

The Improvement Cycle: Analyzing Our Experience

Rose Pajerski
NASA, Goddard Space Flight Center
Flight Dynamics Division
Greenbelt, MD 20771
(301) 286-3010
rose.pajerski@gsfc.nasa.gov

Sharon Waligora
Computer Sciences Corporation
4061 Powder Mill Rd.
Calverton, MD 20705
(301) 572-3751
swaligor@csc.com

Abstract

NASA's Software Engineering Laboratory (SEL), one of the earliest pioneers in the areas of software process improvement and measurement, has had a significant impact on the software business at NASA Goddard. At the heart of the SEL's improvement program is a belief that software products can be improved by optimizing the software engineering process used to develop them and a long-term improvement strategy that facilitates small incremental improvements that accumulate into significant gains. As a result of its efforts, the SEL has incrementally reduced development costs by 60%, decreased error rates by 85%, and reduced cycle time by 25%. In this paper, we analyze the SEL's experiences on three major improvement initiatives to better understand the cyclic nature of the improvement process and to understand why some improvements take much longer than others.

Background

Since 1976, the Software Engineering Laboratory (SEL) has been dedicated to understanding and improving the way in which one NASA organization, the Flight Dynamics Division (FDD) at Goddard Space Flight Center, develops, maintains, and manages complex flight dynamics systems. It has done this by developing and refining a continual process improvement approach that allows an organization such as the FDD to fine tune its process for its particular domain. Experimental software engineering and measurement play a significant role in this approach.

The Software Engineering Laboratory (SEL) is a partnership of NASA Goddard's Flight Dynamics Division (FDD), its major software contractor, Computer Sciences Corporation (CSC), and the University of Maryland's (UM) Department of Computer Science. The FDD primarily builds software systems that provide ground-based flight dynamics support for scientific satellites. They fall into two sets: ground systems and simulators. Ground systems are midsize systems that average around 250 thousand source lines of code (KSLOC). Ground system development projects typically last approximately 2 years. Most of the systems have been built in FORTRAN on mainframes, but recent projects contain subsystems written in C and C++ on workstations. The simulators are smaller systems averaging around 60 KSLOC that provide the test data for the ground systems. Simulator development lasts between 1 and 1.5 years. Most of the simulators have been built in Ada on a VAX computer. The project characteristics of these systems are shown in Table 1. The SEL is responsible

for the management and continual improvement of the software engineering processes used on these FDD projects.

Table 1. Characteristics of SEL Projects

Characteristics	Applications	
	Ground Systems	Simulators
System Size	150 - 400 KSLOC	40 - 80 KSLOC
Project Duration	1.5 - 2.5 years	1 -1.5 years
Staffing (technical)	10 - 35 staff-years	1 - 7 staff-years
Language	FORTRAN, C, C++	Ada, FORTRAN
Hardware	IBM Mainframes, Workstations	VAX

The SEL process improvement approach shown in Figure 1 is based on the Quality Improvement Paradigm [Reference 1] in which process changes and new technologies are 1) selected based on a solid understanding of organization characteristics, needs, and business goals, 2) piloted and assessed using the scientific method to identify those that add value, and 3) packaged for broader use throughout the organization. Using this approach, the SEL has successfully established and matured its process improvement program throughout the organization.

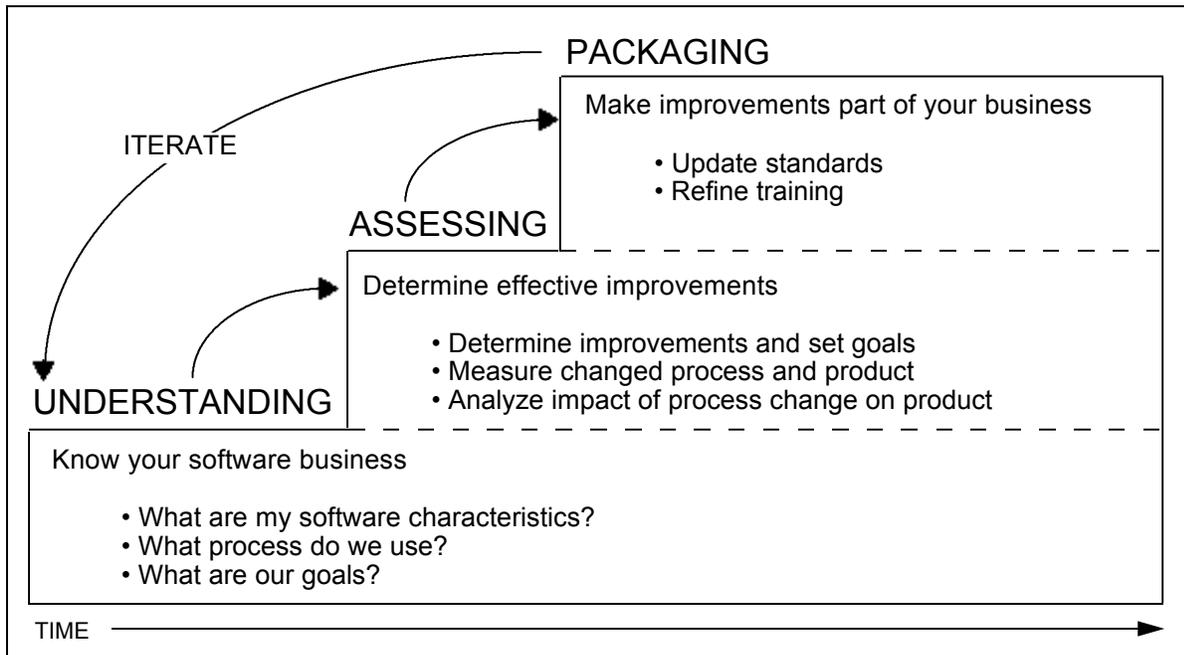


Figure 1. SEL Process Improvement Paradigm

The SEL organization consists of three functional areas: software developers, software engineering process analysts, and data base support (Figure 2). The largest part of the SEL is the 150-200 software personnel who are responsible for the development and maintenance of over 4 million

source lines of code (SLOC) that provide orbit and attitude ground support for all GSFC missions. Since the SEL was founded, software project personnel have provided software measurement data on over 130 projects. This data has been collected by the database support personnel and stored in the SEL data base for use by both the software project personnel and the process analysts. The process analysts are responsible for defining the experiments and studies, analyzing the data and producing reports. These reports affect such things as project standards, development procedures, and how projects are managed. The data base support staff is responsible for entering measurement data into the SEL data base, quality assuring the data, and maintaining the data base and its reports.

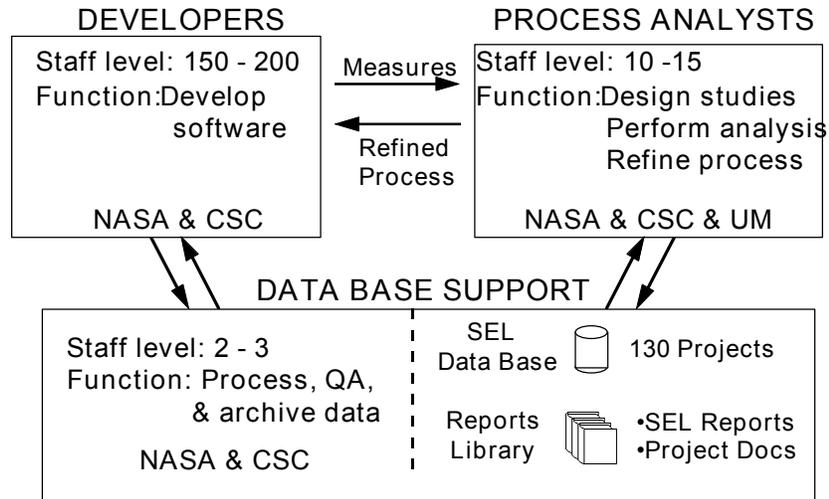


Figure 2. SEL Organizational Structure

Improvement Cycles

Although the improvement process is a never ending endeavor, it is cyclic in nature. At the SEL, improvement cycles operate within the context of the SEL process improvement paradigm. Each improvement cycle tends to focus on a single organizational goal and only one or two process or technology changes that address that goal. Often these build on earlier experimental results. Each SEL improvement cycle has four major steps:

1. Each improvement cycle begins with setting improvement goals based on the current business needs and strategic direction of the organization. Based on a solid understanding of the problem domain (application), the development environment, and the current process and product characteristics of the organization, process analysts identify leverage areas, i.e., software process or product characteristics that could have a significant impact on the overall performance of the organization. For example, increasing software reuse would have a high probability of reducing project cost and development cycle time. Therefore, if the business goals are to reduce cost and/or cycle time, increasing reuse would be a reasonable leverage area.
2. The next step is to identify software engineering technologies (processes, methods and/or tools) that are likely to affect the leverage area. For example, object-oriented techniques (OOT) are reported to facilitate reuse. The ultimate goal of this step is to select one technology or process change that has the greatest potential for meeting the improvement goal.

3. The third and longest step of the improvement cycle is to conduct experiments to understand the value and applicability of the new technology in the local organization. Scientific methods are used to pilot the technology on one or more real projects and observe the use and effect of the technology on the development process, products, and project performance. Process analysts use both qualitative feedback and quantitative measurements to evaluate the value of the technology/process change. Key project measurements are compared with those from a control group (similar contemporary projects using the standard process) to assess overall value. Several experiments that successively refine the process/technology may be required before it is ready to deploy.
4. The final step in an improvement cycle is to deploy the beneficial process/technology throughout the organization. This involves integrating the process change/technology into the standard process guidebooks, providing training to project personnel, and providing ongoing process consulting support to facilitate the adoption of the new technology/process change.

Since its inception, the SEL has completed numerous improvement cycles spanning from one to seven years. The amount of time it takes to complete a cycle depends on the maturity and breadth of the technology/process change. Several improvement cycles are usually active at one time; however, they involve different subsets of the organization's projects.

SEL Improvement Examples

In 1985, the SEL set two fairly common improvement goals: 1) reduce the cost of developing software systems, and 2) improve the quality of delivered systems. In 1990, in response to NASA's new emphasis on launching missions more quickly, a third goal was adopted: 3) reduce the cycle time needed to develop new systems. All of these goals were addressed by leveraging different process and technology areas within the context of a unified improvement program.

The following examples illustrate the different approaches taken and results achieved within three representative SEL improvement initiatives. As shown in Table 2, each initiative used a different number of improvement cycles and a somewhat different deployment strategy to achieve the desired results. The number of improvement cycles were driven by the experiment approach and results, while the deployment strategy was selected based on a risk/benefit analysis of the process change using the experiment results.

Table 2. SEL Improvement Examples

Goal	Improvement Initiative	Cycles	Experimentation Approach	Deployment Approach
Reduce Cost	Maximize Reuse	2	Iterative learning of how to apply OO concepts; develop new reuse methods	Full use in highest payback applications (subset of projects)
Increase Quality	Leverage Human Discipline	3	Iterative refinement of existing, external testing techniques and Cleanroom Methodology	Subset of 'best' techniques across all projects
Reduce Cycle Time/Cost	Streamline Testing Process	1	Refine and consolidate local (familiar) processes	Full use across all projects

Example 1: Maximizing Reuse

To reduce costs, the SEL chose to introduce and experiment with two software engineering technologies, the Ada language and object-oriented design (OOD), that had high potential for maximizing software reuse. Experimentation began across a single class of applications, flight dynamics simulators, as the first improvement cycle focused on defining a generalized architecture based on more theoretical OO concepts. Once the developers were able to apply the architecture to their systems, the application scope expanded to include generalizing more elements of flight dynamics systems. The second group of experiments expanded the definition of ‘generalized’ to include reusable specifications which has resulted in a large library of reusable flight dynamics components. Figure 3 shows the experimental focus areas and timeline for these two improvement cycles. Because the early work with OO was more conceptual, several phases of experimentation across different development projects were undertaken prior to deploying the supporting process changes.

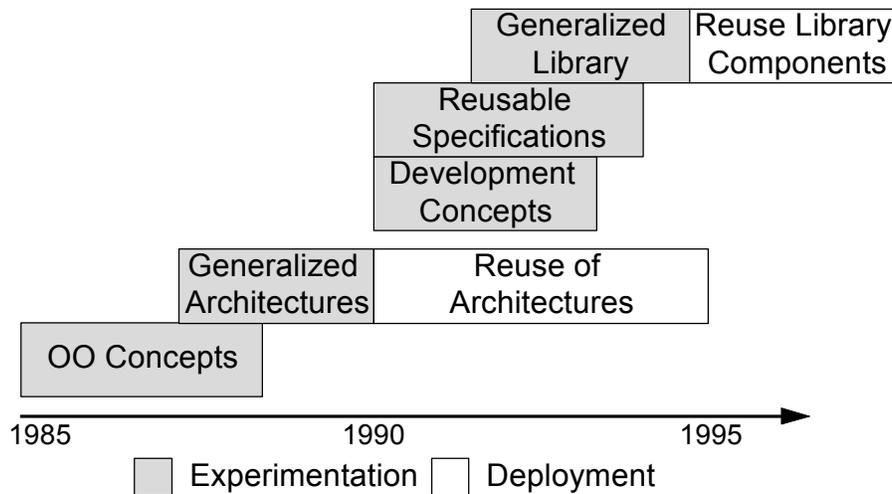


Figure 3. SEL Reuse Improvement Cycle Timeline

Within 4 years, this effort culminated in the first deployment of reusable generalized architectures that have led to a 300% increase in software reuse per system and an overall cost reduction of 55% during the next 4 years [Reference 2]. Further development of these object-oriented concepts has produced a set of reusable specifications and a corresponding component library that promises even greater improvements in 1997 systems. Figure 4 depicts the measured impact to the FDD resulting from these changes.

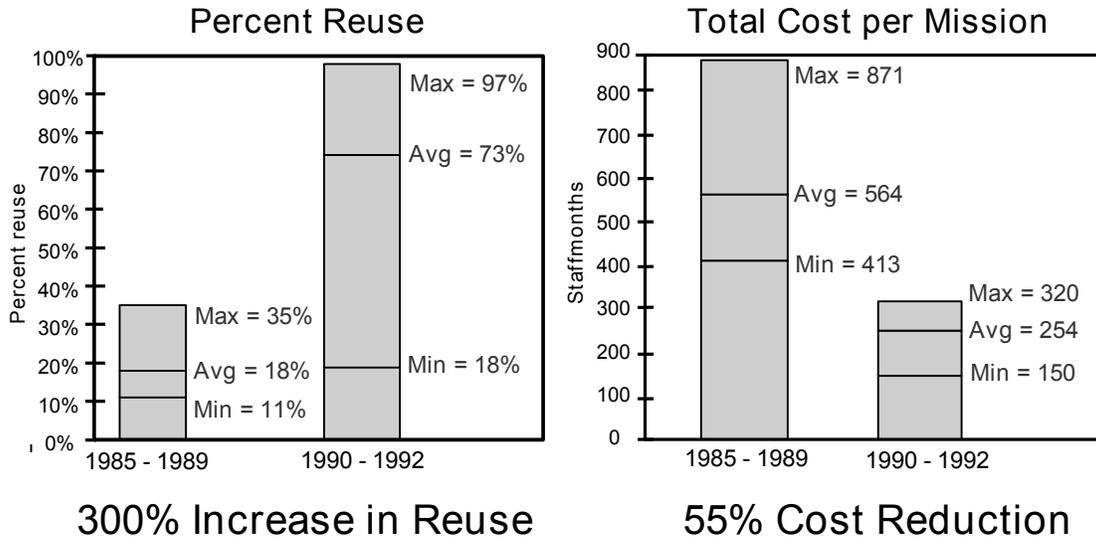


Figure 4. Results of Introducing OOD and Ada

Example 2: Leveraging Human Discipline

Early experimental results showed the positive impact that leveraging the experience and perspective of the individual developer brought to software development. Based on these results, the SEL chose to focus on software engineering methodologies that support human discipline to meet our quality goal [Reference 3]. The first improvement cycle which investigated different testing techniques such as code reading, and unit and functional testing confirmed that those methods which relied on human discipline were the most effective. This led to a significant effort within the SEL to maximize the potential of human discipline by experimenting with the Cleanroom Methodology.

The SEL has completed two improvement cycles over four projects (two large, 2 small) that specifically addressed Cleanroom; the initial SEL Cleanroom project began in 1988, with the fourth and final effort completed this year. The focus of the Cleanroom Methodology is on producing software with high probability of zero defects. The key elements of the methodology include an emphasis on human discipline in the development process via code inspections and requirements classification, and a statistical testing approach based on anticipated operational usage. Development and testing teams operate independently, and all development team activities are performed without on-line testing. Analysis of the first three Cleanroom efforts had indicated greater success in applying the methodology to smaller (< 50K developed lines of code (DLOC)) in-house GSFC projects than to larger-scale efforts typically staffed by joint contractor-government teams. The final Cleanroom project involved development of a large scale system (480K SLOC, 140K DLOC), with the primary study goal to examine it as an additional data point in the SEL's analysis of Cleanroom applicability in the organization, especially in the area of scalability.

The goal of the SEL's Cleanroom study was not to make a decision on adopting Cleanroom in its entirety within the organization, but rather to highlight those aspects of the methodology which had favorable impacts and to incorporate them into the standard approach. This approach of incremental deployment, shown in Figure 5, proved very successful in instilling these changes throughout the organization. Experimentation with Cleanroom raised the general awareness of the organization

regarding quality techniques and discipline-enhancing processes. This emphasis is one of the key reasons for the FDD's steady improvement in reducing development error rates by 85% over a 15 year period as shown in Figure 6.

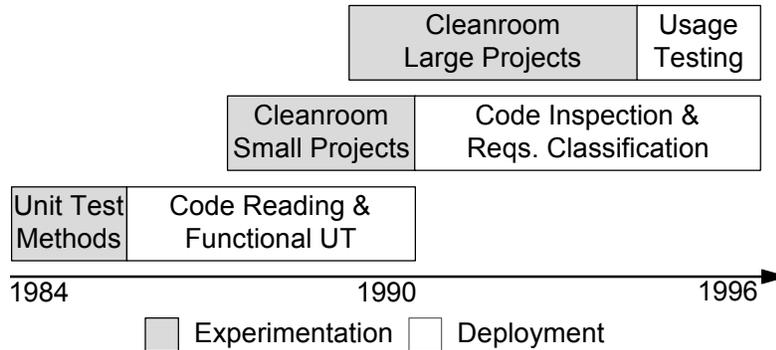


Figure 5. SEL Quality Improvement Cycle Timeline

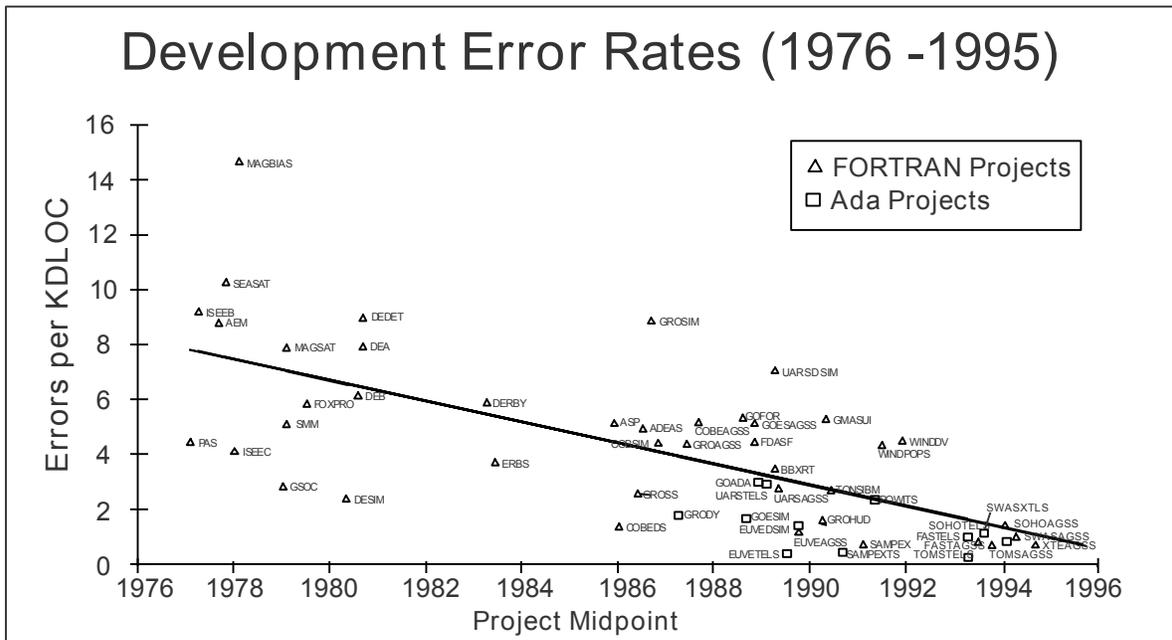


Figure 6. Quality Improvement in the SEL

Example 3: Streamlining the Testing Process

In 1992, the SEL saw the cost of system development was decreasing significantly due to increasing code reuse; however, no corresponding decrease in development cycle time was occurring. In addition, although the cost associated with design and code effort had been reduced, testing costs remained virtually the same. This led to an assessment of the testing processes in use, and resulted in a decision to focus testing in one group. This group, called the independent testers, effectively collapsed two separate testing phases (system and acceptance) conducted by two different groups (developers and users), into one phase (independent) performed by one group composed of

experienced flight dynamics analysts and testers. This change, in both process and organization, was introduced in order to reduce the cycle time required to deliver a system, to improve the efficiency of the testing process, and to do so without sacrificing the quality of any product delivered.

Since this change was limited to one organization which was already heavily involved in defining the new testing process, the experimentation portion was brief and the risk of full deployment was judged to be low (Figure 7). Once the organizational changes were made, process changes were implemented quickly, simultaneously across all current test efforts. The resultant measurements (Figure 8) indicate that independent testing has yielded a definite shift in life-cycle effort distribution, with the testing effort being reduced from 41% to 31% of the total project effort [Reference 5]. Reductions in cycle time on the order of 5% to 20% have been verified with no loss of quality.

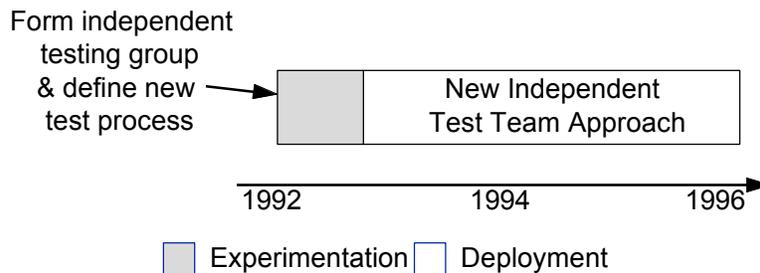


Figure 7. SEL Independent Testing Improvement Cycle Timeline

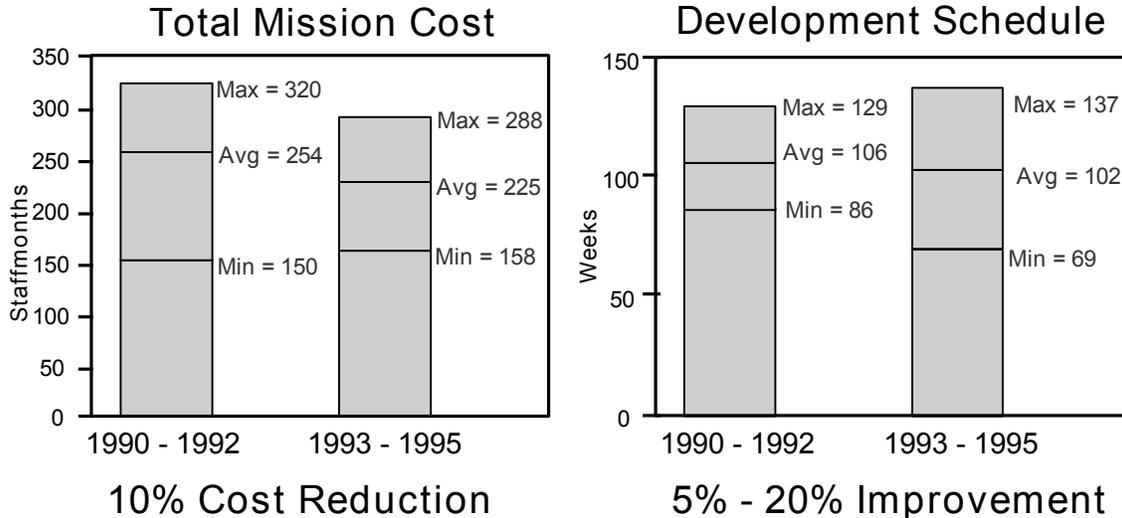


Figure 8. Results of Streamlining the System Testing Process

Measuring Overall Improvement

Each of the above initiatives resulted in measurable improvement, however each was measured in isolation on a particular set of projects. On a long-term, continual improvement program, it is important to periodically assess how the organization is doing as a whole. To make this assessment

and to update the organizational characteristics that will drive future improvement decisions, the SEL periodically computes an organizational baseline. This consists of key measurements that characterize the performance of the project organization over a specified time period and represent the organization's ability to perform similar work in the future.

We use a fairly small set of baseline measurements to evaluate improvement. They include total cost, total duration, development error rate, reuse percentage, cost per new line of code, and cost per delivered line of code. For each baseline measurement, a maximum, a minimum, and a project mean are computed for a particular time period, referred to as the baseline period. Overall improvement in each measurement is determined by comparing the means of two baseline periods, i.e., (current mean - previous mean) / previous mean.

Since 1985, the SEL has computed three baselines to overall improvement. Figure 9 shows when these baselines were computed in relation to the three examples discussed earlier. Notice that baselines were computed a few years after a set of improvements were deployed allowing time for projects to use the improved process. Figures 10 and 11 show how the results of the individual initiatives combined to make significant overall improvements.

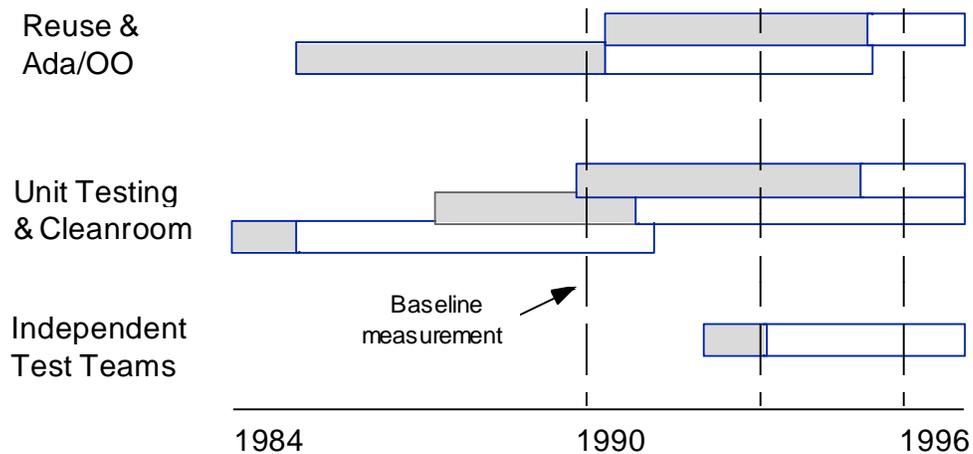


Figure 9. Improvement Cycle Timelines

The SEL's recently completed 1996 organizational baseline shows across-the-board improvement in all measurements.

- Average mission cost decreased by 15% when compared with the 1993 baseline, totaling a 60% overall reduction in mission cost since 1985. (Figure 10)
- The cost to develop a line of new code decreased nearly 35% since 1993. (There had been no previous improvement in this measure.)
- Ground system projects saw a modest 7% reduction in project cycle time, while simulators experienced a 20% reduction since 1993. (Figure 8)
- Error rates continued to drop, with a 40% reduction in development error rates since 1993. This combines with earlier improvements to total an 85% drop in development error rates over the past 10 years. (Figure 6)

- After the initial 300% increase in reuse seen in the 1993 baseline, software reuse remained high with an average of 80% on all projects; however, the minimum amount of reuse has now risen from 18% in the 1993 baseline project set to 62% in the recent project set, demonstrating a much more consistent use of reusable products in the SEL. (Figure 11)

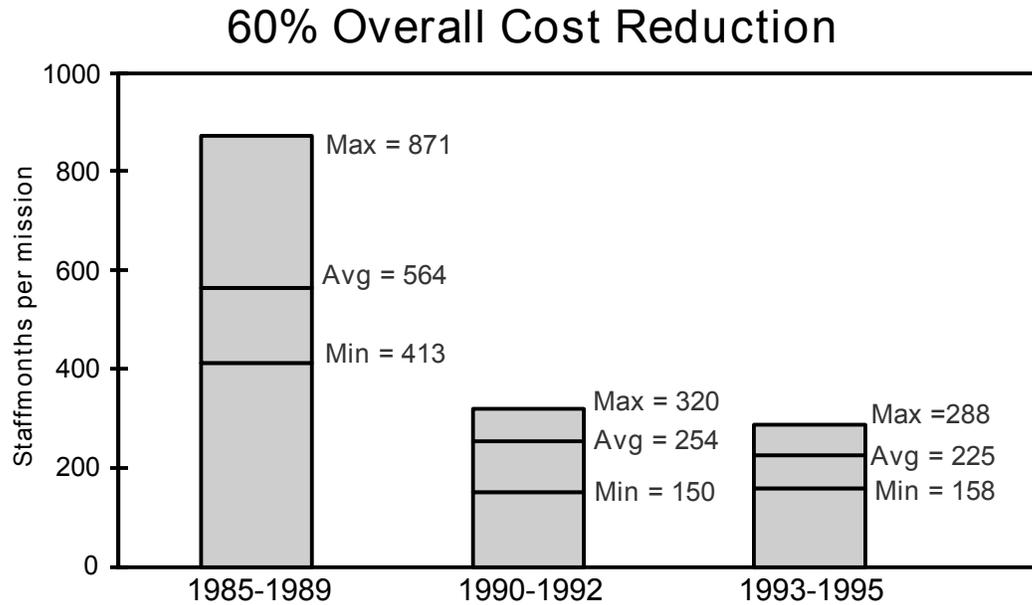


Figure 10. Overall Cost Reduction in SEL

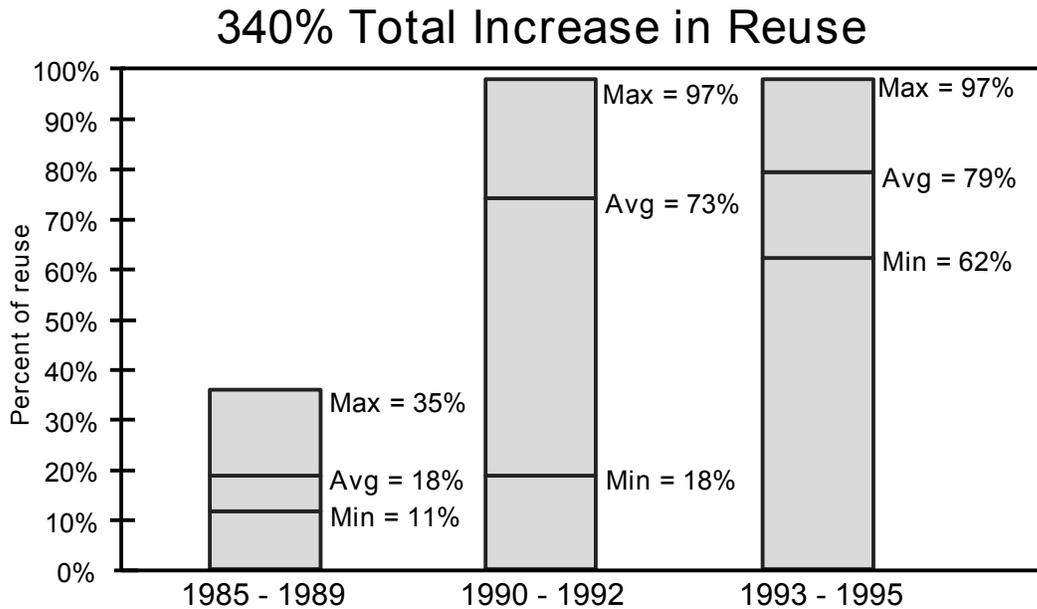


Figure 11. Overall Improvement in Reuse

Observations and Conclusions

The SEL's success with incremental process change, as opposed to leading edge technology adoption, has led us to select the experimental approach to changing process gradually. Experimentation has allowed the beneficial changes to be deployed incrementally with low risk to ongoing projects. Deployment has been quicker for those process changes that were confined to a single phase or development activity, as with the test team process change. Following are several observations and recommendations based on our analysis of the improvement cycles discussed in this paper.

- Focus on a single goal for each process/technology change to provide a clear definition of the expected change and non-ambiguous measurement of its effect. There is a temptation to overload a single project with multiple changes, often in the hope that at least one will work. SEL experience suggests that this approach will not result in sustained improvement; it will only confuse the team and obscure the impact of the individual technologies.
- Select process changes that leverage peoples' talents. Processes that enhance human discipline and intellectual abilities provide significant improvement. Tools should be used to replace or facilitate routine tasks such as configuration and change management.
- Allocate sufficient experimental time for tailoring and iterative application/learning of new concepts. The SEL's experience in first developing OOD concepts followed by a generalized architecture, prior to deployment, shows the benefit of taking a little more time to develop a more usable product (the architecture) rather than deploying the more abstract concepts first.
- Set improvement time expectations appropriately. The more familiar the organization is with the process being changed, the faster it can be tuned and deployed and its impact realized. Existing processes can be refined and adapted more rapidly than abstract concepts; however, the adaption of an external (unfamiliar) process, such as Cleanroom, will take longer than refining an existing local process, such as streamlining the SEL testing process.
- Deploy a subset of the changes as soon as the benefit is shown. Often it is clear that certain subprocesses or techniques are very beneficial even though the entire new process/technology may not yet be proven. Early deployment allows the organization to reap its benefits as early as possible and it also begins to pave the way for the rest of the method that may follow.

References

- [1] Basili, V., "Quantitative Evaluation of a Software Engineering Methodology," Proceedings of the First Pan Pacific Computer Conference, Melbourne, Australia, September 1985.
- [2] Waligora, S., M. Stark, and J. Bailey, "The Impact of Ada and Object-Oriented Design in NASA Goddard's Flight Dynamics Division," Proceedings of the 13th Annual Washington Ada Symposium (WAdaS96), July 1996.
- [3] Basili, V. and R. Selby, Jr., "Comparing the Effectiveness of Software Testing Strategies," IEEE Transactions of Software Engineering, Vol. SE-13, No. 12, December 1987.
- [4] Basili, V. and S. Green, "Software Process Evolution at the SEL," IEEE Software, July 1994, pp. 58-66.
- [5] Waligora, S. and R. Coon, "Improving the Software Testing Process in NASA's Software Engineering Laboratory," Proceedings of the Twentieth Annual Software Engineering Workshop, Goddard Space Flight Center, December 1995.

