THE SOFTWARE ENGINEERING LABORATORY—AN OPERATIONAL SOFTWARE EXPERIENCE FACTORY

Victor Basili and Gianluigi Caldiera
University of Maryland

Frank McGarry and Rose Pajerski
National Aeronautics and Space Administration/
Goddard Space Flight Center

Gerald Page and Sharon Waligora
Computer Sciences Corporation

ABSTRACT

For 15 years, the Software Engineering Laboratory (SEL) has been carrying out studies and experiments for the purpose of understanding, assessing, and improving software and software processes within a production software development environment at the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC). The SEL comprises three major organizations:

- NASA/GSFC, Flight Dynamics Division
- University of Maryland, Department of Computer Science
- Computer Sciences Corporation, Flight Dynamics Technology Group

These organizations have jointly carried out several hundred software studies, producing hundreds of reports, papers, and documents, all of which describe some aspect of the software engineering technology that has been analyzed in the flight dynamics environment at NASA. The studies range from small, controlled experiments (such as analyzing the effectiveness of code reading versus that of functional testing) to large, multiple-project studies (such as assessing the impacts of Ada on a production environment). The organization's driving goal is to improve the software process continually, so that sustained improvement may be observed in the resulting products. This paper discusses the SEL as a functioning example of an operational software experience factory and summarizes the characteristics of and major lessons learned from 15 years of SEL operations.

1. THE EXPERIENCE FACTORY CONCEPT

Software engineering has produced a fair amount of research and technology transfer in the first 24 years of its existence. People have built technologies, methods, and tools that are used by many organizations in development and maintenance of software systems.

Unlike other disciplines, however, very little research has been done in the development of models for the various components of the discipline. Models have been developed primarily for the software product, providing mathematical models of its function and structure (e.g., finite state machines in object-oriented design), or, in some advanced instances, of its observable quality (e.g., reliability models). However, there has been very little modeling of several other important components of the software engineering discipline, such as processes, resources, and defects. Nor has much been done toward understanding the logical and physical integration of software engineering models, analyzing and evaluating them via experimentation and simulation, and refining and tailoring them to the characteristics and the needs of a specific application environment.

Currently, research and technology transfer in software engineering are done mostly bottom-up and in isolation. To provide software engineering with a rigorous, scientific foundation and a pragmatic framework, the following are needed [1]:

- A top-down, experimental, evolutionary framework in which research can be focused and logically integrated to produce models of the discipline, which can then be evaluated and tailored to the application environment
- An experimental laboratory associated with the software artifact that is being produced and studied to develop and refine comprehensive models based upon measurement and evaluation

The three major concepts supporting this vision are

- A concept of evolution: the Quality Improvement Paradigm [2]
- A concept of measurement and control: the Goal/Question/Metric Approach [3]
- A concept of the organization: the Experience Factory [4]

The **Quality Improvement Paradigm** is a two-feedback loop process (project and organization loops) that is a variation of the scientific method. It consists of the following steps:

- Characterization: Understand the environment based upon available models, data, intuition, etc., so that similarities to other projects can be recognized
- Planning: Based on this characterization:
  - Set quantifiable goals for successful project and organization performance and improvement
  - Choose the appropriate processes for improvement, and supporting methods and tools to achieve the goals in the given environment
- Execution: Perform the processes while constructing the products and provide real-time project feedback based on the goal achievement data
- Packaging: At the end of each specific project:
  - Analyze the data and the information gathered to evaluate the current practices, determine problems,

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

©1992 ACM 0-89791-504-6/ 92/ 0500- 0370 1.50

record findings, and make recommendations for future project improvements

- Package the experience gained in the form of updated and refined models and other forms of structured knowledge gained from this and prior projects

- Store the packages in an experience base so they are available for future projects

The **Goal/Question/Metric Approach** is used to define measurement on the software project, process, and product in such a way that

- Resulting metrics are tailored to the organization and its goals
- Resulting measurement data play a constructive and instructive role in the organization
- Metrics and their interpretation reflect the quality values and the different viewpoints (developers, users, operators, etc.)

Although originally used to define and evaluate a particular project in a particular environment, the Goal/Question/Metric Approach can be used for control and improvement of a software project in the context of several projects within the organization [5,6].

The Goal/Question/Metric Approach defines a measurement model on three levels:

- Conceptual level (goal): A goal is defined for an object, for a variety of reasons, with respect to various models of quality, from various points of view, and relative to a particular environment
- Operational level (question): A set of questions is used to define models of the object of study and the focuses on that object to characterize the assessment or achievement of a specific goal
- Quantitative level (metric): A set of metrics, based on the models, is associated with every question in order to answer it in a quantitative way

The concept of the **Experience Factory** was introduced to institutionalize the collective learning of the organization that is at the root of continual improvement and competitive advantage.

Reuse of experience and collective learning cannot be left to the imagination of individual, very talented, managers: they become a corporate concern, like the portfolio of a business or company assets. The experience factory is the organization that supports reuse of experience and collective learning by developing, updating, and delivering, upon request to the project organizations, clusters of competencies that the SEL refers to as experience packages. The project organizations offer to the experience factory their products, the plans used in their development, and the data gathered during development and operation (Figure 1). The experience factory transforms these objects into reusable units and supplies them to the project organizations, together with specific support that includes monitoring and consulting (Figure 2).

The experience factory can be a logical and/or physical organization, but it is important that its activities are separated and made independent from those of the project organization. The packaging of

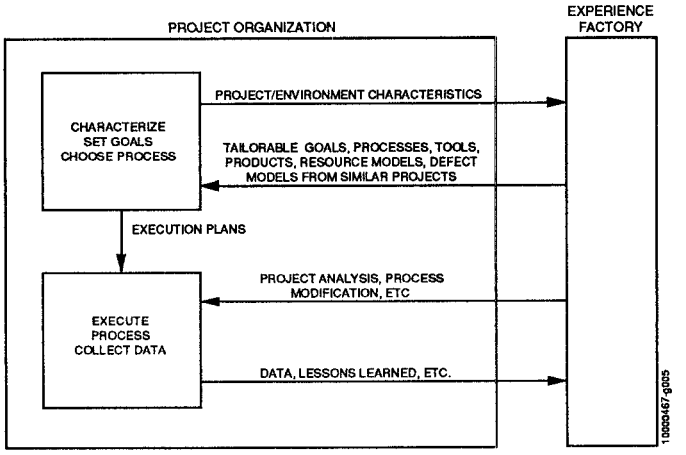


Figure 1. Project Organization Functions

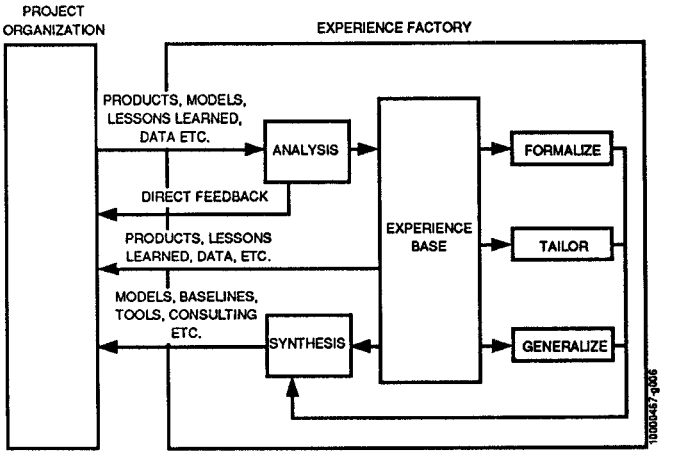


Figure 2. Experience Factory Functions

experience is based on tenets and techniques that are different from the problem solving activity used in project development [7].

On the one hand, from the perspective of an organization producing software, the difference is outlined in the following chart:

PROJECT ORGANIZATION (Problem Solving)	EXPERIENCE FACTORY (Experience Packaging)
Decomposition of a problem into simpler ones	Unification of different solutions and redefinition of the problem
Instantiation	Generalization, formalization
Design/implementation process	Analysis/synthesis process
Validation and verification	Experimentation

On the other hand, from the perspective of software engineering research, there are the following goals:

PROJECT ORGANIZATION (Problem Solving)	EXPERIENCE FACTORY (Experience Packaging)
Develop representative languages for products processes	Develop techniques for abstraction generalization tailoring formalization analysis/synthesis
Develop techniques for design/implementation data collection/validation/analysis validation and verification	Experiment with techniques
Build automatic support tools	Package and integrate for reuse experimental results processes/products

In a correct implementation of the experience factory paradigm, the projects and the factory will have different process models. Each project will choose its process model based on the characteristics of the software product that will be delivered, whereas the factory will define (and change) its process model based upon organizational and performance issues. The main product of the experience factory is the experience package. There are a variety of software engineering experiences that can be packaged: resource baselines and models; change and defect baselines and models; product baselines and models; process definitions and models; method and technique models and evaluations; products; lessons learned; and quality models. The content and structure of an experience package vary based on the kind of experience clustered in the package. There is, generally, a central element that determines what the package is: a software life-cycle product or process, a mathematical relationship, an empirical or theoretical model, a data base, etc. This central element can be used to identify the experience package and produce a taxonomy of experience packages based on the characteristics of this central element:

- Product packages (programs, architectures, designs)
- Tool packages (constructive and analytic tools)
- Process packages (process models, methods)
- Relationship packages (cost and defect models, resource models, etc.)
- Management packages (guidelines, decision support models)
- Data packages (defined and validated data, standardized data, etc.)

The structure and functions of an efficient implementation of the experience factory concept are modeled on the characteristics and the goals of the organization it supports. Therefore, different levels of abstraction best describe the architecture of an experience factory in order to introduce the specificity of each environment at the right level without losing the representation of the global picture and the ability to compare different solutions [8].

The levels of abstraction that the SEL proposes to represent the architecture of an experience factory are as follows:

- Reference level: This first and more abstract level represents the activities in the experience factory by active objects, called architectural agents. They are specified by their ability to perform specific tasks and to interact with each other.
- Conceptual level: This level represents the interface of the architectural agents and the flows of data and control among them. They specify who communicates with whom, what is done in the experience factory, and what is done in the project organization. The boundary of the experience factory, i.e., the line that separates it from the project organization, is defined at this level based on the needs and characteristics of an organization. It can evolve as these needs and characteristics evolve.
- Implementation level: This level defines the actual technical and organizational implementation of the architectural agents and their connections at the conceptual level. They are assigned process and product models, synchronization and communication rules, and appropriate performers (people or computers). Other implementation details, such as mapping the agents over organizational departments, are included in the specifications provided at this level.

The architecture of the experience factory can be regarded as a special instance of an experience package whose design and evolution are based on the levels of abstraction just introduced and on the methodological framework of the improvement paradigm applied to the specific architecture.

The Software Engineering Laboratory (SEL) is an operating example of an experience factory. Figure 3 shows the conceptual level of the SEL experience factory, identifying the primary architectural agents and the interactions among them. The remaining sections describe the SEL implementation of the experience factory concept. They discuss its background, operations, and achievements, and assess the impact it has had on the production environment it supports.

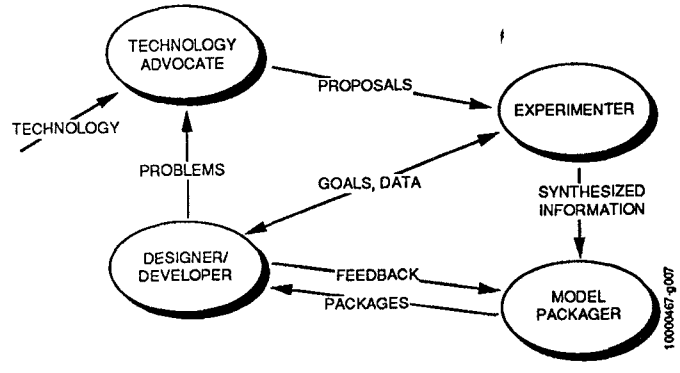


Figure 3. The SEL—Conceptual Level

2. SEL BACKGROUND

The SEL was established in 1976 as a cooperative effort among the University of Maryland, the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC), and Computer Sciences Corporation (CSC). Its goal was to understand and improve the software development process and its products within GSFC's Flight Dynamics Division (FDD). At that time, although significant advances were being made in developing new technologies (e.g., structured development practices, automated tools, quality assurance approaches, and management tools), there was very limited empirical evidence or guidance for applying these promising, yet immature, techniques. Additionally, it was apparent that there was very limited evidence available to qualify or to

quantify the existing software process and associated products, let alone understand the impact of specific process methods. Thus, the SEL staff initiated efforts to develop some means by which the software process could be understood (through measurement), qualified, and measurably improved through continually expanding understanding, experimentation, and process refinement.

This working relationship has been maintained continually since its inception with relatively little change to the overall goals of the organization. In general, these goals have matured rather than changed; they are as follows:

1. Understand: Improve insight into the software process and its products by characterizing a production environment.
2. Assess: Measure the impact that available technologies have on the software process. Determine which technologies are beneficial to the environment and, most importantly, how the technologies must be refined to best match the process with the environment.
3. Package/Infuse: After identifying process improvements, package the technology in a form that allows it to be applied in the production organization.

These goals are addressed sequentially, in an iterative fashion, as shown in Figure 4.

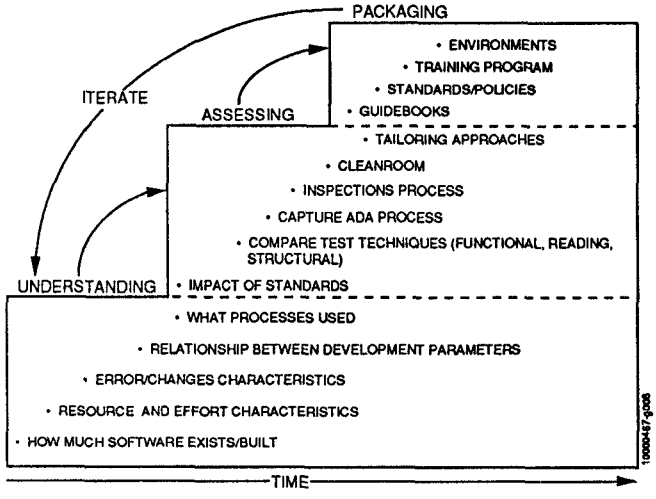


Figure 4. SEL Process Improvement Steps

The approach taken to attaining these goals has been to apply potentially beneficial techniques to the development of production software and to measure the process and product in enough detail to **quantifiably** assess the applied technology. Measures of concern, such as cost, reliability, and/or maintainability, are defined as the organization determines the major near- and long-term objectives for its software development process improvement program. Once those objectives are known, the SEL staff designs the experiment; that is, it defines the particular data to be captured and the questions that must be addressed in each experimental project.

All of the experiments conducted by the SEL have occurred within the production environment of the flight dynamics software development facility at NASA/GSFC. The SEL production environment consists of projects that are classified as mid-sized software systems. The average project lasts 2 to 3-1/2 years, with an average staff size of 15 software developers. The average software size is 175 thousand source lines of code (KSLOC), counting commentary, with about 25 percent reused from previous development efforts. Virtually all projects in this environment are scientific ground-based systems, although some embedded systems have been developed. Most software is developed in FORTRAN, although Ada is starting to be used more frequently. Other languages, such as Pascal and Assembly, are used occasionally. Since this environment is relatively consistent, it is conducive to the experimentation process. In the SEL, there exists a homogeneous class of software, a stable development environment, and a controlled, consistent, management and development process.

3. SEL OPERATIONS

The following three major functional groups support the experimentation and studies within the SEL (Figure 5):

1. **Software developers**, who are responsible for producing the flight dynamics application software
2. **Software engineering analysts**, who are the researchers responsible for carrying out the experimentation process and producing study results
3. **Data base support staff**, who are responsible for collecting, checking, and archiving all of the information collected from the development efforts

During the past 15 years, the SEL has collected and archived data on over 100 software development projects in the organization. The data are also used to build typical project profiles against which ongoing projects can be compared and evaluated. The SEL provides managers in this environment with tools (online and paper) for monitoring and assessing project status.

Typically, there are 6 to 10 projects simultaneously in progress in the flight dynamics environment. As was mentioned earlier, they average 175 KSLOC, ranging from small (6–8 KSLOC) to large (300–400 KSLOC), with a few exceeding 1 million source lines of code (MSLOC). Each project is considered an experiment within the SEL, and the goal is to extract detailed information to understand the process better and to provide guidance to future projects.

To support the studies and to support the goal of continually increasing understanding of the software development process, the SEL regularly collects detailed data from its development projects. The types of data collected include cost (measured in effort), process, and product data. Process data include information about the project, such as the methodology, tools and techniques used, and information about personnel experience and training. Product data include size (in SLOC), change and error information, and the results of postdevelopment static analysis of the delivered code.

The data may be somewhat different from one project to another since the goals for a particular experiment may be different between projects. There is a basic set of information (such as effort and error data) that is collected for every project. However, as changes are made to specific processes (e.g., Ada projects), the detailed data collected may be modified. For example, Figure 6 shows the standard error report form, used on all projects, and the modified Ada version, used for specific projects where Ada is being studied.

As the information is collected, it is quality assured and placed in a central data base. The analysts then use these data together with other information, such as subjective lessons learned, to analyze the impact of a specific software process and to measure and then feed back results to both ongoing projects and follow-on projects.

The data are used to build predictive models for future projects and to provide a rationale for refining particular software processes being used. As the data are analyzed, papers and reports are generated that reflect results of the numerous studies. Additionally, the results of the analysis are packaged as standards, policies, training materials, and management tools.

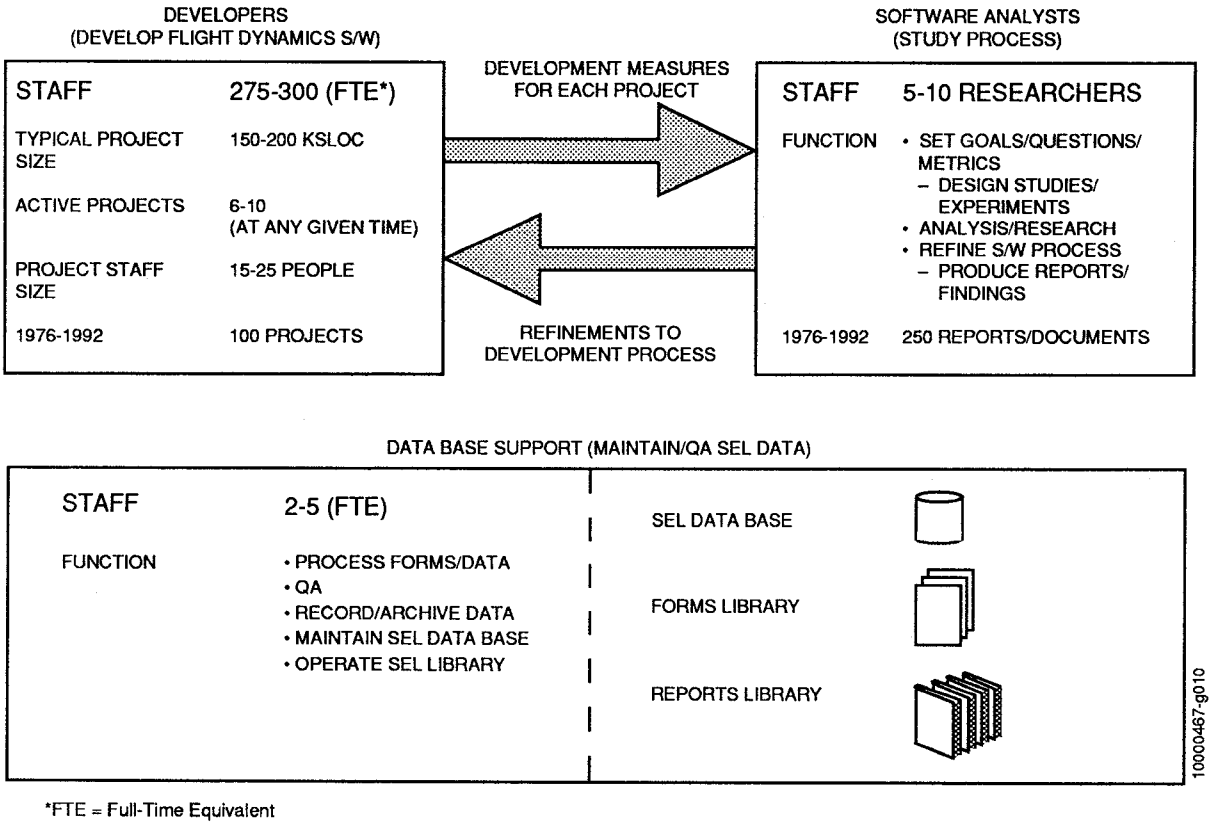


Figure 5. SEL Functional Groups

4. SEL DATA ANALYSIS

The overall concept of the experience factory has continually matured within the SEL as understanding of the software process has increased. The experience factory goal is to demonstrate continual improvement of the software process within an environment by carrying out analysis, measurement, and feedback to projects within the environment. The steps, previously described, include **understanding, assessment/refinement,** and **packaging.** The data described in the previous section are used as one major element that supports these three activities in the SEL. In this section, examples are given to demonstrate the major stages of the experience factory.

4.1. UNDERSTANDING

Understanding what an organization does and how that organization operates is fundamental to any attempt to plan, manage, or improve the software process. This is especially true for software development organizations. The following two examples illustrate how understanding is supported in an operation such as the SEL.

Effort distribution (i.e., which phases of the life cycle consume what portion of development effort) is one baseline characteristic of the SEL software development process. Figure 7 presents the effort distributions for 11 FORTRAN projects, by life-cycle phase and by activity. The phase data count hours charged to a project during each calendar phase. The activity data count all hours attributed to a particular activity (as reported by the programmer), regardless of when in the life cycle the activity occurred. Understanding these distributions is important to assessing the similarities/differences observed on an ongoing project, planning new efforts, and evaluating new technology.

The error detection rate is another interesting model from the SEL environment. There are two types of information in this model. The first is the absolute error rate expected in each phase. By collecting the information on software errors, the SEL has constructed a model of the expected error rate in each phase of the life cycle. The SEL expects about four errors per 1000 SLOC during implementation: two during system test, one during acceptance test, and one-half during operation and maintenance. Analysis of more recent projects indicates that these absolute error rates are declining as the software development process and technology improve.

The trend that can be derived from this model is that the error detection rates reduce by 50 percent in each subsequent phase (Figure 8). This pattern seems to be independent of the actual values of the error rates; it is still true in the recent projects where the overall error rates are declining. This model of error rates, as well as numerous other similar types of models, can be used to better predict, manage, and assess change on newly developed projects.

4.2. ASSESSING/REFINING

In the second major stage of the experience factory, elements of the process (such as specific software development techniques) are assessed, and the evolving technologies are tailored to the particular environment. Each project in the SEL is considered to be an experiment in which some software method is studied in detail. Generally, the subject of the study is a specific modification to the standard process, a process that obviously comprises numerous software methods.

CHANGE REPORT FORM

Name:_____ Approved by:_____

Project:_____ Date:_____

Section A – Identification

Describe the change: (What, why, how)_____

Effect: What components are changed?

Prefix	Name	Version

(Attach list if more space is needed)

Effort: What additional components were examined in determining what change was needed?

Location of developer's source files_____

	month	day	year
Need for change determined on:			
Change completed (incorporated into system)			

Check here if change involves Ada components (if so, complete questions on reverse side) ☐

	1 hr/less	1 hr/1 day	1/3 days	>3 days
Effort in person time to isolate the change (or error):				
Effort in person time to implement the change (or correction):				

Section B – All Changes

Type of Change (Check one)

- ☐ Error correction
- ☐ Planned enhancement
- ☐ Implementation of requirements change
- ☐ Improvement of clarity, maintainability, or documentation
- ☐ Improvement of user services
- ☐ Insertion/deletion of debug code
- ☐ Optimization of time/space/accuracy
- ☐ Adaptation to environment change
- ☐ Other (Describe below)

Effects of Change

Y	N	
☐	☐	Was the change or correction to one and only one component? (Must match Effect in Section A)
☐	☐	Did you look at any other component? (Must match Effort in Section A)
☐	☐	Did you have to be aware of parameters passed explicitly or implicitly (e.g., COMMON blocks) to or from the changed components?

Section C – For Error Corrections Only

Source of Error (Check one)

- ☐ Requirements
- ☐ Functional specifications
- ☐ Design
- ☐ Code
- ☐ Previous change

Class of Error (Check most applicable)*

- ☐ Initialization
- ☐ Logic/control structure (e.g., flow of control incorrect)
- ☐ Interface (internal) (module-to-module communication)
- ☐ Interface (external) (module to external communication)
- ☐ Data (value or structure) (e.g., wrong variable used)
- ☐ Computational (e.g., error in math expression)

*If two are equally applicable, check th one higher on the list.

Characteristics (Check Y or N for all)

Y	N	
☐	☐	Omission error (e.g., something was left out)
☐	☐	Commission error (e.g., something incorrect was included)
☐	☐	Error was created by transcription (clerical)

For Librarian's Use Only

Number:_____
Date:_____
Entered by:_____
Checked by:_____

NOVEMBER 1991

Standard Form

CHANGE REPORT FORM
Ada Project Additional Information

1. Check which Ada feature(s) was involved in this change (Check all that apply)
   - ☐ Data typing
   - ☐ Subprograms
   - ☐ Exceptions
   - ☐ Generics
   - ☐ Program structure and packaging
   - ☐ Tasking
   - ☐ System-dependent features
   - ☐ Other, please specify_____ (e.g., I/O, Ada statements)
2. For an error involving Ada components:
   a. Does the compiler documentation or the language reference manual explain the feature clearly? _____ (Y/N)
   b. Which of the following is most true? (Check one)
      - ☐ Understood features separately but not interaction
      - ☐ Understood features, but did not apply correctly
      - ☐ Did not understand features fully
      - ☐ Confused feature with feature in another language
   c. Which of the following resources provided the information needed to correct the error? (Check all that apply)
      - ☐ Class notes
      - ☐ Ada reference manual
      - ☐ Own project team member
      - ☐ Own memory
      - ☐ Someone not on team
      - ☐ Other
   d. Which tools, if any, aided in the detection or correction of this error? (Check all that apply)
      - ☐ Compiler
      - ☐ Symbolic debugger
      - ☐ Language-sensitive editor
      - ☐ CMS
      - ☐ Source Code Analyzer
      - ☐ P&CA (Performance and Coverage Analyzer)
      - ☐ DEC test manager
      - ☐ Other, specify_____
3. Provide any other information about the interaction of Ada and this change that you feel might aid in evaluating the change and using Ada

10000467-g020

NOVEMBER 1991

Extended Ada Form

Figure 6. Error Report Forms

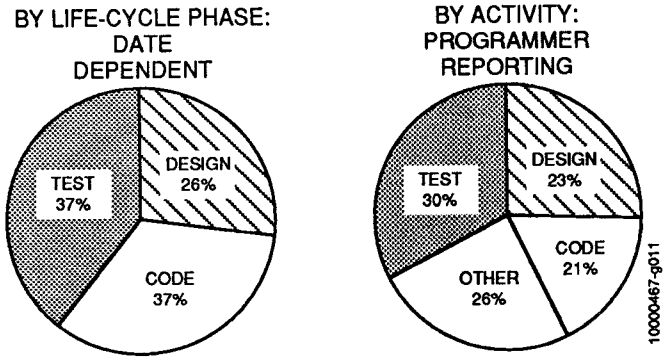


Figure 7. Effort Distribution

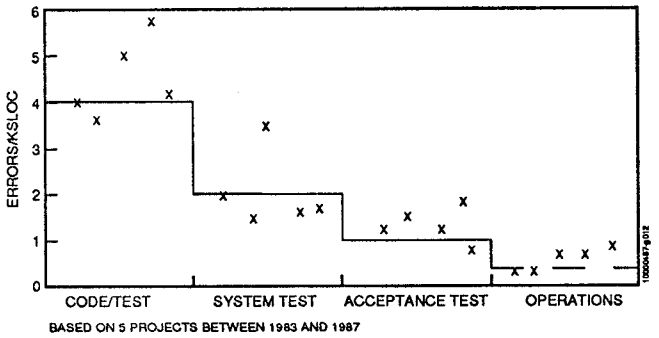


Figure 8. Derived SEL Error Model

One recent study that exemplifies the assessment stage involves the Cleanroom software methodology [9]. This methodology has been applied on three projects within the SEL, each providing additional insight into the Cleanroom process and each adding some element of "refinement" to the methodology for this one environment.

The SEL trained teams in the methodology, then defined a modified set of Cleanroom-specific data to be collected. The projects were studied in an attempt to assess the impact that Cleanroom had on the process as well as on such measures as productivity and reliability. Figure 9 depicts the characteristics of the Cleanroom changes, as well as the results of the three experiments.

The Cleanroom experiments included significant changes to the standard SEL development methodology, thereby requiring extensive training, preparation, and careful execution of the studies. Detailed experimentation plans were generated for each of the studies (as they are for all such experiments), and each included a description of the goals, the questions that had to be addressed, and the metrics that had to be collected to answer the questions.

Since this methodology consists of multiple specific methods (e.g., box structure design, statistical testing, rigorous inspections), each particular method had to be analyzed along with the full, integrated, Cleanroom methodology in general. As a result of the analysis, Cleanroom has been "assessed" as a beneficial approach for the SEL (as measured by specific goals of these studies), but specific elements of the full methodology had to be tailored to better fit the particular SEL environment. The tailoring and modifying resulted in a revised Cleanroom process model, written in the form of a process handbook [10], for future applications to SEL projects. That step is the "packaging" component of the experience factory process.

4.3. PACKAGING

The final stage of a complete experience factory is that of packaging. After beneficial methods and technologies are identified, the organization must provide feedback to ensuing projects by capturing the process in the form of standards, tools, and training. The SEL has produced a set of standards for its own use that reflect the results of the studies it has conducted. It is apparent that such standards must continually evolve to capture modified characteristics of the process. (The SEL typically updates its basic standard every 5 years.) Examples of standards that have been produced as part of the packaging process include:

- *Manager's Handbook for Software Development [11]*
- *Recommended Approach to Software Development [12]*

One additional example of an extensive packaging effort in the SEL is a management tool called the Software Management Environment (SME). The concepts of the SME, which is now an operational tool used locally in the SEL, have evolved over 8 years. This tool accesses SEL project data, models, relationships, lessons learned, and managers' rules of thumb to present project characteristics to the manager of an ongoing project. This allows the manager to gain insight into the project's consistency with or deviation from the norm for the environment (Figure 10).

This example of "packaging" reflects the emphasis that must be placed on making results of software projects, in the form of lessons learned, refined models, and general understanding, easily available to other follow-on development projects in a particular organization.

The tool searches the collection of 15 years of experience archived in the SEL to select appropriate, similar project data so that managers can plan, monitor, predict, and better understand their own project based on the analyzed history of similar software efforts.

As an example, all of the error characteristics of the flight dynamics projects have resulted in the error model depicted in Figure 8, where history has shown typical software error rates in the different phases of the life cycle. As new projects are developed and error discrepancies are routinely reported and added to the SEL data base, the manager can easily compare error rates on his or her project with typical error rates on completed, similar projects. Obviously, the data are environment dependent, but the concepts of measurement, process improvement, and packaging are applicable to all environments.

5. ADA ANALYSIS

A more detailed example of one technology that has been studied in the SEL within the context of the experience factory is that of Ada. By 1985, the SEL had achieved a good understanding of how software was developed in the FDD; it had baselined the development process and had established rules, relationships, and models that improved the manageability of the process. It had also fine-tuned its process by adding and refining techniques within its standard methodology. Realizing that Ada and object-oriented techniques offered potential for major improvement in the flight dynamics environment, the SEL decided to pursue experimentation with Ada.

The first step was to set up expectations and goals against which results would be measured. The SEL's well-established baseline and set of measures provided an excellent basis for comparison. Expectations included a change in the effort distribution of development activities (e.g., increased design and decreased testing); no greater cost per new line of code; increased reuse; decreased maintenance costs; and increased reliability (i.e., lower error rates, fewer interface errors, and fewer design errors).

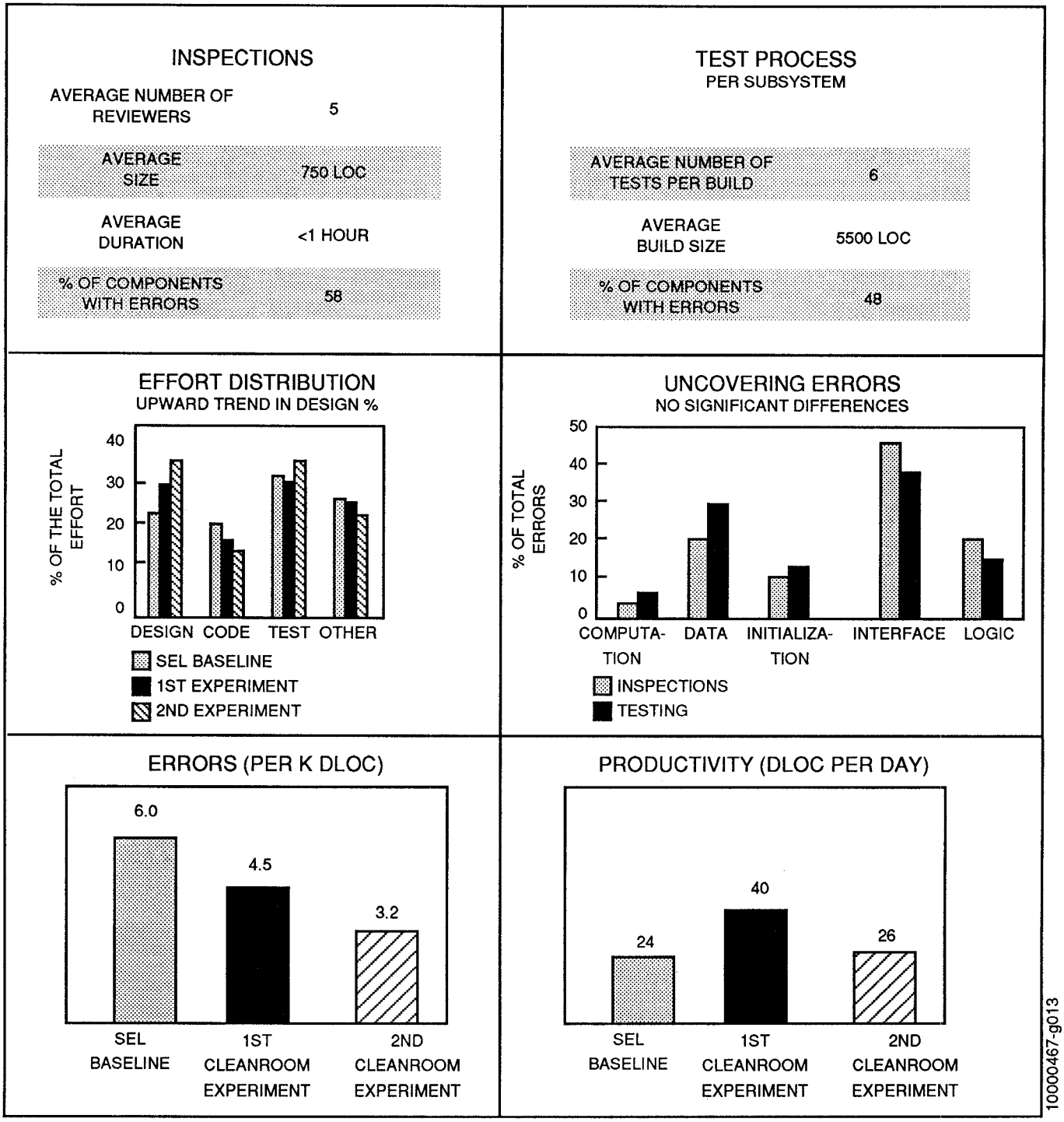


Figure 9. Cleanroom Assessment in the SEL

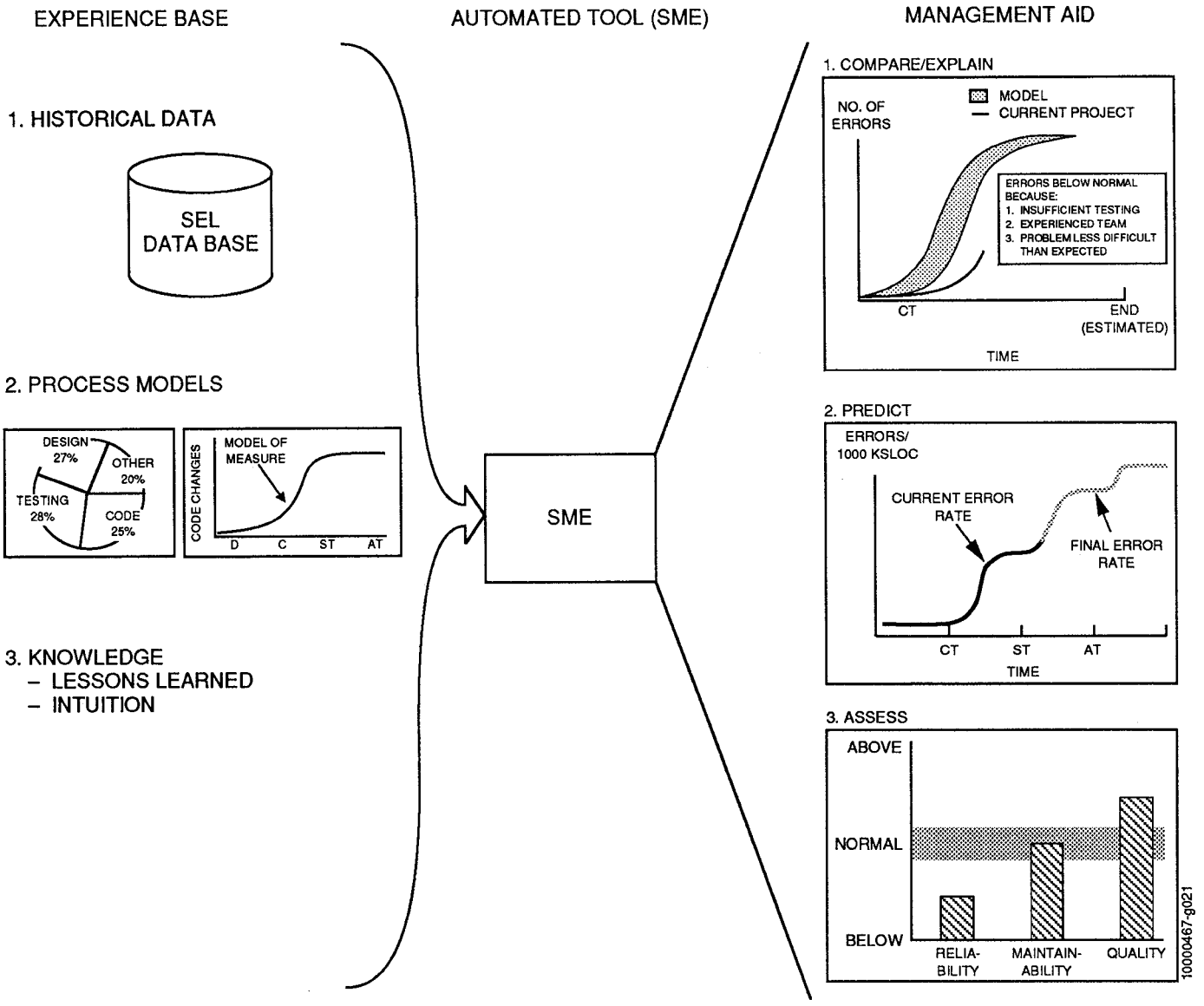


Figure 10. SME: A Tool for "Packaging"

The SEL started with a small, controlled experiment in which two versions of the same system were developed in parallel: one was developed in FORTRAN using the standard SEL structured methodology, and the other was developed in Ada using an object-oriented development (OOD) methodology. Because the Ada system would not become operational, analysts had time to investigate new ideas and learn about the new technology while extracting good calibration information for comparing FORTRAN and Ada projects, such as size ratios, average component size, error rates, and productivity. These data provided a reasonable means for planning the next set of Ada projects that, even though they were small, would deliver mission support software.

Over the past 6 years the SEL has completed 10 Ada/OOD projects, ranging in size from 38 to 185 KSLOC. As projects completed and new ones started, the methodology was continually evaluated and refined. Some characteristics of the Ada environment emerged early and have remained rather constant; others took time to stabilize. For example, Ada projects have shown no significant change in effort distribution or in error classification when compared with the SEL FORTRAN baseline. However, reuse has increased dramatically, as shown in Figure 11.

Over the 6-year period, the use of Ada and OOD has matured. Source code analysis of the Ada systems, grouped chronologically, revealed a maturing use of key Ada features, such as generics, strong typing, and packaging, whereas other features, such as tasking, were deemed inappropriate for the application. Generics, for example, were not only used more often in the recent systems, increasing from 8 to 50 percent of the system, but they were also used in more sophisticated ways, so that parameterization increased eightfold. Moreover, the use of Ada features has stabilized over the last 3 years, creating a SEL baseline for Ada development.

The cost to develop new Ada code has remained higher than the cost to develop new FORTRAN code. However, because of the high reuse, the cost to deliver an Ada system has significantly

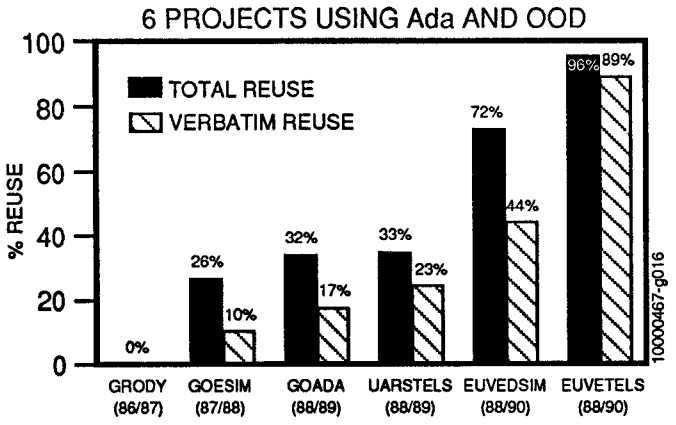


Figure 11. Reuse Trends

decreased and is now well below the cost to deliver an equivalent FORTRAN system (Figure 12).

Reliability of Ada systems has also improved as the environment has matured. Although the error rates for Ada systems, shown in Figure 13, were significantly lower from the start than those for FORTRAN, they have continued to decrease even further. Again, the high level of reuse in the later systems is a major contributor to this greatly improved reliability.

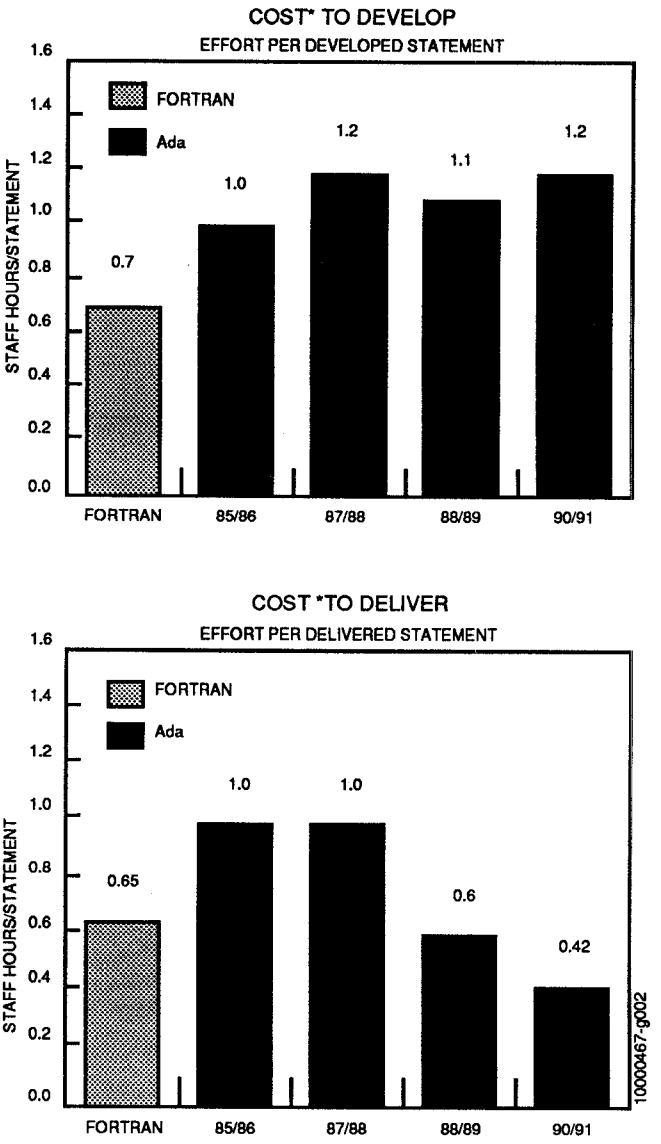
During this 6-year period, the SEL went through various levels of packaging the Ada/OOD methodology. On the earliest project in 1985, when OOD was still very young in the industry, the SEL found it necessary to tailor and package their own General Object-Oriented Development (GOOD) methodology [13] for use in the flight dynamics environment. This document (produced in 1986) adjusted and extended the industry standard for use in the local environment. In 1987, the SEL also developed an Ada Style Guide [14] that provided coding standards for the local environment. Commercial Ada training courses, supplemented with limited project-specific training, constituted the early training in these techniques. The SEL also produced lessons-learned reports on the Ada/OOD experiences, recommending refinements to the methodology.

Recently, because of the stabilization and apparent benefit to the organization, Ada/OOD is being packaged as part of the baseline SEL methodology. The standard methodology handbooks [11, 12] include Ada and OOD as mainstream methods. In addition, a complete and highly tailored training program is being developed that teaches Ada and OOD as an integrated part of the flight dynamics environment.

Although Ada/OOD will continue to be refined within the SEL, it has progressed through all stages of the experience factory, moving from a candidate trial methodology to a fully integrated and packaged part of the standard methodology. The SEL considers it baselined and ready for further incremental improvement.

6. IMPLICATIONS FOR THE DEVELOPMENT ORGANIZATION

For 15 years, NASA has been funding the efforts to carry out experiments and studies within the SEL. There have been significant costs and a certain level of overhead associated with these efforts; a logical question to ask is "Has there been significant benefit?" The historical information strongly supports a very positive answer. Not only has the expenditure of resources been a wise investment for the NASA flight dynamics environment, but members of the SEL strongly believe that such efforts should be



* Cost = Effort/Size
Size (developed) = New statements + 20% of reused
Size (delivered) = Total delivered statements

NOTE: Cost per statement is used here as the basis for comparison, since the SEL has found a 3-to -1 ratio when comparing Ada with FORTRAN source lines of code (carriage returns) but a 1-to-1 ratio when comparing statements.

Figure 12. Costs To Develop and Deliver

commonplace throughout both NASA and the software community in general. The benefits far outweigh the costs.

Since the SEL's inception in 1976, NASA has spent approximately $14 million dollars (contract support) in the three major support areas required by this type of study environment: research (defining studies and analyzing results), technology transfer (producing standards and policies), and data processing (collecting forms and maintaining data bases). Approximately 50 staff-years of NASA personnel effort have been expended on the SEL. During this same period, the flight dynamics area has spent approximately $150 million on building operational software, all of which has been part of the study process.

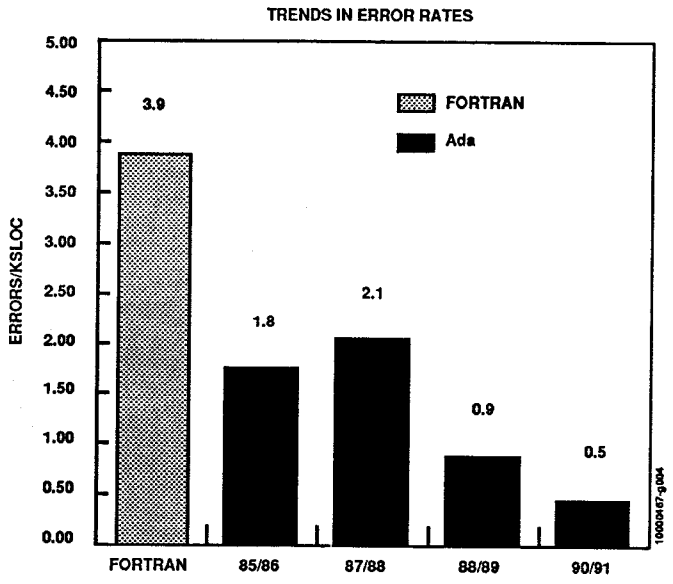


Figure 13. Trends in Error Rates

During the past 15 years, the SEL has had a significant impact on the software being developed in the local environment, and there is strong reason to believe that many of the SEL studies have had a favorable impact on a domain broader than this one environment. Examples of the changes that have been observed include the following:

1. The cost per line of new code has decreased only slightly, about 10 percent—which, at first glance might imply that the SEL has failed at improving productivity. Although the SEL finds that the cost to produce a **new** source statement is nearly as high as it was 15 years ago, there is appreciable improvement in the functionality of the software, as well as a tremendous increase in the complexity of the problems being addressed [15]. Also, there has been an appreciable increase in the reuse of software (code, design, methods, test data, etc.), which has driven the overall cost of the equivalent functionality down significantly. When the SEL merely measures the cost to produce one new source statement, the improvement is small; but when it measures overall cost and productivity, the improvement is significant.
2. Reliability of the software has improved by 35 percent. As measured by the number of errors per thousand lines of code (E/KSLOC), flight dynamics software has improved from an average of 8.4 E/KSLOC in the early 1980s to approximately 5.3 E/KSLOC today. These figures cover the software phases through acceptance testing and delivery to operations. Although operations and maintenance data are not nearly so extensive as the development data, the small amount of data available indicates significant improvement in that area as well.
3. The "manageability" of software has improved dramatically. In the late 1970s and early 1980s, the environment experienced wide variations in productivity, reliability, and quality from project to project. Today, however, the SEL has excellent models of the process; it has well-defined methods; and managers are better able to predict, control, and manage the cost and quality of the software being produced. This conclusion is substantiated by recent SEL data that show a continually improving set of models for planning, predicting, and estimating all development projects in the flight dynamics environment. There no longer is the extreme uncertainty in estimating such common parameters as cost, staffing, size, and reliability.
4. Other measures include the effort put forth in rework (e.g., changing and correcting) and in overall software reuse. These measures also indicate a significant improvement to the software within this one environment.

In addition to the common measures of software (e.g., cost and reliability), there are many other major benefits derived from a "measurement" program such as the SEL's. Not only has the understanding of software significantly improved within the research community, but this understanding is apparent throughout the entire development community within this environment. Not only have the researchers benefited, but the developers and managers who have been exposed to this effort are much better prepared to plan, control, assure, and, in general, develop much higher quality systems. One view of this program is that it is a major "training" exercise within a large production environment, and the 800 to 1000 developers and managers who have participated in development efforts studied by the SEL are much better trained and effective software engineers. This is due to the extensive training and general exposure all developers get from the research efforts continually in progress.

In conclusion, the SEL functions as an operational example of the experience factory concept. The conceptual model for the SEL presented in Section 1 maps to the functional groups discussed under SEL operations in Section 3. The experience base in Figure 2 is realized by the SEL data base and its archives of management models and relationships [16]. The analysis function from Figure 2 is performed by the SEL team of software engineering analysts, who analyze processes and products to **understand** the environment, then plan and execute experiments to **assess** and **refine** the new technologies under study. Finally, the synthesis function of the experience factory maps to the SEL's activities in packaging new processes and technology in a form tailored specifically to the flight dynamics environment. The products of this synthesis, or packaging, are the guidelines, standards, and tools the SEL produces to infuse its findings back into the project organization. These products are the experience packages of the experience factory model.

Current SEL efforts are focused on addressing two major questions. The first is "How long does it take for a new technology to move through all the stages of the experience factory?" That is, from understanding and baselining the current environment, through assessing the impacts of the technology and refining it, to packaging the process and infusing it into the project organization. Preliminary findings from the SEL's Ada and Cleanroom experiences indicate a cycle of roughly 6 to 9 years, but further data points are needed. The second question the SEL is pursuing is "How large an organization can adopt the experience factory model?" The SEL is interested in learning what the scaleup issues are when the scope of the experience factory is extended beyond a single environment. NASA is sponsoring an effort to explore the infusion of SEL-like implementations of the experience factory concept across the entire Agency.

ACKNOWLEDGMENT

Material for this paper represents work not only of the authors listed, but of many other SEL staff members. Special acknowledgment is given to Gerry Heller of CSC, who played a key role in editing this paper.

REFERENCES

Numerous papers, reports, and studies have been generated over the SEL's 15-year existence. A complete listing of these can be found in the *Annotated Bibliography of Software Engineering*

Laboratory Literature, SEl-82-1006, L. Morusiewicz and J. Valett, November 1991.

This bibliography may be obtained by contacting:

The SEL Library
Code 552
NASA/GSFC
Greenbelt, MD 20771

A listing of references specific to this paper follows.

1. V. R. Basili, "Towards a Mature Measurement Environment: Creating a Software Engineering Research Environment," Proceedings of the Fifteenth Annual Software Engineering Workshop, NASA/GSFC, Greenbelt, Maryland, SEL–90–006, November 1990.
2. V. R. Basili, "Quantitative Evaluation of a Software Engineering Methodology," Proceedings of the First Pan Pacific Computer Conference, Melbourne, Australia, September 1985.
3. V.R. Basili and D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," IEEE Transactions on Software Engineering, November 1984, pp. 728–738.
4. V. R. Basili, "Software Development: A Paradigm for the Future (Keynote Address)," Proceedings COMPSAC '89, Orlando, Florida, September 1989, pp. 471–485.
5. V.R. Basili and H.D. Romback, "Tailoring the Software Process to Project Goals and Environments," Proceedings of the Ninth International Conference on Software Engineering, Monterey, California, March 30 – April 2, 1987, pp. 345–357.
6. V.R. Basili and H.D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," IEEE Transactions on Software Engineering, Vol. 14, No. 6., June 1988, pp. 758–773.
7. V. R. Basili and G. Caldiera, "Methodological and Architectural Issues in the Experience Factory," Proceedings of the Sixteenth Annual Software Engineering Workshop, NASA/GSFC, Greenbelt, Maryland, Software Engineering Laboratory Series, December 1991.
8. V. R. Basili, G. Caldiera, and G. Cantone, "A Reference Architecture for the Component Factory," ACM Transactions on Software Engineering and Methodology, Vol. 1, No. 1, January 1992, pp. 53–80.
9. H.D. Mills, M. Dyer, and R.C. Linger, "Cleanroom Software Engineering," IEEE Software, November 1990, pp. 19–24.
10. S. Green, *Software Engineering Laboratory (SEL) Cleanroom Process Model*, SEL-91-004, November 1991.
11. L. Landis, F. E. McGarry, S. Waligora, et al., *Manager's Handbook for Software Development (Revision 1)*, SEL-84-101, November 1990.
12. F.E. McGarry, G. Page, S. Eslinger, et al., *Recommended Approach to Software Development*, SEL-81-205, April 1983. Revision 3 in preparation; scheduled for publication June 1992.
13. E. Seidewitz and M. Stark, *General Object-Oriented Software Development*, SEL-86-002, August 1986.
14. E. Seidewitz et al., *Ada® Style Guide (Version 1.1)*, SEL-87-002, May 1987.
15. D. Boland et al., *A Study on Size and Reuse Trends in Attitude Ground Support Systems (AGSSs) Developed for the Flight Dynamics Division (FDD) (1976-1988)*, NASA/GSFC, CSC/TM-89/6031, February 1989.
16. W. Decker, R. Hendrick, and J. Valett, *Software Engineering Laboratory (SEL) Relationships, Models, and Management Rules*, SEL-91-001, February 1991.