Session 3: Reuse

Jon Valett, NASA/Goddard, Discussant

Sharon Waligora, Computer Sciences Corporation

Neil Iscoe, EDS Research

William Wessale, CAE-Link Corporation

PRECEDING PAGE BLANK NOT FILMED

N94-11429

MAXIMIZING REUSE: APPLYING COMMON SENSE AND DISCIPLINE

Sharon Waligora
James Langston

COMPUTER SCIENCES CORPORATION
GreenTec II
10110 Aerospace Road
Lanham-Seabrook, MD 20706
(301) 794-4460

ABSTRACT

Computer Sciences Corporation (CSC)/System Sciences Division (SSD) has maintained a long-term relationship with NASA/Goddard, providing satellite mission ground-support software and services for 23 years. As a partner in the Software Engineering Laboratory (SEL) since 1976, CSC has worked closely with NASA/Goddard to improve the software engineering process. This paper examines the evolution of reuse programs in this uniquely stable environment and formulates certain recommendations for developing reuse programs as a business strategy and as an integral part of production. It focuses on the management strategy and philosophy that have helped make reuse successful in this environment.

INTRODUCTION

For 23 years, Computer Sciences Corporation (CSC)/System Sciences Division (SSD) has built satellite ground-support software systems for the Goddard Space Flight Center (GSFC) of the National Aeronautics and Space Administration (NASA). In this uniquely stable environment, CSC has gradually developed a reuse program that has allowed the company to meet increasingly challenging business needs and has now become a major element of SSD's business strategy. Reuse at CSC is seen not only as a technical issue but also as a matter of management; meaningful reuse strategies are characterized by the consistent, coordinated application of common sense and discipline. This paper examines how this successful reuse program evolved over the past 23 years and presents five key factors that contributed to its success.

Scope of the Study

The work described here has been performed at CSC under its Systems, Engineering, and Analysis Support (SEAS) contract and predecessor contracts. The primary objective of these contracts is the development of scientific satellite ground-support systems for NASA/Goddard's Mission Operations and Data Systems Directorate (Code 500). In particular, the development of attitude ground-support systems (AGSSs) and simulators built for Code 550 (the Flight Dynamics Division) and payload operations control center systems (POCCs) built for Code 510 (the Mission Operations Division) serve as good examples of the

PRECEDING PAGE BLANK NOT FILMED

benefits derived from reuse practices developed in this environment.

During the early 1970s, documentation procedures were at worst nonexistent and at best intermittent; CSC, like most of the industry, had not yet developed the practice of recording project histories and statistics. Thus, much of the evidence for the early years of this period – the "prehistory" of reuse in our organization – is anecdotal; still, the views presented by long-term employees are consistent with the overall picture and existing technical documents of the time. After 1976, with the founding of the Software Engineering Laboratory (SEL), documentation of our project experience became both more extensive and more analytical (References 1, 2).

The SEL was founded by the Flight Dynamics Software Engineering Branch of NASA/Goddard (Code 552), the Department of Computer Science of the University of Maryland, and the Flight Dynamics Software Engineering Operation of CSC/SSD specifically to investigate the effectiveness of software engineering technologies applied to the development of applications software. During the past 17 years, the SEL has measured and evaluated the development process, conducted numerous experiments, and documented its findings in many professional papers and reports. In particular, the Software Engineering Laboratory Series offers ample documentation of the evolution of software engineering in this environment. As a whole, this collection of documentation forms an unusually rich body of historical information for examining the evolution of software reuse (Reference 3).

The 32 attitude systems charted here in detail were all built between 1977, when the first projects measured by the SEL were completed, and 1992. They are sequentially ordered by project start date. Figure 1 shows the percentage of reuse achieved by each project, based on the amount of code reused (reused code is defined here as existing code used verbatim or with less than 25-percent modification). A linear fit of these data shows a steady

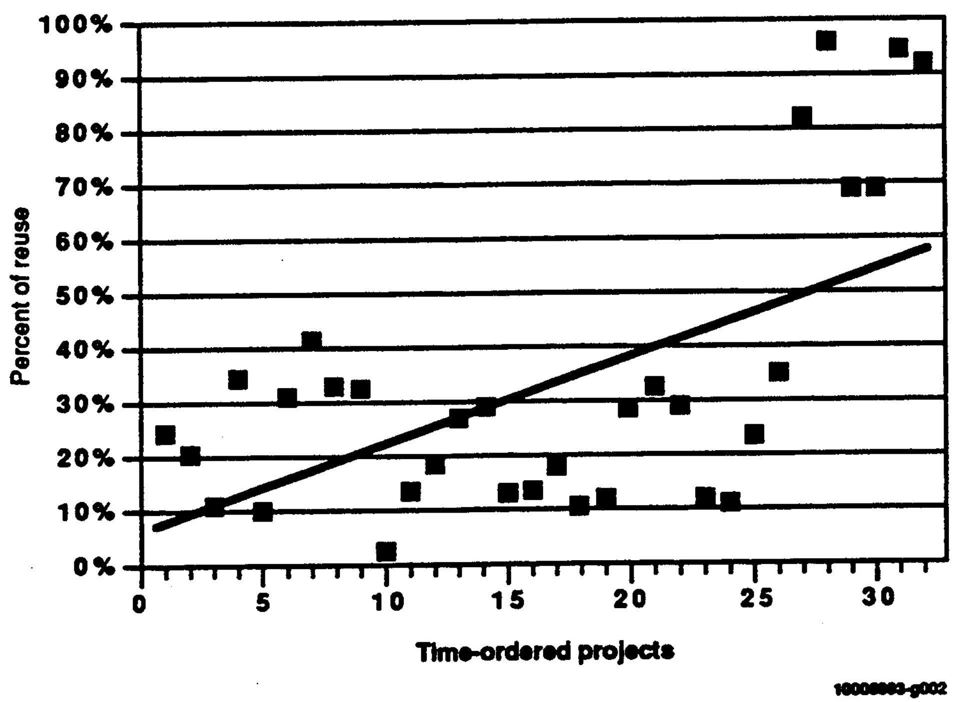


Figure 1. Percentage of Reuse Achieved by the Projects Studied

increase in reuse, rising from an initial 10 percent in 1977 to a current 60 percent.

Examining the cost (measured as effort) to develop these same systems (Figure 2), we see that the cost to create 1000 source lines of code (KSLOC) from scratch did not change significantly during this period, but that the effort to develop systems with reuse dropped significantly, from 325 hours per KSLOC to slightly over 100. This confirms SEL studies that reused portions of software can be delivered with savings of 70 percent to 80 percent over entirely new code (Reference 4).

EVOLUTION OF REUSE AT CSC

Historically, the steady growth of reuse at CSC evolved through four very distinct levels, as shown in Figure 3. At first, reuse was completely dependent upon people, because people are the essential element of the software business. We then began to focus on code reuse, building reusable products. Next, we developed a reusable process. And, most recently, we have expanded our approach to address reuse in the full life cycle, beginning with requirements and design. Experience gained throughout these four levels has given us increasing control over the future development of our reuse programs, as well as our development process and business policies in general. We have already begun moving to what we believe will be the next level, the reuse of architectures.

The levels were built cumulatively; as the organization matured from one level to the next, previous reuse levels were adjusted and further developed to support the new level. At each level, developers were encouraged to make a conscious effort to build something reusable, and managers actively promoted the reuse of these items.

Reuse of Personnel

In the early 1970s, software reuse was almost unheard of; programming was just beginning to gain recognition as a profession. Most programs were small tools that were developed by scientists

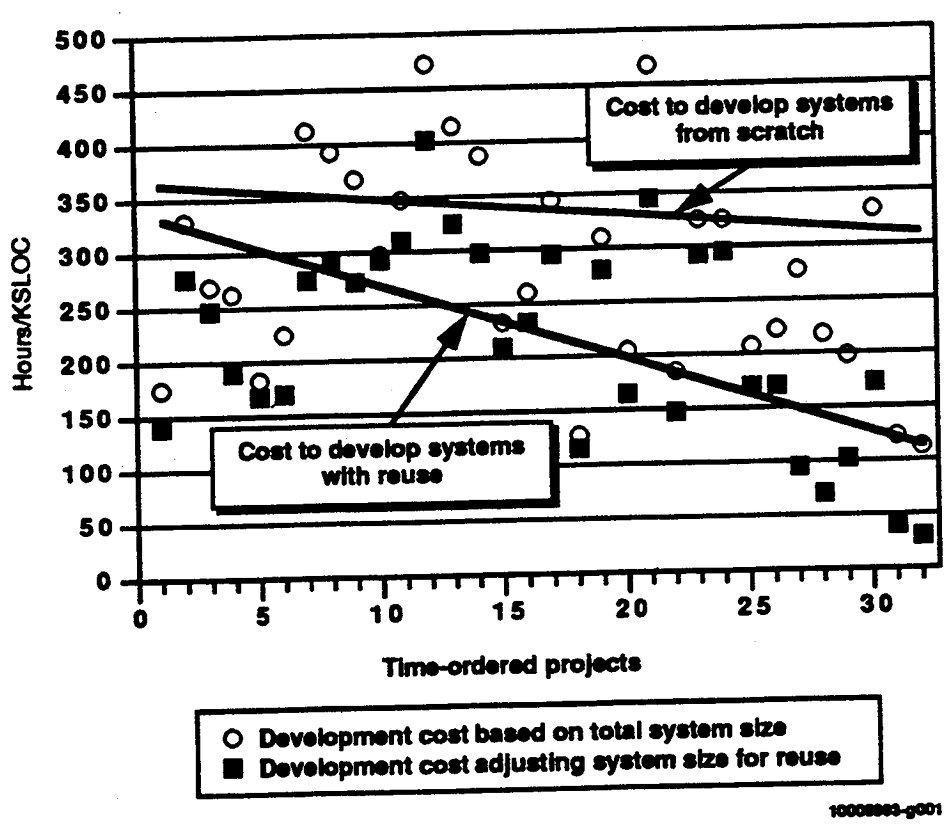


Figure 2. Cost To Deliver the Projects Studied

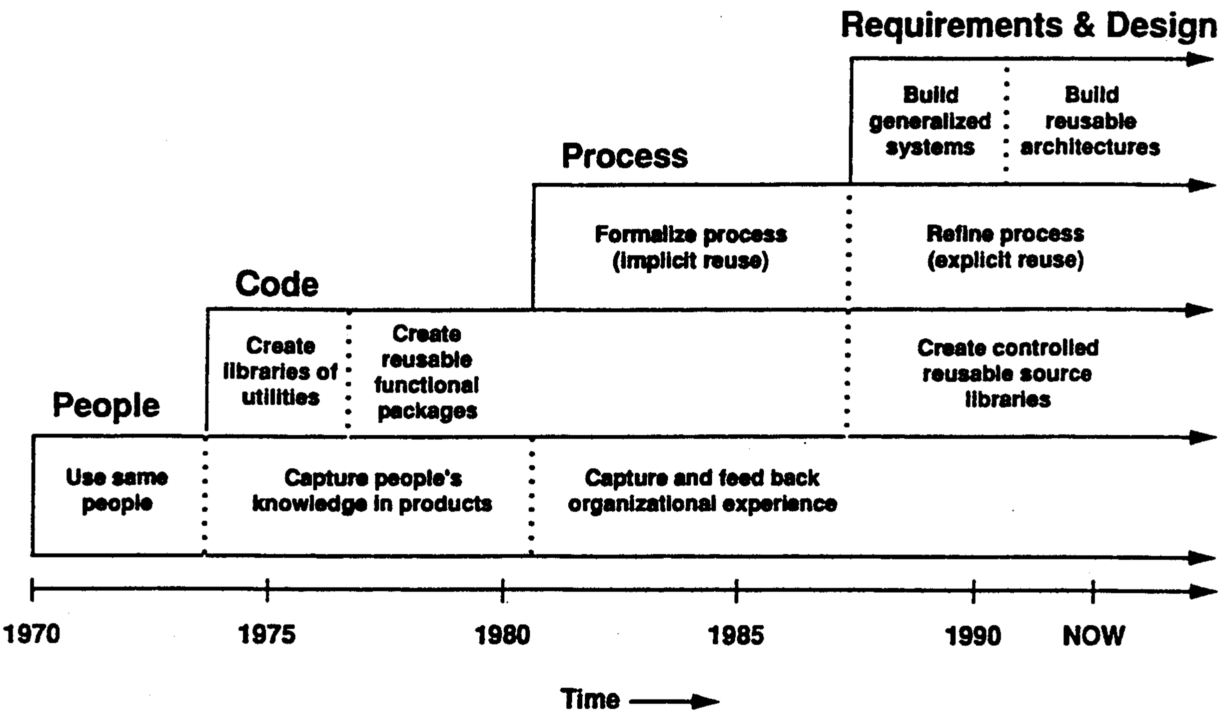


Figure 3. Reuse Levels

as a means to solve a problem; little attention was paid to the structure, maintainability, or reusability of software products. Such was the case here. We were just beginning our relationship with GSFC, providing flight dynamics ground support for scientific satellites. Physicists, astronomers, and aeronautical engineers wrote their own programs and tools to help them perform their mission support duties.

However, our software reuse program had its beginnings in this unstructured period of software development. Because software was merely a means to an end, the scientists would reuse any pieces of code that they had created for a previous program, as a common-sense way of saving some time and effort. Usually, they reused only what they had written themselves, because they knew it existed, they understood what it did, they trusted it, and they knew how and where to get it. Only rarely would they borrow something from a colleague.

Software reuse was thus confined to the scope of the individual developer or to a small group of people, who viewed it as an informal, ad-hoc way to get the problem solved as quickly as possible. That attitude also prevailed at an organizational level. Our management philosophy at this time was simply to "reuse" people from one project to another, assigning personnel on the basis of past experience and specific expertise. Using this approach, we were building a small staff of application experts who could provide satellite mission support and develop software tools to solve flight dynamics problems.

The weakness in this management approach was that the knowledge and experience were embodied in individual employees; when people were reassigned to other projects, or when they left the organization, their knowledge and experience went with them. It is significant, however, that the idea of reuse, even in these highly personalized forms, was already inherent in our way of doing business. This set the stage for the next level of development. The nature of the product itself served as the catalyst that started crystallizing these impromptu forms of reuse into a formalized process.

As we approached the mid-1970s, attitude ground support problems were getting more complex; it was clear that we needed a more structured approach to provide an integrated set of tools to support each spacecraft mission. Logic told us that we should build a single system to provide attitude support for multiple missions similar to the Goddard Trajectory Determination System (GTDS), which provided orbit determination for all missions. But attempts to build similar all-purpose systems for attitude support failed because specific requirements for attitude data processing were too complex and too varied. This resulted in a plan to build a separate attitude ground-support system (AGSS) for each spacecraft.

The demand for this work was also increasing. In 1973, we were facing the challenge of building 10 AGSSs that would provide similar services for 10 different spacecraft in only 6 years. Clearly, a way had to be found to minimize the amount of software developed, and the experts' knowledge somehow had to be captured, systematized, and made accessible across many systems.

Reuse of Code

Although each mission's attitude requirements were unique, as a whole, all attitude systems were required to perform the same basic functions and provide the same kinds of service. Given that certain parts of an attitude system should be the same or very similar in all of the future systems, it was obvious that, with some up-front planning, those similar parts could be developed once and reused in all subsequent systems. In many ways, this common-sense insight, gleaned by the CSC/GSFC team, was the beginning of real reuse at CSC.

We worked closely with our GSFC customer to develop a reuse strategy based on reuse of proven software components such as coded utilities and design templates. Obviously, this strategy would dramatically reduce development effort, and it would extend individual experience across system boundaries. The first step in this early reuse program was to identify three elements common to future systems:

- **Low-level utilities.** Coordinate transformations, matrix operations, Sun-vector calculations, and the like are identical in all flight dynamics applications.
- **User interface.** All AGSSs would be used by the same mission operations team; there was a strong desire for all AGSSs to have a consistent look and feel and use the same operating procedures.
- **High-level system structure.** The specific hardware and the science requirements of each spacecraft would dictate the details of the software system, but the functions that AGSSs were to perform were basically the same; these functions could therefore be allocated to subsystems and a standard high-level system structure could be created.

We then set out to capture people's knowledge in products by generating the low-level utilities described above and, later, reusable functional packages. To build these products into easily reusable code, we took four specific steps that addressed these areas and led to a significant amount of reuse over the next decade.

First, we assembled the basic utility functions into a library. This library was simply a collection of existing routines, kept in one place so that programmers could find them and copy them easily when they were needed. Usually, these low-level routines could be reused without change.

The library approach alone still did not capture application-specific knowledge in its entirety. However, the early success of the library showed that the concept was basically sound, and it indicated that the process should be expanded to include reusable functional packages—bigger pieces of software like orbit propagators and differential correctors that could be reused in most of the systems. These functional packages would capture higher-level knowledge about the application and make it more easily available to less-experienced personnel.

Second, GSFC sponsored the development of the Graphic Executive Support System (GESS) (Reference 5), which was designed to reside between the operating system and the application programs and

deliver all executive control functions and user interface support for the attitude ground-support software. This system was intended to give consistency, the same look and feel, to the support software for each mission, and indeed it did so. In retrospect, though, this initiative did much more. User-interface support routines had been the objects of endless tinkering and "gold plating," which works contrary to reuse philosophy; GESS effectively limited the scope of the development effort by removing the user-interface support routines from the programmer's domain.

Third, a team of technical people most knowledgeable and experienced in the application domain looked at the requirements for the next four missions and developed a system structure that would support that family of satellites. The high-level AGSS structure that they assembled became the "standard AGSS design template" that was to be used for the following 12 to 15 years (References 6, 7). This template encouraged a high degree of design similarity from one mission to the next, which in turn facilitated code reuse. It was very easy to map individual components in a previous system to the design for a new system; the design brought forth coded units for consideration for reuse rather than forcing the programmer to search for reusable units that might fit in an unrelated design. With this approach, most of the coded units needed to be modified for each mission, but a substantial amount of code was reused that would otherwise have been overlooked. In addition, this high-level structure communicated from one development team to its successor the knowledge of what an AGSS must do, how it is structured and why, how it interfaces with other systems, and how functional requirements can most effectively be allocated to subsystems.

Finally, our managers adopted a philosophy that enforced reuse of the reusable items. Detailed designs were required to indicate which units were going to be reused. High-level designs were required to be based on the standard AGSS structure and were required to use GESS for all user-interface and executive functions. Software reuse was addressed at preliminary and critical design reviews; a system design that did not comply with reuse directives did not pass management review. Managers also required cost estimates to address both new and reused code.

This management enforcement was a key element of the reuse strategy; without it, reuse certainly never would have received the primary emphasis that it did from the mid-1970s through the mid-1980s. With the successful implementation of reuse strategies, the technical side of our corporate culture had been changed; managers and team leaders were now thinking in terms of reusing ideas, designs, and system structures from one mission to another. Code reuse was now more organized, and people were reusing units that other people had created. The percentage of actual code reuse varied depending on how much the standard high-level design needed to be adjusted, but during the first few years of purposeful reuse, the average rose from the initial 10 percent to an average of 25 percent.

Code reuse, then, proved successful as a strategy for capturing the available application knowledge, knowledge about what was being built. Yet it could not capture knowledge about how the organization did business. Control of process was the next logical step toward maximizing the effectiveness of reuse as a means of increasing general efficiency and productivity.

Reuse of Process

CSC's distinctive corporate culture during the 1970s, although it implicitly included reuse, was not framed around any very definite corporate process. It operated on an informal process that had developed spontaneously, without specific directives and without documentation. This process was transmitted by on-the-job experiences and by word of mouth; each project leader was free to follow it or to improvise new processes.

By formalizing the process, we expected to standardize the procedures used across all projects. This would minimize the training needed by personnel as they moved from one project to the next, and it would facilitate the training of new personnel. In addition to these obvious technical advantages, it would also help managers estimate and control costs and schedules.

The first step in this process was methodologically the same as had been used before: we began

capturing, recording, and reusing the organizational experience of our personnel. In the case of process, these activities took the form of capturing weekly project metrics in a corporate database, formally recording lessons learned in software development project history reports, and maintaining a library of this information accessible to managers and project leaders of new projects. We observed the current practices and gathered informal standards and procedures that project leaders had drawn from their experiences and were then using to train their teams.

This information gave us a broad and accurate view of the existing process of software development in the flight dynamics domain. By 1984, the CSC/GSFC team formalized the process in its *Recommended Approach to Software Development* (Reference 8) and its *Manager's Handbook* (Reference 4), and, at about the same time, CSC developed its own corporate process document, the *Digital System Development Methodology (DSDM)* (References 9, 10). Because both of these methods were based on work actually being done in the organizations at that time and on lessons learned from previous work, they recorded and systematized the common-sense procedures that worked rather than prescribing an ideal standard without reference to experience. Both defined the management process as well as the technical development process. Because the process in both cases was based on current practices, reuse was an integral component from the beginning. In fact, reuse was by then seen as so basic to the process that it was embodied as an underlying theme in these documents rather than singled out as a separate topic.

Managers at all levels enforced the fully documented process. It was used (and reused) on all subsequent projects. During this period, we were able to build larger, more complex systems (Reference 6) even though we were experiencing higher-than-normal turnover of our experienced application experts. Design templates, reusable code, and a documented process provided a well-defined environment in which new personnel quickly became productive contributors.

Reuse of Requirements and Design

The same drivers that had spurred the formulation of reuse programs during the 1970s prompted us to improve them during the 1980s. In those years, NASA gradually abandoned its practice of flying numerous, highly specific satellites in favor of flying fewer satellites of greater complexity. This new policy required larger, more complex ground systems, but budgets did not immediately rise in response to this increased demand. Further refinement of proven reuse policies promised to build on past successes, reduce the company's adaptation time, and, most important, boost the efficiency of production to meet the challenge of the period's unprecedented requirements.

SEL studies at the time showed that verbatim code reuse was by far the most beneficial form of reuse, but surveys also showed that the functional packages reused from system to system were almost always modified before reuse (References 6, 11); subsequent studies related this trend to the fact that requirements and specifications were customarily written for a specific spacecraft, a practice that introduced variations that had to be reflected in each mission's software.

In 1987, an opportunity to use this knowledge presented itself when two major missions were scheduled for concurrent development, the Upper Atmosphere Research Satellite (UARS) and the Extreme Ultraviolet Explorer (EUVE). Because NASA was beginning to standardize spacecraft design at that time, the requirements for EUVE were largely a subset of those for UARS, but variations in the specifications of the two would have caused considerable code modification, resulting in typical system development costs. Our project managers, committed to reuse and aware of the potential cost savings, agreed to build the system for EUVE through verbatim reuse of major components built for UARS, addressing reuse from the very beginning of the life cycle.

We began by modifying the UARS functional specifications to satisfy the requirements for both missions, and thereby for a whole family of similar spacecraft. We then focused on designing a multi-

mission, generalized system. In the past, our focus had been on a reusable high-level design and verbatim reuse of the lowest-level utilities, but we were now addressing reuse throughout the entire system; functional packages and major components were specified and designed to serve multiple missions without modification. Functional packages were partitioned to map to actual hardware elements on the spacecraft, and data interfaces were generalized and encapsulated with related functions; this approach created a set of building blocks that could be easily reconfigured to support another mission.

One of the key elements of our generalized systems approach was that we made no attempt to generalize those parts of the systems that were highly variable from one mission to the next. As Figure 4 shows, this approach assumes that some new highly specific components would be developed for each system, while the larger generalized part of the system would be reused without change.

This approach has proven very successful. It has been used to develop three different families of systems across the SEAS contract: AGSSs, attitude telemetry simulators, and POCCs. At present in Code 550, two versions of the generalized AGSS have been built to support satellites with two different attitude control systems. As shown in Table 1, these systems have been reused to support three missions and are planned to be reused for at least three more. The generalized attitude telemetry simulator developed for UARS has also been reused to support two other missions. It will support two future three-axis-stabilized satellites. In addition, the Transportable POCC (TPOCC), a generalized system built in Code 510, has already been reused to support two satellites, and it will be used to support five others. All of these generalized systems show verbatim code reuse in the range of 65 percent to 85 percent.

It is important to note that the generalized approach seems to be language and platform independent. Figure 5 shows data from three subsequent systems that we built in each of the three different system families. At the top are attitude telemetry simulators, in the middle are AGSSs, and at the bottom are POCCs. All of these generalized systems show the same favorable trends – in each case, as verbatim reuse goes up, the cost to develop goes down, and the error rates drop, indicating an increase in quality.

Table 1. Reuse of Generalized Systems

Generalized System*	Developed for	Reused for	Future Reuse
MTASS AGSS	UARS	EUVE SAMPEX	SOHO TOMS
MSASS AGSS	WIND	POLAR	FAST
TELEMETRY SIMULATOR	UARS	EUVE SAMPEX	SOHO TOMS
TPOCC	GENERAL USE	SAMPEX ICE/IMP	FAST SWAS SOHO TRMM XTE

*See Acronym List

Although each of these system families was developed in a different language and on different platforms, they all used some degree of object-oriented design (OOD). We used formal object-oriented techniques to develop the telemetry simulators using Ada on VAX systems, where the designers made extensive use of Ada generics to produce reusable code (Reference 12). On the other hand, the TPOCC family of systems, developed in a C/UNIX environment, and the AGSS generalized systems, developed in FORTRAN, did not formally apply object-oriented techniques. However, their mapping of software components to spacecraft hardware objects and their encapsulation of data with the functions that use them are evidence of object-oriented thinking and organization.

As is common with major changes in any business, this higher level of reuse introduced a new set of challenges. Configuration management, in particular, took on a whole new level of importance. Systems no longer "owned" the code they were reusing. Introducing a change to a commonly reused component now had the potential of affecting many systems. Proposed changes to reusable components had to be controlled carefully, and changes to shared code necessitated testing all systems that used it. We addressed these issues by setting up formal procedures (Reference 13) and setting up configured libraries and official mainte-

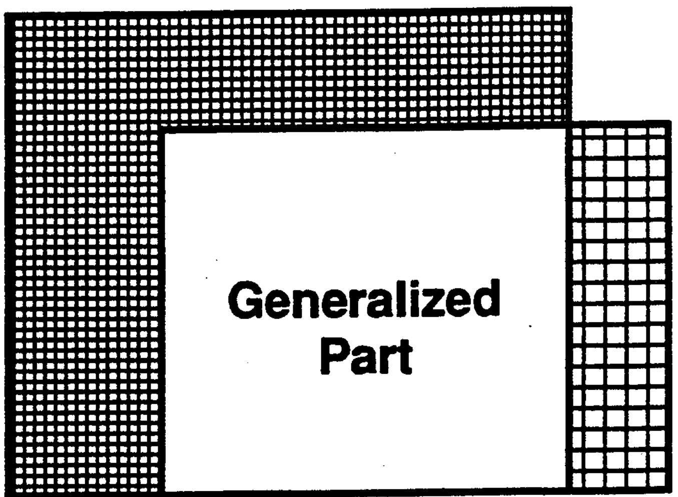


Figure 4. Structure of Generalized Systems

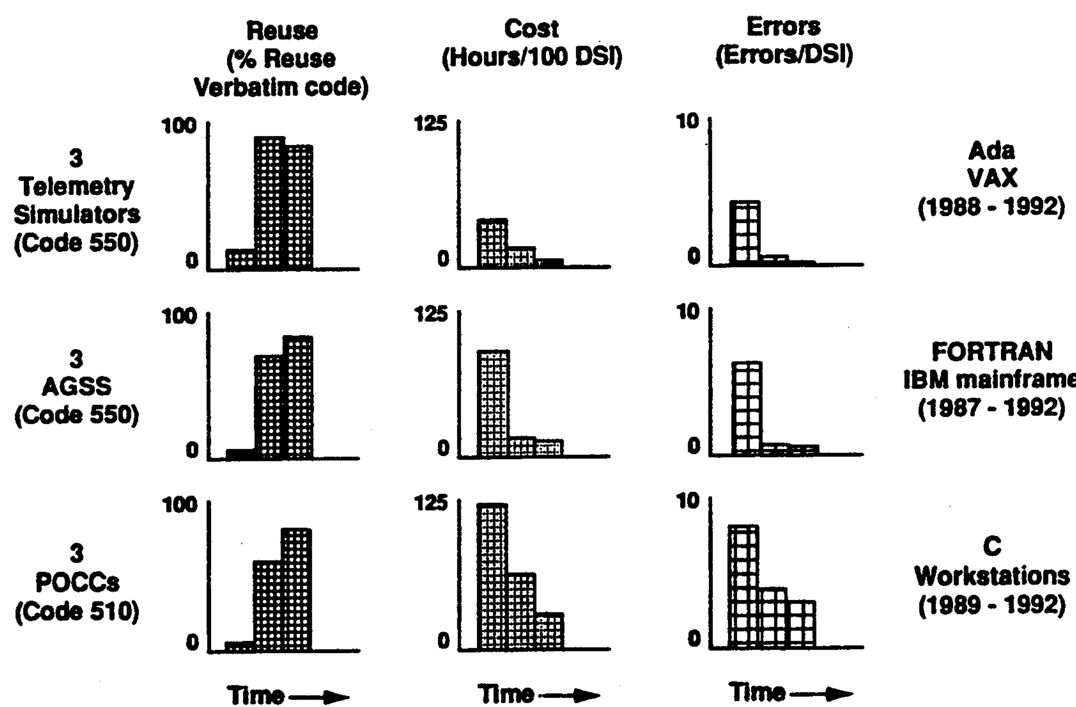


Figure 5. Results of Generalized Systems

nance groups responsible for managing, evolving, and certifying the reusable code. This has been done at two levels.

First, reusable source libraries have been set up for the widely used low-level utilities, where the emphasis is on making the components and functional packages as generally reusable and as efficient as possible; units are thoroughly tested and certified but they are not tested in the context of every reusing system. Conversely, generalized system parts are maintained and enhanced totally within the context of the family of systems they support. Changes are driven by either the needs of a future reusing system or the changing needs of the current reusing systems, and changes are always regression tested in the context of every reusing system before being certified.

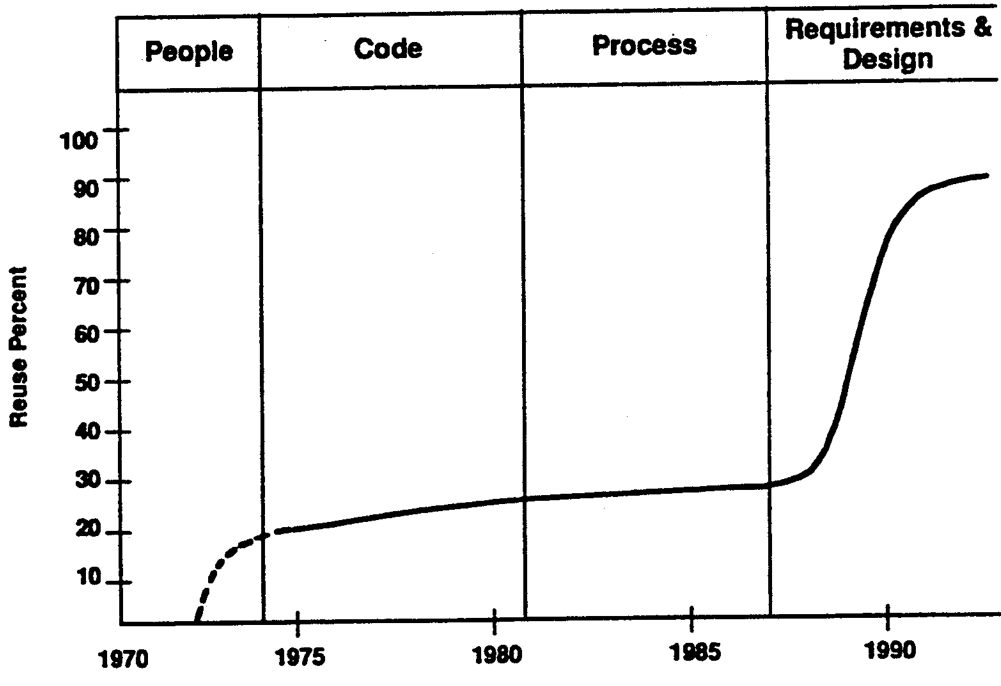
Our new approach to reuse also called for further refinement of our process. From 1990 through 1992, the SEL *Recommended Approach* (Reference 8) and the *Manager's Handbook* (Reference 4) were revised to explicitly address reuse; their scope was expanded to include the Requirements Definition phase that we now understood to be so critical to the reuse process. A two-part reuse process was clearly defined that now addressed enabling reuse (creating reusable software) as well the reusing process itself. In addition, more specific guidance was given to managers on managing reuse.

Future Reuse of Architectures

Today, we have begun to move to the next stage of requirements and design reuse. We have begun to build some reusable architectures from which new systems will be configured rather than developed from a larger set of standardized reusable system pieces (Reference 14). This is an attempt to eliminate the configuration-control and maintenance problems that are associated with the generalized systems.

BUSINESS BENEFITS OF REUSE

Figure 6 shows overall code reuse plotted on a time scale and mapped to the reuse levels. On average, as we have seen from Figure 2, costs decreased as



Note: Percent of code reused with less than 25% modification. Data from 32 attitude systems.

Figure 6. Measured Code Reuse

reuse increased. Looking at the trends within each of the reuse levels, the increase in reuse shows two major incremental jumps. The first payoff came when we began reusing code, at reuse levels of 20 to 25 percent. Although there was little change in the amount of code reuse as we defined and began reusing the process, this effort provided the foundation for the move to the next level by establishing a stable working environment and organization.

The second quantum leap occurred, and the big payoff came, in the late 1980s when we began reusing the generalized systems at the Requirements and Design level. This sudden increase in reuse, based as it is on five data points, is not a spike; there was little change in reuse percentages during the early years of this period because this period was spent in building generalized systems for reuse. When these systems were actually reused, overall reuse increased dramatically.

CONCLUSIONS

There are five major contributors to CSC's success with reuse in this environment: management commitment, process maturity, organizational maturity, judicious use of new technology, and a tight focus on creating and reusing products within the application domain.

- **Management commitment.** Managers looked for innovative technical and management solutions to respond to business challenges, such as increasing workloads and shrinking budgets. Seeing the potential benefits, our managers became committed to a reuse strategy in the mid-1970s and they have maintained their strong commitment to the present. The development of a successful reuse program required solid management judgment to balance periods of innovation, in which the organization evaluated and adopted new technical approaches, and periods of managed reuse, during which the organization focused on reaping the benefits of its investment. At each point, the business needs and goals of the next 5 years drove the management focus. Typically, new approaches would be considered and pilot projects started 3 to 5 years before we would need to meet an anticipated challenge. In each cycle, managers were committed to both building reusable products and then seeing that they were reused.
- **Process maturity.** As the development process matured, it provided an increasingly stable environment; developers could focus more on better technical solutions and less on the mechanics of how the job should be done. The process itself, which was tailored to the specific domain, helped perpetuate the reuse philosophy as new people joined the organization. In its current form, the *Recommended Approach* (Reference 8), which explicitly calls out reuse activities throughout the life cycle, serves as a driver for further culture change as we formally integrate the two-part reuse process into our standard way of doing business.
- **Organizational maturity.** Our organization has matured to meet the growing technical challenges of satellite ground support. Twenty years ago, our organization was small; now, it is far more extensive, including more than 10 times as many professionals as it did at first. Individual people working independently have been superseded by well-integrated teams of software engineers and application specialists; highly personal working methods have been replaced by a formalized and unified approach to software development.

  We have matured from an organization dependent on the knowledge of key personnel to one that draws on an extensive and organized store of experience. Talented people remain the most critical element of our business. Yet, as we have matured, we have replaced our dependence on the specific talents of a few people with a reliance on the proper skill mix of a team whose members can be changed to meet changing organizational needs.

  Our organization has also matured in a way that facilitated the development of software reuse in much the same way that a programmer's perspective expands over time. We began by looking at code, widened our focus to design, and most recently began viewing the problem from the requirements perspective. Each time we broadened our perspective and expanded the problem domain, we saw ways to improve the

previous level and a whole new array of possibilities for the next.

- **Judicious use of new technology.** Throughout this period, we investigated and applied only those new technologies that made sense in our environment and were well suited to our process. For example, because of the highly scientific nature of our application domain, buying a large library of generally reusable computer-science utilities would have little impact on this environment, whereas the application of object-oriented techniques and domain analysis showed great potential for large-scale benefits. There were many opportunities during this period to either build or buy an automated reusable software library. We chose to concentrate on figuring out how to create reusable components rather than concentrating on automating a process that we didn't yet understand.

- **Focus on a specific application domain.** The most reusable items were produced by people who understood the local environment and the specific application domain. In every case where we successfully built something that was highly reusable, application experts had considered what they themselves (as future reusers) would need in order to build systems from reusable parts. To ensure that the needs of both user and reuser were being addressed, multiple missions were considered at once and, typically, at least two systems (one that would create the reusable parts and one that would reuse the parts) were developed simultaneously. This kept the development team focused on reusability within the domain by involving active (not just potential) reusers in all reviews and trade off decisions.

Based on our experience, we believe that there is a reuse maturity model consisting of levels that every organization must progress through in developing a successful reuse program. At CSC, it has taken us 20 years to come this far. Certainly the maturity rate of the software engineering industry as a whole contributed to the length of this period; another organization beginning today would no doubt accomplish this much more quickly. But it is clear that a successful reuse program cannot be put in place overnight. A step-by-step approach is essential. First, focus on developing a mature organization; capture and reuse the knowledge and experience of people and projects. Next, develop and mature a reusable process that is specifically tailored or adapted for the specific domain. Then, concentrate on building and reusing software engineering products ranging from code units to requirements and design and beyond.

ACRONYM LIST

AGSS – attitude ground support system

EUVE – Extreme Ultraviolet Explorer

FAST – Fast Auroral Snapshot Tool

ICE/IMP – International Cometary Explorer/ Interplanetary Monitoring Platform

ISTP – International Solar-Terrestrial Physics

MSASS – Multimission Spin-Stabilized Attitude Support System

MTASS – Multimission Three-Axis-Stabilized Attitude Support System

SAMPEX – Solar, Anomalous, and Magnetospheric Particle Explorer

SOHO – Solar and Heliospheric Observatory

SWAS – Submillimeter Wave Astronomy Satellite

TOMS – Total Ozone Mapping Spectrometer

TPOCC – Transportable Payload Operations Control Center

TRMM – Tropical Rainfall Measurement Mission

UARS – Upper Atmosphere Research Satellite

XTE – X-Ray Timing Explorer

ACKNOWLEDGEMENT

We would like to thank Kevin Orlin Johnson for his help in preparing this paper.

REFERENCES

1. NASA/GSFC Software Engineering Laboratory, SEL-77-001, *Proceedings From the First Summer Software Engineering Workshop*, August 1976

2. ———, SEL-81-104, *The Software Engineering Laboratory*, D. N. Card, F. E. McGarry, G. Page, et al., February 1982

3. ———, SEL-82-1106, *Annotated Bibliography of Software Engineering Laboratory Literature*, L. Morusiewicz and J. Valett, November 1992

4. ———, SEL-84-101, *Manager's Handbook for Software Development (Revision 1)*, L. Landis, F. E. McGarry, S. Waligora, et al., November 1990 (previous version published in 1984)

5. Computer Sciences Corporation, CSC/SD-75/6057, *Graphic Executive Support System (GESS) User's Guide*, J. E. Hoover et al., August 1975 (updated 1976, 1977, and 1979)

6. ———, CSC/TM-89/6031, *A Study on Size and Reuse Trends in Attitude Ground Support Systems (AGSSs) Developed for the Flight Dynamics Division (FDD) (1976–1988)*, D. Boland et al., February 1989

7. J. Wertz, ed., *Spacecraft Attitude Determination and Control*, Dordrecht, Holland: D. Reidel Publishing Company, 1978

8. NASA/GSFC Software Engineering Laboratory, SEL-81-305, *Recommended Approach to Software Development (Revision 3)*, L. Landis, S. Waligora, F. McGarry, et al., June 1992 (previous versions published in 1981 and 1983)

9. Computer Sciences Corporation, *Digital System Development Methodology, Version 3.0*, December 1989 (previous versions published in 1981 and 1984)

10. ———, *DSDM® Digest: Digital System Development Methodology*, Version 3.0, E. M. Markson, Sr., T. L. Clark and M. T. Speights, December 1989

11. ———, CSC/TM-87/6062, *Profile of Software Reuse in the Flight Dynamics Environment*, D. Solomon and W. Agresti, November 1987

12. *Proceedings of TRI-Ada 1989*, "Using Ada to Maximize Verbatim Software Reuse," M. E. Stark and E. W. Booth, October 1989

13. Computer Sciences Corporation, CSC/TR-90/6083, *Transportable Payload Operations Control Center (TPOCC) Project Support Plan (Revision 3)*, September 1992

14. Goddard Space Flight Center, Flight Dynamics Division, 550-COMPASS-102, *Combined Operational Mission Planning and Attitude Support System (COMPASS) High-Level Requirements, Architecture, and Operation Concepts*, R. DeFazio (GSFC) et al., May 1991

Maximizing Reuse:
Applying Common Sense and Discipline

Sharon Waligora and Jim Langston
Computer Sciences Corporation

Agenda

- Evidence of our reuse success
- Evolution of reuse in our environment
- Recommendations

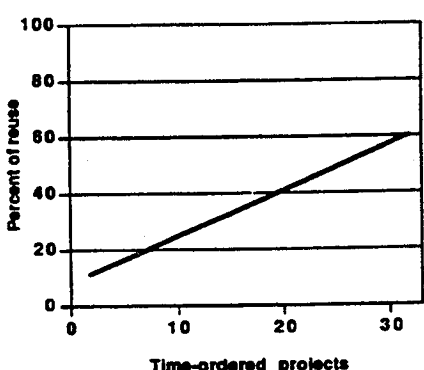
10008893G- 2

Environment

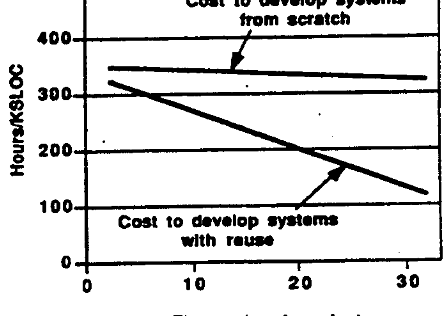
Organization	CSC SEAS contract and predecessor contracts
Customer (since 1970)	GSFC Mission Operations and Data Systems Directorate (Code 500)
Application	Scientific satellite ground support - Attitude systems (Code 550) - Payload Operations Control Centers (Code 510)
Languages	FORTRAN, Ada, C
Computing Environment	HSD 8063 (IBM 3083), VAX 8820, VAX-11/780
Average System Size	180 KSLOC
Average Project Duration	2 years

10006893G- 3

Percent of Reuse *Cost to Deliver*



Reuse has increased from 1978-1992.

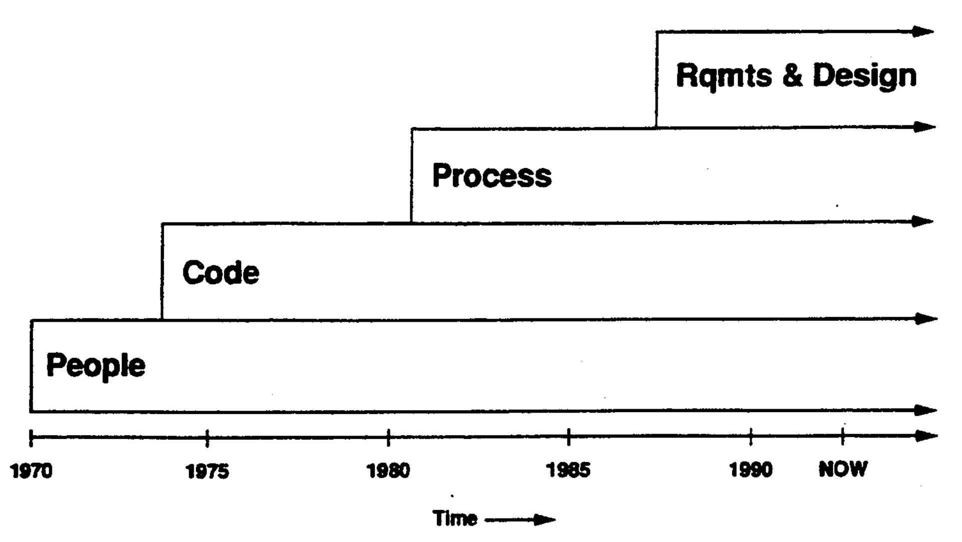


It costs 70-80% less to deliver reused portions of software.

Based on 32 attitude systems

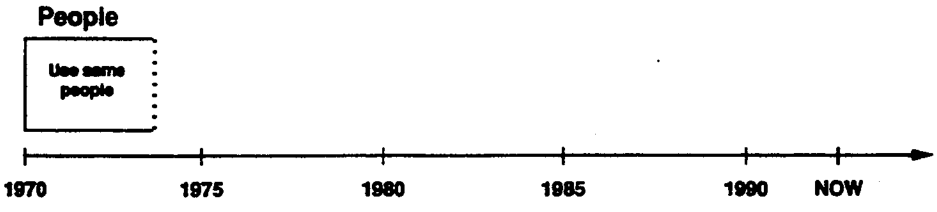
10006893G- 4

Levels of Reuse



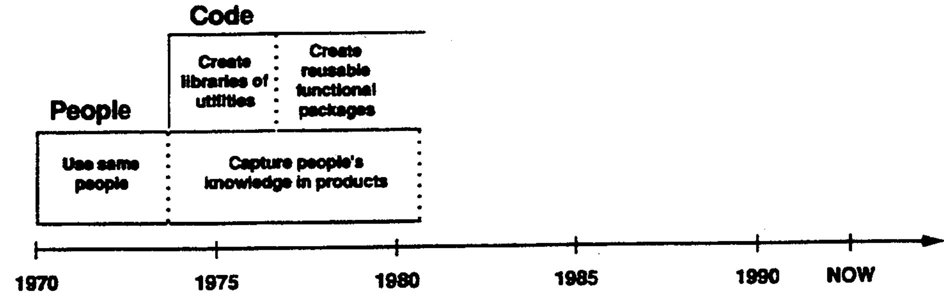
10006893G- 5

Levels of Reuse



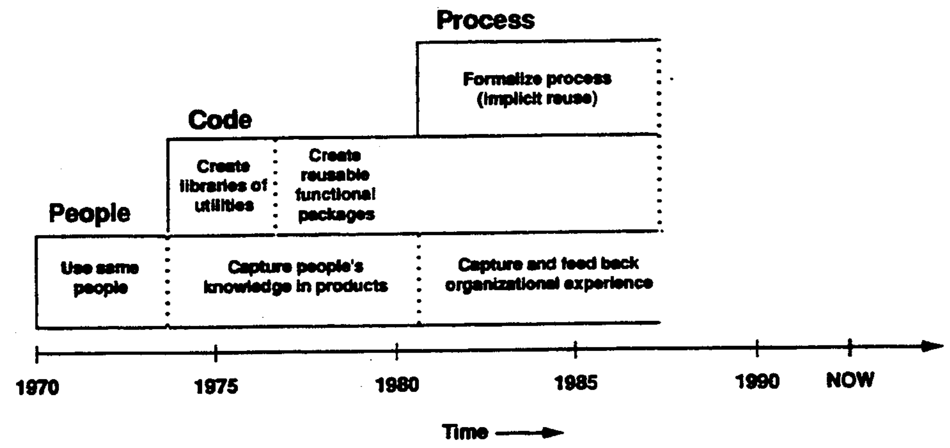
10006893G- 6

Levels of Reuse



10005893G- 7

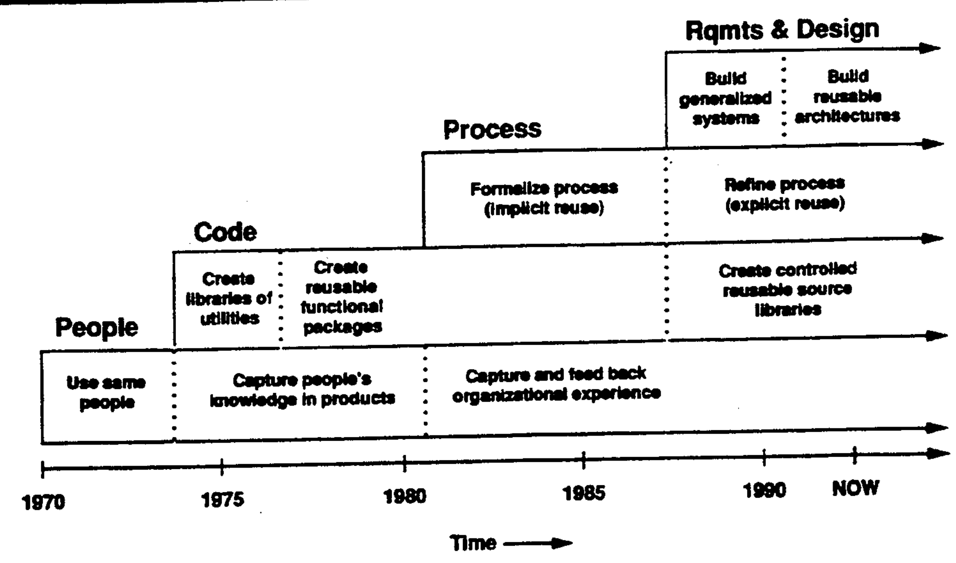
Levels of Reuse



Computer Sciences Corporation
System Sciences Division

10005893G- 8

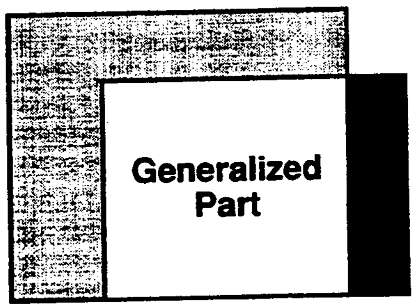
Levels of Reuse



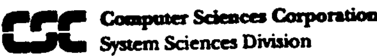
Computer Sciences Corporation
System Sciences Division

10006893G- 9

What Is a Generalized System?



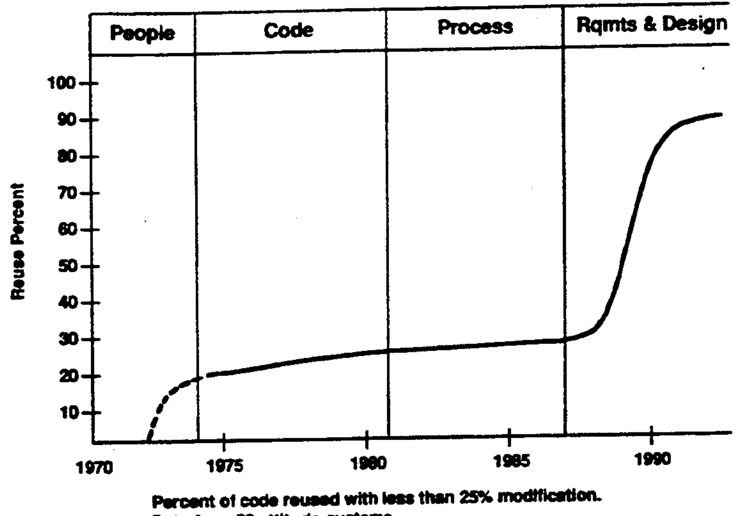
- Generalized part reused verbatim in both systems
- Specialized code for System A
- Specialized code for System B



Computer Sciences Corporation
System Sciences Division

10006893G- 10

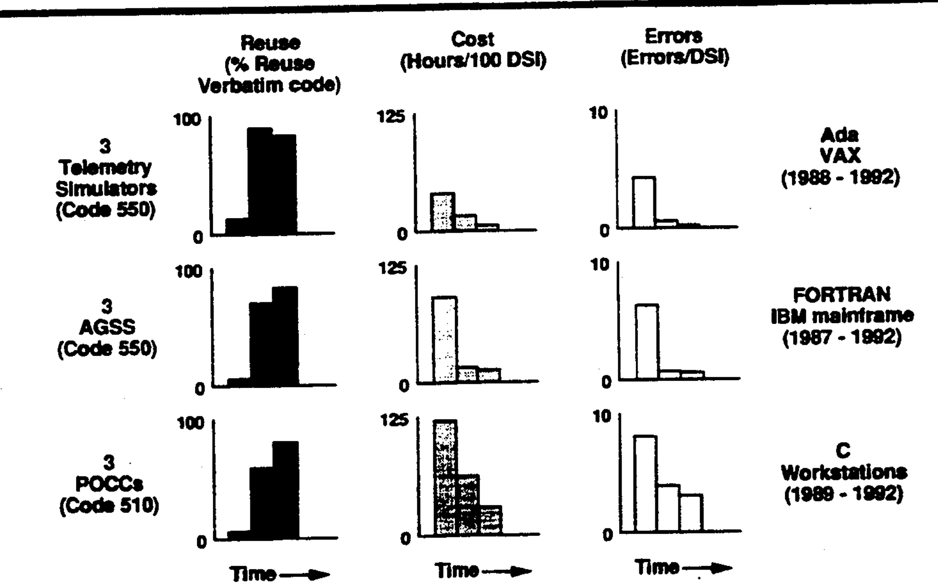
Measured Code Reuse



Percent of code reused with less than 25% modification.
Data from 32 attitude systems.

10006893G- 11

Generalized Systems Reuse Results



Computer Sciences Corporation
System Sciences Division

10006893G- 12

Recommendations

- Start with the basics
- Focus on business/customer needs
- Tailor the process for your organization
- Learn from experience
- Make a conscious effort to expand to the next level