DSRS operates on a MicroVax 4000-300 running VMS operating system.[24] Users can access DSRS from a terminal with VT100 emulation through dial-up modem or via the Defense Data Network (DDN), although full networking capability via DDN is not expected until January 1993. Currently, when a user searches for an RSC, he has to access not only the DSRS central repository but also the ones located at the services. It is planned that all repositories will be interconnected to form an integrated DSRS repository by the end of 1993.

CSRO seeks to improve the user-friendliness of the DSRS interface. Currently, on-line help is made available to familiarize the user with the system. DSRS also has a feature called "Session Maintenance" that allows the users to keep track of his/her interaction with the repository system. A graphical user interface is being studied to replace the character-based and menu-driven interface. Finally, CSRO puts in place a Customer Assistance Office (CAO) to support users.

[24] The DSRS hardware platform is an upgrade of the RAPID configuration.

V. Lessons Learned

A. A Clearinghouse is a Necessary Condition for Software Reuse in DoD

DSRS has been established to respond to a widely recognized need for a centralized repository of reusable components in DoD. Given that donors and recipients are separated by organizational and geographical boundaries, there must be a marketplace to facilitate the exchange of RSCs. A DoD clearinghouse that allows networking of all of its remote sites is expected to further enhance RSC usability. As such, a clearinghouse contributes to the proliferation of software reuse. Without a clearinghouse bridging the gap between RSC donors and recipients, the software reuse effort would fall apart.

B. A Clearinghouse is a Productivity Multiplier

Early signs of success can already be observed, as evidenced by the implementation of the Retired Army Personnel System (RAPS) and the Air Force Logistics Material Automated Retrieval System II (LOGMARS).

The Retired Army Personnel System (RAPS) is a report generator system that was a pilot study by developers and the Army Reuse Center. The goal from a reuse standpoint was to integrate reusable software components into the systems development life-cycle and create a reusable system. Although the lines of code (LOC) in RAPS are not substantial, the fact is that reusable modules accounted for a significant percentage. Also of significance is the use of design components as well as implementation RSCs:

- In the design phase four of ten system components were reused.

- Implementation activity involved the reuse of 88 of 130 subunits. This accounted for 63.9% of LOC dedicated to implementation.
- Reuse of the testing packages for reused modules was also employed.

The time savings calculated as a result of reuse in RAPS development amounted to more than 125 days.

Similar success was obtained by the Air Force with its LOGMARS-II:

- Released worldwide in February 92, LOGMARS-II is a PC-based inventory control system used in supply warehouses. The system consists of 18,600 lines of Ada code, of which 5,600 LOC were extracted from DSRS. An additional 6,300 LOC came from existing in-house modules.

- Also developed was a bar-coded label printing subsystem used for warehouse material and benchstock. Two modules consisting of over 2,500 LOC (28% of total system code) were extracted from DSRS. Five other modules, including those for screens and forms generation, were reused from LOGMARS-II. Overall, 73% of the subsystem code consisted of reused code.

As the utilization of the clearinghouse's RSCs has started to show signs of success, it is expected that more RSCs will be demanded in the future. With a well-populated repository, the clearinghouse would serve as a productivity multiplier.

C. The Clearinghouse Must Populate its Repository with Quality RSCs

In response to the growing appreciation of reuse, RSCs are being inducted into the repository at a rapid pace. The clearinghouse must ensure that RSC quality is not compromised in the process. RSC users are reluctant to utilize components of uncertain quality developed by unknown sources; they want nothing but "error-free" RSCs. The clearinghouse must carefully weigh the benefits of accelerating the population growth of RSCs against the potential damage that might be caused if users experience problems with RSCs.

D. Domain Analysis is the Foundation for a Successful Software Reuse Program

The software reuse program must be preceded by an analysis phase in which (a) the application domain to be supported by the reuse program is determined, and (b) the functionalities commonly required in the applications belonging to that domain are identified. Domain analysis, by assuring that every essential building block is identified and developed into an RSC, promotes an environment in which the major portion of a new application can be completed by simply selecting and putting together existing building blocks.

E. High-Level DoD Management Must Provide Incentives for Reuse

A fully functioning repository populated with pertinent components is a necessary but not sufficient condition to promote a successful reuse program. Some inducement to reuse software is required as well. To carry the necessary weight and visibility, these incentives must be promulgated from the highest authority within DoD.

While many DoD organizations[25] involved in software reuse have recognized this issue, there is no policy regarding financial incentives. It is imperative that incentives research continue and these critical issues be resolved in order to implement a competent reuse strategy.

---

[25] CSRO recognizes the critical issue of reward, and is working closely with organizations such as the Joint Avionics Working Group (JIAWG), and the Ada Joint User's Group (AdaJUG) both of which are conducting extensive research on incentive issues.

Glossary of Terms

ADA - Programming language that facilitates reuse. Ada is the primary programming language in the Department of Defense.

ADAMAT - A static source code analyzer that produces 150+ metrics on the quality of Ada code as it pertains to reliability, maintainability, and portability.

CASE (Computer Aided Software Engineering) - Collective resource to a family of software productivity tools.

CODE REUSE - The most common form of software reuse in which source code is reused.

DESIGN REUSE - Reuse of a type of reusable software component such as logical data models, functional descriptions, and diagrams.

DOMAIN - A group or family of related systems that share a set of common capabilities and/or data.

DOMAIN ANALYSIS - The thorough examination of a domain that produces a representation of the domain and identifies common domain characteristics, primary functions, and objects.

FACETED CLASSIFICATION SCHEME - Components are classified by selecting the most appropriate terms from each facet to best describe the component.

FOURTH GENERATION LANGUAGE - Programming language that uses high-level human-like instructions to retrieve and format data for inquiries and reports

GRANULARITY - The size of the chunk of code.

IMPLEMENTATION REUSE - Reuse of a type of reusable software component such as package specifications, package bodies, subsystems, and test suites.

METRICS - Numeric measures that characterize RSCs.

OBJECT-ORIENTED DESIGN - Method for generating reusable software components. It uses data types as the base for modularization and defining objects.

REPOSITORY - Storage area for reusable software component.

REUSABLE SOFTWARE COMPONENT - Originally a source code component consisting of functions, procedures, or packages. Now it includes requirements, design, implementation, templates, and generic architectures.

REUSABILITY - Ability for a software component to be reused.

References

Apte, U; Sanker, C.S.; Thakur, M; and Turner, J "Reusability Based Strategy for Development of Information Systems:Implementation Experience of a Bank" *MIS Quarterly*, December, 1990, pp 421-433.

Banker, Rajiv and Kauffmann, Robert "Reuse and Productivity in an Integrated Computer Aided Software Engineering (CASE) Environment: An Empirical Study" *MIS Quarterly*, September 1991 pp 375-395.

Banker, Rajiv and Kauffmann, Robert "Factors Affecting Code Reuse: Implications for a Model of Computer Aided Software Engineering Development Performance", December 1990.

Basili, Victor and Musa, John "The Future Engineering of Software: A Management Perspective", *IEEE Computer*, September 1991, pp 90-96.

Biggerstaff, Ted and Lubars, Mitchell "Recovering and Reusing Software Designs", *American Programmer*, March, 1991, pp 3-11.

Biggerstaff, Ted "Reusability Framework, Assessment, and Directions", *IEEE Software*, March, 1987, pp 44-49.

Biggerstaff, Ted "An Assessment and Analysis of Software Reuse", July, 1991.

Burton, Bruce "The Reusable Software Library", *IEEE Software*, July, 1987, pp 25-32.

Caldiera, Gianluigi and Basili, Victor "Identifying and Qualifying Reusable Software Components", *IEEE Computer*, February, 1991, pp 61-69.

Cashin, Jerry, "To Move Beyond a Metaphor, Reusability Needs to Get Real", *Software Magazine*, October, 1991, pp 100-106.

Ferguson, Glover "Reuse and Reengineering", *American Programmer,* March, 1991, pp 39-43.

Fischer, Gerard "A Cognitive View of Reuse and Redesign", *IEEE Software*, July, 1987, pp 60-72.

Gargaro, Anthony "Reusability Issues in ADA", *IEEE Software*, July, 1987, pp 43-51.

Horowitz, Ellis "An Expansive View of Reusable Software" *IEEE Transactions in Software Engineering*, September 1984, pp 477- 487.

Jones, T. Capers "Reusability in Programming: A Survey of the State of the Art", *IEEE Transactions in Software Engineering*, September 1984, pp 488-493.

Jones, Gerald and Prieto-Diaz, Ruben "Building and Managing Software Libraries", *IEEE Software,* February 1988, pp 228-236.

Kaiser, Gail "Melding Software Systems from Reusable Building Blocks", *IEEE Software*, July, 1987, pp 17- 24.

Lanergan, Robert and Grasso, Charles "Software Engineering with Reusable Designs and Code", *IEEE Transactions in Software Engineering*, September 1984, pp 498-501.

Meyer, Berstrand "Reusability: The Case for Object Oriented Design", *IEEE Software*, March 1987, pp 50-63.

Nieder, "RAPID: Implementing a Comprehensive Reuse Program", Dec 1987

Prieto-Diaz, Ruben "Classifying Software for Reusability" *IEEE Software*, January 1987, pp 6-16

Prieto-Diaz, Ruben, "Implementing Faceted Classification for Software Reuse", Communications of the ACM May 1991 International Conference on Software Engineering Special Report (ICSE-12),

SOFTECH, "RAPID Center Standards for Reusable Software", Jan 1989

SOFTECH, "ISEC Reusability Guidelines", December 1985

Standish, Thomas "An Essay on Software Reuse" *IEEE Software Engineering*, September 1984, pp 494-497.

Syms, Trevor and Braun, Christine "Software Reuse: Customer vs. Contractor Point-Counterpoint", March 1991, pp 326-337.

Tracz, William "Reusability Comes Of Age", *IEEE Software*, July 1987, pp 6-8.

Tracz, William "Legal Obligations for Software Reuse", *American Programmer*, March 1987, pp 12-17.

Vogelsong, and Rothrock, "RAPID, Lessons Learned during Pilot Operations", December 1988.

Vogelsong, "RAPID, An Operational Center of Excellence for Software Reuse", December 1988.

Wartik, Steven "Filling: A Reusable Tool for Object Oriented Software", *IEEE Software*, March 1986, pp 61-69.

Distribution List

Agency	No. of copies
Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library, Code 0142 Naval Postgraduate School Monterey, CA 93943	2
Office of Research Administration Code 08 Naval Postgraduate School Monterey, CA 93943	1
Library, Center for Naval Analyses 4401 Ford Avenue Alexandria, VA 22302-0268	1
Department of Administrative Sciences Library Code AS Naval Postgraduate School Monterey, CA 93943	1
Director of Information OASI (C3I) Washington, D.C. 20301-3040	10
Tung X. Bui Code AS/Bd Naval Postgraduate School Monterey, CA 93943	10