SEL'S SOFTWARE PROCESS-IMPROVEMENT PROGRAM

In 1993, the IEEE Computer Society and the Software Engineering Institute jointly established the Software Process Achievement Award to recognize outstanding improvement accomplishments. This award is to be given annually if suitable nominations are received by the SEI before November 1 each year. The nominations are reviewed by an award committee of Barry Boehm, Manny Lehman, Bill Riddle, myself, and Vic Basili (who did not participate in this award decision because of his involvement in the Software Engineering Laboratory).

It is particularly fitting that the SEL was selected as the first winner for this award. They started their pioneering work nearly a decade before the Software Engineering Institute was founded, and their work has been both a guide and an inspiration to all of us who have attempted to follow in their footsteps.

— Watts Humphrey

VICTOR BASILI
and MARVIN ZELKOWITZ
University of Maryland

FRANK McGARRY,
JERRY PAGE,
and SHARON WALIGORA
Computer Sciences Corporation

ROSE PAJERSKI
NASA Goddard Space
Flight Center

For nearly 20 years, the Software Engineering Laboratory has worked to understand, assess, and improve software and the software-development process within the production environment of the Flight Dynamics Division of NASA's Goddard Space Flight Center. We have conducted experiments on about 125 FDD projects, applying, measuring, and analyzing numerous software-process changes. As a result, the SEL has adopted and tailored processes — based on FDD goals and experience — to significantly improve software production.

The SEL is a cooperative effort of NASA/Goddard's FDD, the University of Maryland Department of Computer Science, and Computer Sciences Corporation's Flight Dynamics Technology Group. It was established in 1976 with the goal of reducing

- the defect rate of delivered software,
- the cost of software to support flight projects, and
- the average time to produce mission-support software.

Our work has yielded an extensive set of empirical studies that has guided the evolution of standards, management practices, technologies, and training within the organization. The result has been a 75 percent reduction in defects, a 50 percent reduction in cost, and a 25 percent reduction in cycle time. Over time, the goals of SEL have matured. We now strive to:

- *Understand* baseline processes and product characteristics, such as cost, reliability, software size, reuse levels, and error classes. By characterizing a production environment, we can gain better insight into the software process and its products.
- *Assess* improvements that have been incorporated into development projects. By measuring the impact of available technologies on the software process, we can determine which technologies are beneficial to the environment and — most importantly — how the technologies should be refined to best match the process with the environment.
- *Package and infuse* improvements into the standard SEL process and update and refine standards, handbooks, training materials, and development-support tools.[1-3] By identifying process improvements, we can package the technology so it can be applied in the production environment.

As Figure 1 shows, these goals are pursued in a sequential, iterative process that has been formalized by Basili as the Quality Improvement Paradigm[4] and its use within the SEL formalized as the Experience Factory.[5]

IMPROVING THE PROCESS

We select candidates

0740-7459/94/$04.00 © 1994 IEEE

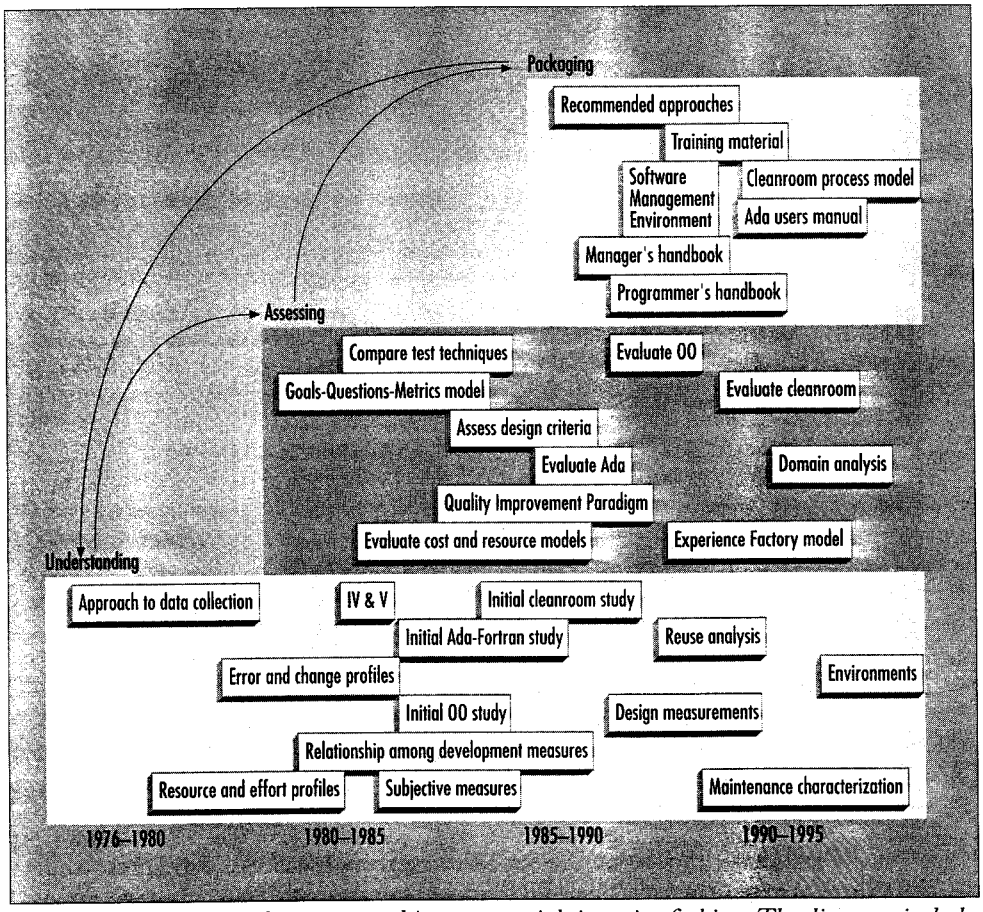

Figure 1. *The SEL goals are pursued in a sequential, iterative fashion. The diagram includes some of the many SEL studies that have been conducted over the years, including those of Cleanroom, Ada, and Fortran.*

for process change on the basis of quantified SEL experiences (such as the most significant causes of errors) and clearly defined goals for the software (such as to decrease error rates). After we select the changes, we provide training and formulate experiment plans. We then apply the new process to one or more production projects and take detailed measurements. We assess a process's success by comparing these measures with the continually evolving baseline. Based upon the results of the analysis, we adopt, discard, or revise the process.

Process improvement applies to individual projects, experiments (the observation of two or three projects), as well as the overall organization (the observation of trends over many years). In the early years, the SEL emphasized building a clear understanding of the process and products within the environment. This led us to develop models, relations, and general characteristics of the SEL environment. Most of our process changes consisted of studying specific, focused techniques (such as program-description language, structure charts, and reading techniques), but the major enhancement was the infusion of measurement, process-improvement concepts, and the realization of the significance of process in the software culture.

SEL OPERATIONS

The SEL has collected and archived data on more than 125 of its software-development projects. We use the data to build typical-project profiles against which we compare and evaluate ongoing projects. The SEL provides its managers with tools for monitoring and assessing project status. The FDD typically runs six to 10 projects simultaneously, each of which is considered an experiment within the SEL.

For each project, we collect a basic set of information (such as effort and error data). From there, the data we collect may vary according to the experiment or be modified as changes are made to specific processes (such as the use of Ada). As the information is collected, it is validated and placed in a central database. We then use this data with other information — such as the subjective lessons learned — to analyze the impact of a specific software process and to measure and feed back results to both ongoing and follow-on projects.

We also use the data to build predictive models and to provide a rationale for refining current software processes. As we analyze the data, we generate papers and reports that reflect the results of numerous studies. We also package the results as standards, policies, training materials, and management tools.

PROCESS AND PRODUCT ANALYSIS

The FDD is responsible for the development and maintenance of flight-dynamics ground-support software for all Goddard flight projects. Typical FDD projects range in size from 100,000 to 300,000 lines of code. Several projects exceed a million lines of code; others are as small as 10,000 lines of code. (At SEL, reused code is not "free"; it is counted as 20 percent of new Fortran code and 30 percent of new Ada code.) The SEL improvement goal is to demonstrate continual improvement of the software process within the FDD environment by carrying out analysis, measurement, and feedback to projects within this environment.

Understanding. Understanding what an organization does and how it operates is fundamental to any

attempt to plan, manage, or improve the software process. This is especially true for software-development organizations. The SEL supports this understanding in several ways, including, for example, the study of effort distribution and error-detection rate.

- Effort distribution identifies which phases of the life cycle consume which portion of development effort. Figure 2 presents the effort distribution of 11 Fortran projects by life-cycle phase and activity. Understanding these distributions helps us plan new efforts, evaluate new technologies, and assess the similarities and differences within an ongoing project.
- Error-detection rate provides the absolute error rate expected in each phase. At SEL, we collected information on software errors and built a model of the expected errors in each life-cycle phase. For 1,000 lines of code, we found about four errors during implementation; two during system test; one during acceptance test; and one-half during operation and maintenance. The trend we derive from this model is that error detection rates fall by 50 percent in each subsequent phase. This pattern seems to be independent of the actual error rates; it is true even in recent projects, in which the overall error rates are declining. We use this model of error rates, as well as other similar types of models, to better predict, manage, and assess change on newly developed projects.

Assessing and refining. We consider each SEL project to be an experiment, in which we study some software method in detail. Generally, the subject of the study is a specific modification to the standard process — a process that obviously comprises numerous software methods.

For example, the Cleanroom software methodology[6] has been applied on four projects within the SEL, three of which have been analyzed thus far. Each project gave us additional insight into the Cleanroom process and helped us refine the method for use in the FDD environment. After training teams in the Cleanroom methodology, we defined a modified set of Cleanroom-specific data to be collected. The teams studied the projects to assess the impact that Cleanroom had on the process, as well as on measures such as productivity and reliability. Figure 3 shows the results of the three analyzed projects.

The Cleanroom experiments required significant changes to the standard SEL development methodology and thus extensive training, preparation, and careful study execution. As in all such experiments, we generated detailed experimentation plans that described the goals, the questions that had to be addressed, and the metrics that had to be collected to answer the questions. Because Cleanroom consists of many specific methods — such as box-structure design, statistical testing, and rigorous inspections — each particular method had to be analyzed, along with the Cleanroom methodology itself. As a result of these projects, a slightly modified Cleanroom approach was deemed beneficial for smaller SEL projects. Anecdotal evidence from the recently completed fourth Cleanroom project confirms the effectiveness of Cleanroom. The revised Cleanroom-process model was captured in a process handbook for future applications to SEL projects. We have analyzed and applied many other methodologies in this way.

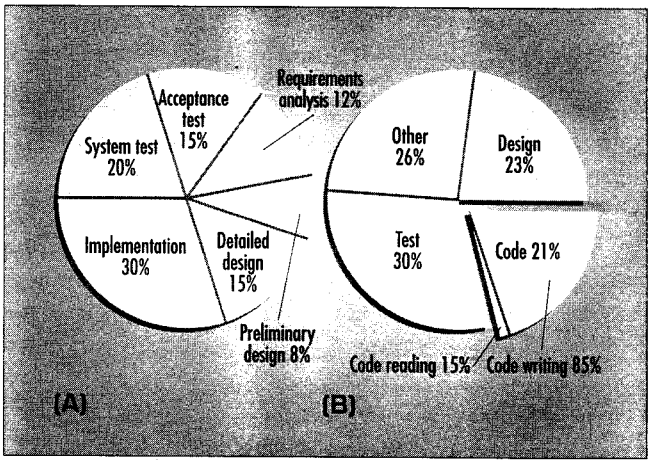

Figure 2. Effort distribution by (A) life-cycle phase and (B) activity. Phase data counts hours charged to a project during each calendar phase. Activity data counts hours attributed to a particular activity (as reported by the programmer), regardless of when in the life cycle the activity occurred.

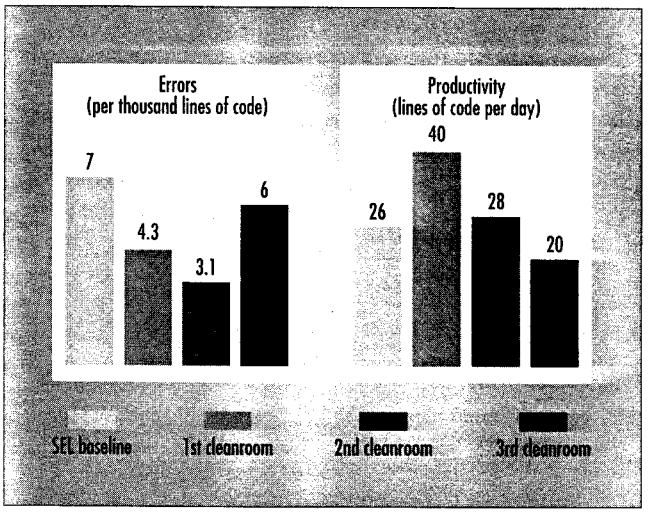

Figure 3. Results of three completed Cleanroom projects, compared against the SEL baseline.

Packaging. Once we have identified beneficial methods and technologies, we provide feedback for future projects by capturing the process in standards, tools, and training. The SEL has produced a set of standards for its own use that reflect the results of its studies. Such standards must continually evolve to capture modified characteristics of the process (the SEL typically updates its basic standard every five years.) Standards we have pro-

TABLE 1
EARLY SEL BASELINE

Project (number & name)	Reuse (percent)	Mission Cost* (staff months)	Reliability (error/KSLOC)
1. GROAGSS	14	381	4.42
2. COBEAGSS	12	348	5.22
3. GOESAGSS	12	261	5.18
4. UARSAGSS	10	675	2.81
5. GROSIM	18	79	8.91
6. COBSIM	11	39	4.45
7. GOESIM	29	96	1.72
8. UARSTELS	35	80	2.96

* *Mission cost = cost of telemetry simulator + cost of AGSS (GRO = projects 1 + 5, COBE = 2 + 6, GOES = 3 + 7, UARS = 4 + 8).*

TABLE 2
CURRENT SEL BASELINE

Project (number & name)	Reuse (percent)	Cost* (staff months)	Reliability (error/KSLOC)
1. EUVEAGSS	18	155	1.22
2. SAMPEX	83	77	.76
3. WINDPOLR	18	476	n/a†
4. EUVETELS	96	36	.41
5. SAMPEXTS	95	21	.48
6. POWITS	69	77	2.39
7. TOMSTELS	97	n/a‡	.23
8. FASTELS	92	n/a‡	.69

* *Mission cost = cost of telemetry simulator + cost of AGSS (GRO = projects 1 + 5, COBE = 2+6, GOES = 3+7, UARS = 4 + 8).*

† *Excluded because it used the Cleanroom development methodology, which counts errors differently.*

‡ *Total mission cost for TOMS and FAST cannot be calculated because AGSSs are incomplete (they are not included in the cost baseline).*

duced include:

- *Manager's Handbook for Software Development,*[1]
- *Recommended Approach to Software Development,*[2] and
- *The SEL Relations and Models.*[3]

In addition to the evolving development standards, policies, and training material, successful packaging includes generating experiment results in the form of post-development analysis, formal papers, and guidebooks for applying specific software techniques.

IMPACT OF SEL

Our studies have involved many technologies, ranging from development and management practices to automation aids and technologies that affect the full life cycle. We have collected and archived detailed information so we can assess the impact of technologies on both the software process and product.

Product impact. To determine the effect of sustained SEL efforts as measured against our major goals, we routinely compare groups of projects developed at different times. Projects are grouped on the basis of size, mission complexity, mission characteristics, language, and platform. On these characteristic projects, we compared defect rates, cost, schedule, and levels of reuse. The reuse levels were studied carefully with the full expectation that there would be a correlation between higher reuse and lower cost and defect rates. These characteristic projects become our "baselines." Table 1 shows an early baseline — eight projects completed between 1985 and 1989. These projects were all ground-based attitude-determination and -simulation systems ranging in size from 50,000 to 150,000 lines of code that were developed on large IBM mainframes. Each was also a success, meeting mission dates and requirements within acceptable cost. Table 2 shows the current SEL baseline, which comprises seven similar projects completed between 1990 and 1994.

As the tables show, the early baseline projects had a reliability rate that ranged from 1.7 to 8.9 errors per 1,000 lines of code, with an average rate of 4.5 errors. The current baseline projects had a reliability rate ranging from 0.2 to 2.4 errors per 1,000 lines of code, with an average rate of 1 error. This is about a 75-percent reduction over the eight-year period.

The dramatic increase in our reuse levels — aided by experimentation with techniques such as object-oriented development and domain-engineering concepts — have been a major contributor to improved project cost and quality. Reuse, along with increased productivity, also contributed to a significant decrease in project cost. We examined selected missions from the two baselines and found that, although the total lines of code per mission remained relatively equal, the total mission cost decreased significantly. The average mission cost in the early baseline ranged from 357 to 755 staff-months, with an average of 490. The current baseline projects had costs ranging from 98 to 277 staff-months with an average of 210. This is a decrease in average cost per mission of more than 50 percent over the eight-year period. This reduction occurred despite the increased mission complexity, shown in Table 3.

Process impact. The most significant changes in the SEL environment are illustrated by the standards, training programs, and development approaches incorporated into the FDD

process. Although specific techniques and methods have had a measurable impact on a class of projects, significant improvement to the software-development process — and an overall change in the environment — has occurred because we have continuously incorporated detailed techniques into higher level organizational processes.

The most significant process attributes that distinguish our current production environment from that of a decade earlier include:

♦ Process change and improvement has been infused as a standard business practice. All standards and training material now contain elements of our continuous-improvement approach to experimentation.

♦ Measurement is now our way of doing business rather than an add-on to development. Measurement is as much a part of our software standards as documentation. It is expected, applied, and effective.

♦ Change is driven by process *and* product. As the process-improvement program matured over the years, our concern for product attributes grew to equal our concern for process attributes. Product goals are always defined before process change is infused. Measures of product are thus as important as those of process (if not more so).

♦ Change is bottom-up. Although process-improvement analysts originally assumed they could work independently of developers, we have realized over the years that change must be guided by development-project experience. Direct input from developers as well as measures extracted from development activities are key factors in change.

♦ "People-oriented" technologies are emphasized, rather than automation. The most effective process changes are those that leverage the thinking of developers. These include reviews, inspections, Cleanroom techniques, management practices, and independent-testing techniques — all of which are driven by disciplined programmers and managers. Automation techniques have sometimes provided improvement, but people-driven approaches have had farther reaching impacts.

The SEL has invested approximately 11 percent of its total software budget into process-improvement. This expense includes project overhead, as well as overhead for data archiving and processing and process and product analysis. We have maintained detailed records so we can accurately record and report process-improvement costs.

Our investment in process-improvement has brought many benefits. The cost, defect rates, and cycle time of flight-dynamics software have decreased significantly since we started the program. Today, our software developers are building better software more efficiently — using many techniques and methods considered experimental only a few years ago. Their progress has been facilitated throughout by the SEL focus on defining organizational goals, expanding domain understanding, and judiciously applying new technology, allowing the FDD to maximize the lessons from local experience. ◆

TABLE 3
COMPARING INCREASE IN BASELINE COMPLEXITY

Attribute	Early SEL baseline	Current SEL baseline
Control	Spin stabilized	Three-axis stabilized
Sensors	1	8 to 11
Torques	1	2 to 3
Onboard computer	Analog simple control	Digital control
Telemetry	5	12 to 15
Data rates	2.2 kbs	32 kbs
Accuracy	1 degree	0.02 degree

REFERENCES

1. L. Landis et al., "Manager's Handbook for Software Devel opment," Revision 1, Tech. Report SEL-84-101, Software Eng. Laboratory, Greenbelt, Md., 1989.
2. L. Landis et al., "Recommended Approach to Software Develop ment," Revision 3, Tech. Report SEL-81-305, Software Eng. Laboratory, Greenbelt, Md. 1992.
3. W. Decker, R. Hendrick, and J. Valett, "Relationships, Models and Measurement Rules," Tech. Report SEL-91-001, Software Eng. Laboratory, Greenbelt, Md., 1991.
4. V.R. Basili and D.M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *IEEE Trans. Software Eng.*, Nov. 1984, pp. 728-738.
5. V.R. Basili, "Software Develop ment: A Paradigm for the Future," *Proc. Compsac*, IEEE CS Press, Los Alamitos, Calif., 1989, pp. 471-485.
6. V.R. Basili and S. Green, "Soft ware Process Evolution at the SEL," *IEEE Software*, July 1994, pp. 58-66.

Victor Basili is a professor in the Institute for Advanced Computer Studies and the Computer Science Department at the University of Maryland. He is co-editor-in-chief of the *International Journal on Empirical Software Engineering.*

Marvin Zelkowitz is a professor in the Institute for Advanced Computer Studies and the Computer Science Department at the University of Maryland and has been involved with the SEL since its inception in 1976. His research interests include language design, environments, and formal methods.

Frank McGarry is a senior member of the executive staff at Computer Sciences Corporation. Previously at NASA, he was a founding director of the SEL in 1976.

Jerry Page is the vice president of the System Science Division at Computer Sciences Corporation. Until last year, he managed SEL activities within CSC.

Sharon Waligora has worked for the Computer Science Corporation since 1974 and directs the CSC branch of the SEL, leading efforts in software process improvement, process definition, and measurement activities.

Rose Pajerski has worked for the Goddard Space Flight Center for more than 20 years and directs the GSFC branch of the SEL. Her research interests include testing processes, systems management through measurement, and tailoring approaches for process-improvement programs.

Address questions about this article to Basili at the Department of Computer Science, University of Maryland, 4121 A.V. Williams, College Park, MD 20742; basili@cs.umd.edu.