Software Process Evolution at the SEL

VICTOR BASILI, *University of Maryland*
SCOTT GREEN, *NASA Goddard Space Flight Center* ·

◆ *The Software Engineering Laboratory has been adapting, analyzing, and evolving software processes for the last 18 years. Their approach is based on the Quality Improvement Paradigm, which is used to evaluate process effects on both product and people. The authors explain this approach as it was applied to reduce defects in code.*

Since 1976, the Software Engineering Laboratory of the National Aeronautics and Space Administration's Goddard Space Flight Center has been engaged in a program of understanding, assessing, and packaging software experience. Topics of study include process, product, resource, and defect models, as well as specific technologies and tools. The approach of the SEL — a consortium of the Software Engineering Branch of NASA Goddard's Flight Dynamics Division, the Computer Science Department of the University of Maryland, and the Software Engineering Operation of Computer Sciences Corp. — has been to gain an in-depth understanding of project and environment characteristics using

process models and baselines. A process is evaluated for study, applied experimentally to a project, analyzed with respect to baselines and process model, and evaluated in terms of the experiment's goals. Then on the basis of the experiment's conclusions, results are packaged and the process is tailored for improvement, applied again, and reevaluated.

In this article, we describe our improvement approach, the Quality Improvement Paradigm, as the SEL applied it to reduce code defects by emphasizing reading techniques. The box on p. 63 describes the Quality Improvement Paradigm in detail. In examining and adapting reading techniques, we go through a systematic process of evaluating the candidate

process and refining its implementation through lessons learned from previous experiments and studies.

As a result of this continuous, evolutionary process, we determined that we could successfully apply key elements of the Cleanroom development method in the SEL environment, especially for projects involving fewer than 50,000 lines of code (all references to lines of code refer to developed, not delivered, lines of code). We saw indications of lower error rates, higher productivity, a more complete and consistent set of code comments, and a redistribution of developer effort. Although we have not seen similar reliability and cost gains for larger efforts, we continue to investigate the Cleanroom method's effect on them.

EVALUATING CANDIDATE PROCESSES

To enhance the possibility of improvement in a particular environment, the SEL introduces and evaluates new technology within that environment. This involves experimentation with the new technology, recording findings in the context of lessons learned, and adjusting the associated processes on the basis of this experience. When the technology is notably risky — substantially different from what is familiar to the environment — or requires more detailed evaluation than would normally be expended, the SEL conducts experimentation off-line from the project environment.

Off-line experiments may take the form of either controlled experiments or case studies. Controlled experiments are warranted when the SEL needs a detailed analysis with statistical assurance in the results. One problem with controlled experiments is that the project must be small enough to replicate the experiment several times. The SEL then performs a case study to validate the results on a project of credible size that is representative of the environment. The case study adds

validity and credibility through the use of typical development systems and professional staff. In analyzing both controlled experiments and case studies, the Goal/Question/Metric paradigm, described in the box on p. 63, provides an important framework for focusing the analysis.

On the basis of experimental results, the SEL packages a set of lessons learned and makes them available in an experience base for future analysis and application of the technology.

Experiment 1: Reading versus testing. Although the SEL had historically been a test-driven organization, we decided to experiment with introducing reading techniques. We were particularly interested in how reading would compare with testing for fault detection. The goals of the first off-line, controlled experiment[1] were to analyze and compare code reading, functional testing, and structural testing, and to evaluate them with respect to fault-detection effectiveness, cost, and classes of faults detected.

We needed an analysis from the viewpoint of quality assurance as well as a comparison of performance with respect to software type and programmer experience. Using the GQM paradigm, we generated specific questions on the basis of these goals.

We had subjects use reading by stepwise abstraction,[2] equivalence-partitioning boundary-value testing, and statement-coverage structural testing.

We conducted the experiment twice at the University of Maryland on graduate students (42 subjects) and once at NASA Goddard (32 subjects). The experiment structure was a fractional factorial design, in which every subject applied each technique on a different program. The programs included a text formatter, a plotter, an abstract data type, and a database, and they ranged from 145 to 365 lines of code. We seeded each program with faults. The reading performed was at the unit level.

Although the results from both experiments support the emphasis on reading techniques, we report only the results of the controlled experiment on the NASA Goddard subjects because it involved professional developers in the target environment.

Figure 1 shows the fault-detection effectiveness and rate for each approach for the NASA Goddard experiment. Reading by stepwise abstraction proved superior to testing
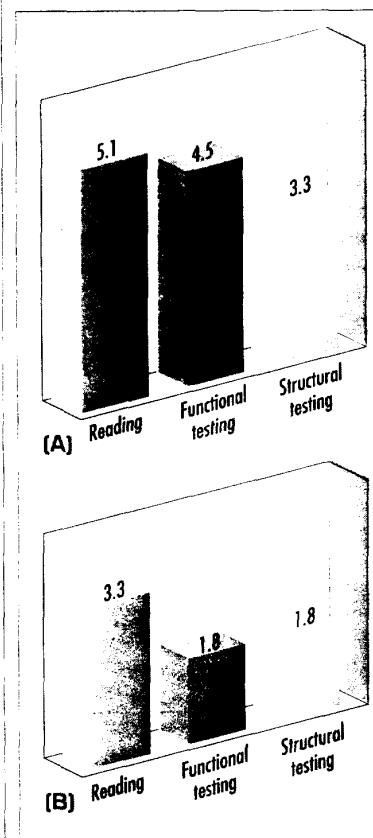
Figure 1. Results of the reading-versus-testing controlled experiment, in which reading was compared with functional and structural testing. (A) Mean number of faults detected for each technique and (B) number of faults detected per hour of use for each technique.

techniques in both the effectiveness and cost of fault detection, while obviously using fewer computer resources.

Even more interesting was that the subjects did a better job of estimating the code quality using reading than they did using testing. Readers thought they had found only about half the faults (which was nominally correct), while functional testers felt that had found essentially all the faults (which was never correct).

Furthermore, after completing the experiment, more than 90 percent of the participants thought functional testing had been the most effective technique, although the results clearly showed otherwise. This gave us some insight into the psychological effects of reading versus testing. Perhaps one reason testing appeared more satisfying was that the successful execution of multiple test cases generated a greater comfort level with the product quality, actually providing the tester with a false sense of confidence.

Reading was also more effective in uncovering most classes of faults, including interface faults. This told us

that perhaps reading might scale up well on larger projects.

Experiment 2: Validation with Cleanroom. On the basis of these results, we decided to emphasize reading techniques in the SEL environment. However, we saw little improvement in overall reliability of the development systems. Part of the reason may have been that SEL project personnel had developed such faith in testing that the quality of their reading was relaxed, with the assumption that testing would ultimately uncover the same faults. We conducted a small off-line experiment at the University of Maryland to test this hypothesis; the results supported our assumption. (We did this on a small scale just to verify our hypothesis before continuing with the Cleanroom experiment.)

Why the Cleanroom method? The Cleanroom method emphasizes human discipline in the development process, using a mathematically based design approach and a statistical testing approach based on anticipated opera-

tional use.[3] Development and testing teams are independent, and all development-team activities are performed without on-line testing.

Techniques associated with the method are the use of box structures and state machines, reading by stepwise abstraction, formal correctness demonstrations, and peer review. System development is performed through a pipeline of small increments to enhance concentration and permit testing and development to occur in parallel.

Because the Cleanroom method removes developer testing and relies on human discipline, we felt it would overcome the psychological barrier of reliance on testing.

Applying the QIP. The first step of the Quality Improvement Paradigm is to characterize the project and its environment. The removal of developer unit testing made the Cleanroom method a high-risk technology. Again, we used off-line experimentation at the University of Maryland as a mitigating approach.[4] The environment was a laboratory course at the university, and the project involved an electronic message system of about 1,500 LOC. The experiment structure was a simple replicated design, in which control and experiment teams are defined. We assigned 10 three-person experiment teams to use the Cleanroom method. We gave five three-person control teams the same development methodology, but allowed them to test their systems. Each team was allowed five independent test submissions of their programs. We collected data on programmer background and attitude, computer-resource activity, and actual testing results.

The second step in the Quality Improvement Paradigm is to set goals. The goal here was to analyze the effects of the Cleanroom approach and evaluate it with respect to process, product, and participants, as compared with the non-Cleanroom approach.

	Sample measures	Sample baseline	Sample expectation
Process	Effort distribution Change profile	Other 26% Design 23% Test 30% Code 21%	Increased design effort because of emphasis on peer-review process
Cost	Productivity Level of rework Impact of specification changes	Historically, 26 lines of code per day	No degradation from current level
Reliability	Error rate Error distribution Error source	Historically, seven errors per thousand lines of code	Decreased error rate

Figure 2. Sample measures, baselines, and expectations for the case studies investigating the Cleanroom method.

We generated questions corresponding to this goal, focusing on the method's effect on each aspect being studied.

The next step of the Quality Improvement Paradigm involves selecting an appropriate process model. The process model selected for this experiment was the Cleanroom approach as defined by Harlan Mills at IBM's Federal Systems Division, but modified for our environment. For example, the graduate-student assistant for the course served as each group's independent test team. Also, because we used a language unfamiliar to the subjects to prevent bias, there was a risk of errors due solely to ignorance about the language. We therefore allowed teams to cleanly compile their code before submitting it to the tester.

Because of the nature of controlled experimentation, we made few modifications during the experiment.

Cleanroom's effect on the software-development process resulted in the Cleanroom developers more effectively applying the off-line reading techniques; the non-Cleanroom teams focused their efforts more on functional testing than reading. The Cleanroom teams spent less time on-line and were more successful in making scheduled deliveries. Further analysis revealed that the Cleanroom products had less dense complexity, a higher percentage of assignment statements, more global data, and more code comments. These products also more completely met the system requirements and had a higher percentage of successful independent test cases.

The Cleanroom developers indicated that they modified their normal software-development activities by doing a more effective job of reading, though they missed the satisfaction of actual program execution. Almost all said they would be willing to use Cleanroom on another development assignment.

Through observation, it was also clear that the Cleanroom developers did not apply the formal methods associated with Cleanroom very rigorously. Furthermore, we did not have enough failure data or experience with Cleanroom testing to apply a reliability model. However, general analysis did indicate that the Cleanroom approach had potential payoff, and that additional investigation was warranted.

You can also view this experiment from the following perspective: We applied two development approaches. The only real difference between them was that the control teams had one extra piece of technology (developer testing), yet they did not perform as well as the experiment teams. One explanation might be that the control group did not use the available nontesting techniques as effectively because they knew they could rely on testing to detect faults. This supports our earlier findings associated with the reading-versus-testing experiment.

EVOLVING SELECTED PROCESS

The positive results gathered from these two experiments gave us the justification we needed to explore the Cleanroom method in case studies, using typical development systems as data points. We conducted two case studies to examine the method, again following the steps of the Quality Improvement Paradigm. A third case study was also recently begun.

First case study. The project we selected, Project 1, involved two subsystems from a typical attitude ground-support system. The system performs ground processing to determine a spacecraft's attitude, receiving and processing spacecraft telemetry data to meet the requirements of a particular mission.

ALMOST ALL THE CLEANROOM TEAM SAID THEY'D USE THE METHOD AGAIN.

The subsystems we chose are an integral part of attitude determination and are highly algorithmic. Both are interactive programs that together contain approximately 40,000 LOC, representing about 12 percent of the entire attitude ground-support system. The rest of the ground-support system was developed using the standard SEL development methodology.

The project was staffed principally by five people from the Flight Dynamics Division, which houses the SEL. All five were also working on other projects, so only part of their time was allocated to the two subsystems. Their other responsibilities often took time and attention away from the case study, but this partial allocation represents typical staffing in this environment. All other projects with which the Project 1 staff were involved were non-Cleanroom efforts, so staff members would often be required to use multiple develop-ment methodologies during the same workday.

The primary goal of the first case study was to increase software quality and reliability without increasing cost. We also wanted to compare the characteristics of the Cleanroom method with those typical of the FDD environment. A well-calibrated baseline was available for comparison that described a variety of process characteristics, including effort distribution, change rates, error rates, and productivity. The baseline represents the history of many earlier SEL studies. Figure 2 shows a sample of the expected variations from the SEL baselines for a set of process characteristics.

Choosing and tailoring processes. The process models available for examination were the standard SEL model,[5] which represents a reuse-oriented waterfall life-cycle model; the
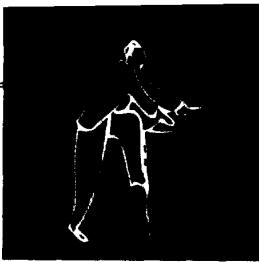
IBM/FSD Cleanroom model, which appeared in the literature and was available through training; and the experimental University of Maryland Cleanroom model, which was used in the earlier controlled experiment.[4]

We examined the lessons learned from applying the IBM and University of Maryland models. The results from the IBM model were notably positive, showing that the basic process, methods, and techniques were effective for that particular environment. However, the process model had been applied by the actual developers of the methodology, in the environment for which it was developed. The University of Maryland model also had specific lessons, including the effects of not allowing developers to test their code, the effectiveness of the process on a small project, and the conclusion that formal methods appeared particularly difficult to apply and required specific skills.

On the basis of these lessons and the characteristics of our environment, we selected a Cleanroom process model with four key elements:

♦ separation of development and test teams,

♦ reliance on peer review instead of unit-level testing as the primary developer verification technique,

♦ use of informal state machines and functions to define the system design, and

♦ a statistical approach to testing based on operational scenarios.

We also provided training for the subjects, consistent with a University of Maryland course on the Cleanroom process model, methods, and techniques, with emphasis on reading through stepwise abstraction. We also stressed code reading by multiple reviewers because stepwise abstraction was new to many subjects. Michael Dyer and Terry Baker of IBM/FSD

provided additional training and motivation by describing IBM's use of Cleanroom.

To mitigate risk and address the developers' concerns, we examined backout options for the experiment. For example, because the subsystems were highly mathematical, we were afraid it would be difficult to find and correct mathematical errors without any developer testing. Because the project was part of an operational system with mission deadlines, we discussed options that ranged from allowing developer unit testing to discontinuing Cleanroom altogether. These discussions helped allay the primary apprehension of NASA Goddard management in using the new methodology. When we could not get information about process application, we followed standard SEL process-model activities.

We also noted other management and project-team concerns. Requirements and specifications change frequently during the development cycle in the FDD environment. This instability was of particular concern because the Cleanroom method is built on the precept of developing software right the first time. Another concern was that, given the difficulties encountered in the University of Maryland experiment about applying formal methods, how successfully could a classical Cleanroom approach be applied? Finally, there was concern about the psychological effects of separating development and testing, specifically the inability of the developers to execute their code. We targeted all these concerns for our postproject analysis.

Project 1 lasted from January 1988 through September 1990. We separated the five team members into a three-person development team and a two-person test team. The development

PROJECT RESULTS LED US TO EMPHASIZE PEER REVIEWS AND USE OF INDEPENDENT TESTING.

team broke the total effort into six incremental builds of approximately 6,500 LOC each. An experimenter team consisting of NASA Goddard managers, SEL representatives, a technology advocate familiar with the IBM model, and the project leader monitored the overall process.

We modified the process in real time, as needed. For example, when we merged Cleanroom products into the standard FDD formal review and documentation activities, we had to modify both. We altered the design process to combine the use of state machines and traditional structured design. We also collected data for the monitoring team at various points throughout the project, although we tried to do this with as little disturbance as possible to the project team.

Analyzing and packaging results. The final steps in the QIP involve analyzing and packaging the process results. We found significant differences in effort distribution during development between the Cleanroom project and the baseline. Approximately six percent of the total project effort shifted from coding to design activities in the Cleanroom effort. Also, the baseline development teams traditionally spent approximately 85 percent of their coding effort writing code, 15 percent reading it. The Cleanroom team spent about 50 percent in each activity.

The primary goal of the first case study had been to improve reliability without increasing cost. Analysis showed a reduction in change rate of nearly 50 percent and a reduction in error rate of greater than a third. Although the expectation was for productivity equivalent to the baseline, the Cleanroom effort also improved in that area by approximately 50 percent. We also saw a decrease in rework, as defined by the amount of time spent correcting errors. Additional analysis of code reading revealed that three fourths of all errors uncovered were found by only one reader. This prompted a renewed emphasis on mul-

QUALITY IMPROVEMENT PARADIGM: FOUNDATION FOR IMPROVEMENT

The Quality Improvement Paradigm is an effective framework for conducting experiments and studies like those described in the main text. It is an experimental but evolutionary concept for learning and improvement.[1]

The QIP has six steps:

1. Characterize the project and its environment.

2. Set quantifiable goals for successful project performance and improvement.

3. Choose the appropriate process models, supporting methods, and tools for the project.

4. Execute the processes, construct the products, collect and validate the prescribed data, and analyze the data to provide real-time feedback for corrective action.

5. Analyze the data to evaluate current practices, determine problems, record findings, and make recommendations for future process improvements.

6. Package the experience in the form of updated and refined models, and save the knowledge gained from this and earlier projects in an experience base for future projects.

The QIP uses two tools: the Goal/Question/Metric paradigm and the Experience Factory Organization.

GQM paradigm. The GQM paradigm is a mechanism used in the planning phase of the Quality Improvement Paradigm for defining and evaluating a set of operational goals using measurement.[2] It provides a systematic approach for tailoring and integrating goals with models of the software processes, products, and quality perspectives of interest, according to the specific needs of the project and organization.

You define goals in an operational, tractable way by refining them into a set of questions that extract appropriate information from the models. The questions, in turn, define the metrics needed to define and interpret the goals.

A goal-generation template helps in developing goals. The template specifies the essential elements: the object of interest (like product or process), the aspect of interest (like cost or ability

to detect defects), the purpose of the study (like assessment or prediction), the point of view from which the study is performed (like customer's or manager's), and the context in which the study is performed (like people-oriented or problem-oriented factors).

For example, two goals associated with the application of the Cleanroom method in the SEL were analysis of the Cleanroom process to characterize resource allocation from the project manager's point of view, and analysis of the Cleanroom product to characterize defects from the customer's point of view.

Experience Factory Organization. The Experience Factory Organization is an organizational structure that supports the activities specified in the QIP by continuously accumulating evaluated experiences, building a repository of integrated experience models that projects can access and modify to meet their needs.[3] The Experience Factory extends project-development activities by providing systematic

learning and packaging of reusable experiences. It packages experiences by building informal, schematized, formal, and automated models and measures of software processes, products, and other forms of knowledge, and distributes them through consultation, documentation, and automated support.

While project organization follows an evolutionary process model that reuses packaged experiences, the Experience Factory provides the set of processes needed for learning, packaging, and storing the project organization's experience for reuse. The Experience Factory Organization represents the integration of these two functions.

REFERENCES

1. V. Basili, "Quantitative Evaluation of Software Engineering Methodology," Tech. Report TR-1519, CS Dept., Univ. of Maryland, College Park, July 1985.
2. V. Basili and H. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Trans. Software Eng.*, June 1988, pp. 758-773.
3. V. Basili, "Software Development: A Paradigm for the Future," *Proc. Compsac*, IEEE CS Press, Los Alamitos, Calif., 1989.

---

tiple readers throughout the SEL environment.

We also examined the earlier concerns expressed by managers and the project team. The results showed increased effort in early requirements-analysis and design activities and a clearer set of in-line comments. This led to a better understanding of the whole system and enabled the project team to understand and accommodate changes with greater ease than was typical for that environment.

We reviewed the application of classical Cleanroom and noted successes and difficulties. The structure of independent teams and the emphasis on peer review during development was easy to apply. However, the devel-

opment team did have difficulty using the associated formal methods. Also, unlike the scheme in the classical Cleanroom method, the test team followed an approach that combined statistical testing with traditional functional testing.

Finally, the psychological effects of independent testing appeared to be negligible. All team members indicated high job satisfaction as well as a willingness to apply the method in future projects.

We packaged these early results in various reports and presentations, including some at the SEL's 1990 Software Engineering Workshop. As a reference for future SEL Cleanroom projects, we also began efforts to pro-

duce a document describing the SEL Cleanroom process model, including details on specific activities.[6] (The completed document is now available to current Cleanroom projects.)

Second case study. The first case study showed us that we needed better training in the use of formal methods and more guidance in applying the testing approach. We also realized that experiences from the initial project team had to be disseminated and used.

Again, we followed the Quality Improvement Paradigm. We selected two projects: one similar to the initial Cleanroom project, Project 2A, and one more representative of the typical FDD contractor-support environment,

(A)

(B)
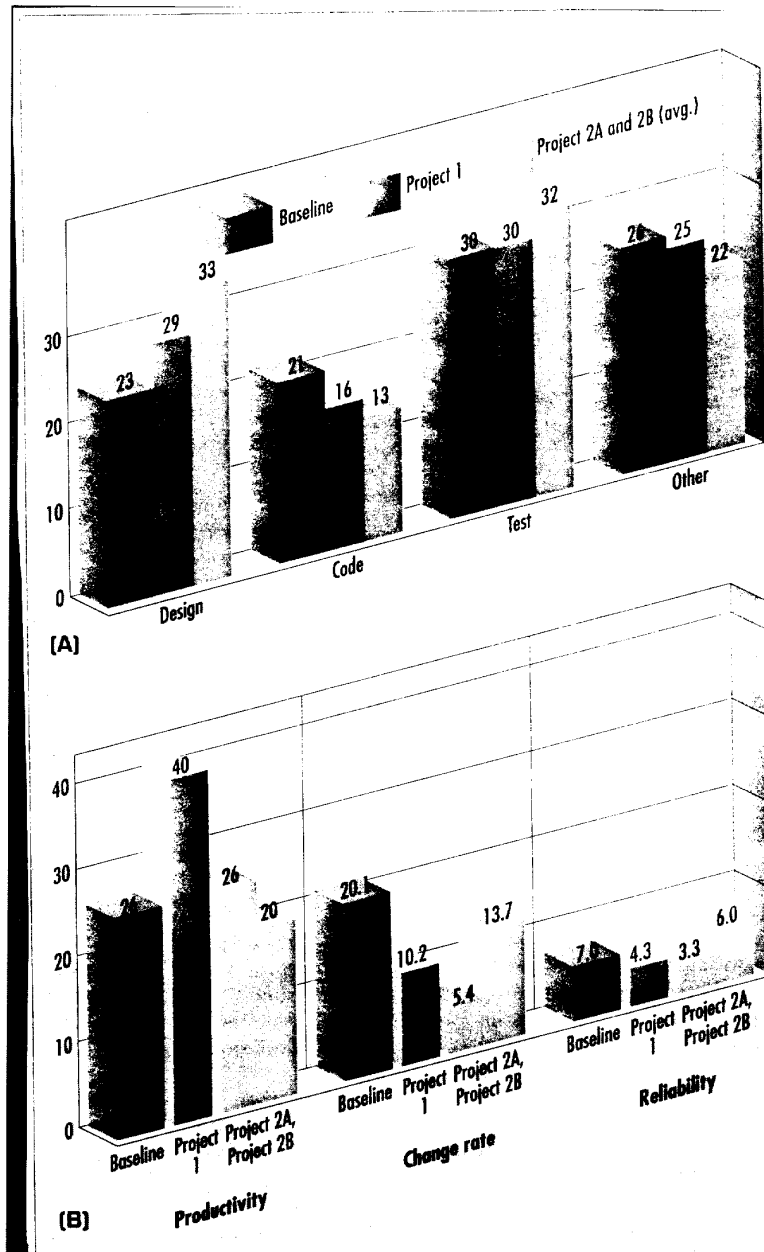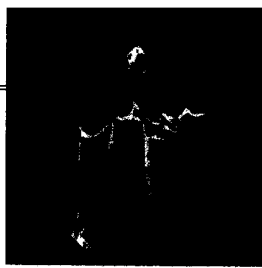
Figure 3. Measurement comparisons for two case studies investigating Cleanroom. The first case study involved one project, Project 1. The second case study involved two projects, Projects 2A and 2B. (A) Percentage of total development effort for various development activities, and (B) productivity in lines of code per day, change rate in changes per thousand lines of code, and reliability in errors per thousand lines of code.

Project 2B.

Project 2A involved a different subsystem of another attitude ground-support system. This subsystem focused on the processing of telemetry data, comprising 22,000 LOC. The project was staffed with four developers and two testers. Project 2B involved an entire mission attitude ground-support system, consisting of approximately 160,000 LOC. At its peak, it was staffed with 14 developers and four testers.

Setting goals and choosing processes. The second case study had two goals. One was to verify measures from the first study by applying the Cleanroom method to Project 2A, a project of similar size and scope. The second was to verify the applicability of Cleanroom on Project 2B, a substantially larger project but one more representative of the typical environment. We also wanted to further tailor the process model to the environment by using results from the first case study and applying more formal techniques.

Packages from the SEL Experience Factory (described in the box on p. 63) were available to support project development. These included an evolved training program, a more knowledgeable experimenter team to monitor the projects, and several in-process interactive sessions with the project teams. Although we had begun producing a handbook detailing the SEL Cleanroom process model, it was not ready in time to give to the teams at the start of these projects.

The project leader for the initial Cleanroom project participated as a member of the experimenter team, served as the process modeler for the handbook, and acted as a consultant to the current projects.

We modified the process according to the experiences of the Cleanroom team in the first study. Project 1's team had had difficulty using state machines in system design, so we changed the emphasis to Mills' box-structure algorithm.[7] We also added a more extensive

TABLE 1
PROJECT COMPARISONS FOR SEL TECHNOLOGY EVALUATION

Evaluation aspect	Controlled experiments		Cleanroom case studies		
	Reading vs.testing	Cleanroom	Project 1	Project 2A	Project 2B
Team size	32 participants	Three-person development teams (10 experiment teams; five control teams); common independent tester	Three-person development team; two-person test team	Four-person development team; two-person test team	Fourteen-person development team; four-person test team
Project size and application	Small (145-365 LOC) sample Fortran programs	1,500 LOC, Fortran, electronic message system for graduate laboratory course	40,000 LOC, Fortran, flight-dynamics ground-support system	22,000 LOC, Fortran, flight-dynamics ground- support system	160,000 LOC, Fortran, flight-dynamics ground-support system
Results	Reading techniques appear more effective than testing techniques for fault detection	Cleanroom teams use fewer computer resources, satisfy requirements more successfully, and make higher percentage of scheduled deliveries	Project spends higher percentage of effort in design, uses fewer computer resources, and achieves better productivity and reliability than environment baseline	Project continues trend in better reliability while maintaining baseline productivity	Project reliability only slightly better than baseline while productivity falls below baseline

training program focusing on Cleanroom techniques, experiences from the initial Cleanroom team, and the relationship between the Cleanroom studies and the SEL's general goals. The instruction team included representatives from the SEL, members of the initial team, and Mills. Mills gave talks on various aspects of the methodology, as well as motivational remarks on the potential benefits of the Cleanroom method in the software community.

Project 2A ran from March 1990 through January 1992. Project 2B ran from February 1990 through December 1992. Again, we examined reliability, productivity, and process characteristics, comparing them to Project 1 results and the SEL baseline.

Analyzing and packaging results. As Figure 3 shows, there were significant differences between the two projects. Error and change rates for Project 2A continued to be favorable. Productivity rate, however, returned to the SEL baseline value. Error and change rates for Project 2B increased from Project 1 values, although they remained lower than SEL baseline numbers. Productivity, however, dropped below the baseline.

When we examined the effort distribution among the baseline and Projects 1, 2A, and 2B, we found a continuing upward trend in the percentage of design effort, and a corresponding decrease in coding effort. Additional analysis indicated that although the overall error rates were below the baseline, the percentage of system components found to contain errors during testing was still representative of baseline projects developed in this environment. This suggests that the breadth of error distribution did not change with the Cleanroom method.

In addition to evaluating objective data for these two projects, we gathered subjective input through written and verbal feedback from project participants. In general, input from Project 2A team members, the smaller of the two projects, was very favorable, while Project 2B members, the larger contractor team, had significant reservations about the method's application. Interestingly, though, specific shortcomings were remarkably similar for both teams. Four areas were generally cited in the comments. Participants were dissatisfied with the use of design abstractions and box structures, did not fully accept the rationale for having no developer compilation, had problems coordinating information between developers and testers, and cited the need for a reference to the SEL Cleanroom process model.

Again, we packaged these results into various reports and presentations, which formed the basis for additional process tailoring.

Third case study. We have recently begun a third case study to examine difficulties in scaling up the Cleanroom method in the typical contractor-support environment and to verify previous trends and analyze additional tailoring of the SEL process model. We expect the study to complete in September.

In keeping with this goal, we again selected a project representative of the FDD contractor-support environment, but one that was estimated at 110,000 LOC, somewhat smaller than Project 2B. The project involves development of another entire mission attitude ground-support system. Several team members have prior experience with the Cleanroom method through previous SEL studies.

Experience Factory packages available to this project include training in the Cleanroom method, an experienced experimenter team, and the *SEL Cleanroom Process Model* (the completed handbook). In addition to modifying the process model according to the results from the first two case studies, we are

providing regularly scheduled sessions in which the team members and experimenters can interact. These sessions give team members the opportunity to communicate problems they are having in applying the method, ask for clarification, and get feedback on their activities. This activity is aimed at closing a communication gap that the contractor team felt existed in Project 2B.

The concepts associated with the QIP and its use of measurement have given us an evolutionary framework for understanding, assessing, and packaging the SEL's experiences.

Table 1 shows how the evolution of our Cleanroom study progressed as we used measurements from each experiment and case study to define the next experiment or study. The SEL Cleanroom process model has evolved on the basis of results packaged through earlier evaluations. Some aspects of the target methodology continue to evolve: Experimentation with formal methods has transitioned from functional decomposition and state machines to box-structure design and again to box-structure application as a way to abstract requirements. Testing has shifted from a combined statistical/functional approach, to a purely statistical approach based on operational scenarios. Our current case study is examining the effect of allowing developer compilation.

Along the way, we have eliminated some aspects of the candidate process; we have not examined reliability models, for example, since the environment does not currently have sufficient data to seed them. We have also emphasized some aspects. For example, we are conducting studies that focus on the effect of peer reviews and independent test teams for non-Cleanroom projects. We are also studying how to improve reading by developing reading techniques through off-line experimentation.

The SEL baseline used for comparison is undergoing continual evolution. Promising techniques are filtered into the development organization as general

process improvements, and corresponding measures of the modified process (effort distribution, reliability, cost) indicate the effect on the baseline.

The SEL Cleanroom process model has evolved to a point where it appears applicable to smaller projects (fewer than 50,000 LOC), but additional understanding and tailoring is still required for larger scale efforts. The model will continue to evolve as we gain more data from development projects. Measurement will provide baselines for comparison, identify areas of concern and improvement, and provide insight into the effects of

process modifications. In this way, we can set quantitative expectations and evaluate the degree to which goals have been achieved.

By adhering to the Quality Improvement Paradigm, we can refine the process model from study to study, assessing strengths and weaknesses, experiences, and goals. However, our investigation into the Cleanroom method illustrates that the evolutionary infusion of technology is not trivial and that process improvement depends on a structured approach of understanding, assessment, and packaging. ◆

REFERENCES
1. V. Basili and R. Selby, "Comparing the Effectiveness of Software Testing Strategies," *IEEE Trans. Software Eng.*, Dec. 1987, pp. 1278-1296.
2. R. Linger, H. Mills, and B. Witt, *Structured Programming: Theory and Practice*, Addison-Wesley, Reading, Mass., 1979.
3. H. Mills, M. Dyer, and R. Linger, "Cleanroom Software Engineering," *IEEE Software*, Sept. 1987, pp. 19-24.
4. R. Selby, Jr., V. Basili, and T. Baker, "Cleanroom Software Development: An Empirical Evaluation," *IEEE Trans. Software Eng.*, Sept. 1987, pp. 1027-1037.
5. L. Landis et al., "Recommended Approach to Software Development: Revision 3," Tech. Report SEL-81-305, Software Engineering Laboratory, Greenbelt, Md., 1992.
6. S. Green, *Software Engineering Laboratory (SEL) Cleanroom Process Model*, Tech. Report SEL-91-004, Software Engineering Laboratory, Greenbelt, Md., 1991.
7. H. Mills, "Stepwise Refinement and Verification in Box-Structured Systems," *IEEE Software*, June 1988, pp. 23-36.
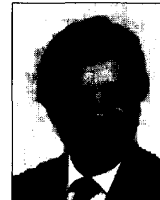
Victor Basili is a professor of computer science at the Institute for Advanced Computer Studies at the University of Maryland at College Park. One of the founders and principals of the Software Engineering Laboratory, his interests include quantitative approaches for software management, engineering, and quality assurance. He is on the editorial board of *Journal of Systems and Software*.

Basili received a BS in mathematics from Fordham College, an MS in mathematics from Syracuse University, and a PhD in computer science from the University of Texas at Austin. He is an IEEE fellow and a member of the IEEE Computer Society.

Scott Green is a senior software engineer in NASA Goddard's Flight Dynamics Division, where he is involved in the project management of ground-support systems and in leading software-engineering studies at the Software Engineering Laboratory.

Green received a BS in computer science from Loyola College.

Address questions about this article to Basili at CS Dept., University of Maryland, College Park, MD 20742; basili@cs.umd.edu; or to Green at NASA/GFSC, Code 552.1, Greenbelt, MD 20771; segreen@gsfcmail.nasa.gov.