Software Engineering Program

Software Process Improvement Guidebook

January 1996

NATIONAL AERONAUTICS AND
SPACE ADMINISTRATION
WASHINGTON,DC

Software

Engineering

Program

Software Process
Improvement Guidebook

January 1996

NATIONAL AERONAUTICS AND
SPACE ADMINISTRATION
WASHINGTON,DC

Foreword

This document is a product of the NASA Software Program, an Agency-wide program to promote continual improvement of software engineering within NASA. The goals and strategies for this program are documented in the *NASA Software Strategic Plan*, July 13, 1995.

Additional information is available from the NASA Software IV&V facility on the World Wide Web at site http://www.ivv.nasa.gov/.

Office of the Chief Engineer

Software Process Improvement Guidebook

Approvals

Rose Pajerski
Technical RTOP Manager

Donald Sova
Manager, Software Engineering Program

Contents

Chapter 4. Implementation of the Software Process Improvement Program

Chapter 5. Management of the Software Process Improvement Program

Appendix A. Glossary of Terms

Abbreviations and Acronyms

References

Index

Figures

Tables

Chapter 1. Introduction

Document Highlights	
CHAPTER 1: **Introduction**	
CHAPTER 2: **Software Process Improvement Framework**	Organization · Assess · Understand · Domains · Package
CHAPTER 3: **Structure and Operation of the Software Process Improvement Organization**	Analysts · Support Staff · Developers
CHAPTER 4: **Implementation of the Software Process Improvement Program**	Start · Baseline · Process · Structure · Commitment
CHAPTER 5: **Management of the Software Process Improvement Program**	Key Points · Cost · Benefits

1.1 Background

One of the most significant challenges faced by the software engineering community in recent years has been to continually capitalize on software development and maintenance experiences, whether good or bad. Capitalizing involves application of new technologies and evolution of technologies already widely in use, as well as the definition and adoption of standards. One goal of software engineering practitioners is to make sure that all those activities, which generally can be classified as process and product improvements, are based upon understanding the impact on the target application domain in an appropriate way.

There is an evident need to implement some means by which every software activity provides new and improved insight into continually improving methods for developing and maintaining software; every experience must be used to gain new knowledge. To do so, every software organization should be embedded in an infrastructure aimed at capitalizing on previous experience. This concept is derived from several specific programs within the National Aeronautics and Space Administration (NASA) [e.g., the Software Engineering Laboratory (SEL) of NASA/Goddard Space Flight Center (GSFC), the Software Engineering and Analysis Laboratory (SEAL) at NASA/Langley Research Center (LaRC), and the Jet Propulsion Laboratory's (JPL's) Software Resource Center (SORCE)] and is similar in functionality to another concept called the software Experience Factory (Reference 1). Continual improvement based on specific experiences is the underlying concept of the NASA software process improvement program.

Although not specifically designated in the title, both software process and software product are emphasized in this guidebook and the overall NASA program. Improvements in the software process result in measurable improvements to the software product, hence "software process improvement" implies "software process AND product improvement." The importance of both is emphasized throughout this document.

The NASA software process improvement program supports continual software quality improvement and the use and reuse of software experience by developing, updating, and making available key software technologies, knowledge, and products originating from operational software projects and specific experimentation.

This guidebook addresses the needs of the NASA software community by offering a framework based on an evolutionary approach to quality management tailored for the software business. This approach is supported by an organizational infrastructure for capturing and packaging software experiences and supplying them to ongoing and future projects.

1.2 Purpose

The purpose of this document is to provide experience-based guidance in implementing a software process improvement program in any NASA software development or maintenance community.

This guidebook details how to define, operate, and implement a working software process improvement program. It describes the concept of the software process improvement program and its basic organizational components. It then describes the structure, organization, and

operation of the software process improvement program, illustrating all these concepts with specific NASA examples. The information presented in the document is derived from the experiences of several NASA software organizations, including the SEL, the SEAL, and the SORCE. Their experiences reflect many of the elements of software process improvement within NASA.

This guidebook presents lessons learned in a form usable by anyone considering establishing a software process improvement program within his or her own environment. This guidebook attempts to balance general and detailed information. It provides material general enough to be usable by NASA organizations whose characteristics do not directly match those of the sources of the information and models presented herein. It also keeps the ideas sufficiently close to the sources of the practical experiences that have generated the models and information.

1.3   Organization

This "Introduction" is followed by four additional chapters.

Chapter 2 presents an overview of concepts pertaining to software process improvement, including the organizational structure needed to support process improvement, the three-phase software process improvement approach, the scope of an organization to which process improvement is to be applied (domain), and the unique aspects of the software process improvement framework presented in this document versus other software process improvement approaches.

Chapter 3 presents the structure of a typical NASA software process improvement program and describes the major components of the software process improvement organization. The chapter gives an overview of each of the three organizational components and discusses the resources each component requires. It also presents details regarding the operation of the software process improvement program, describing the responsibilities, activities, and interaction of each of the three organizational elements. The chapter details how each component performs the activities associated with its process improvement responsibilities and how the three groups interact and operate on a daily basis.

Chapter 4 presents the steps for implementing software process improvement in an organization.

Chapter 5 addresses key management issues associated with the implementation and operation of a software process improvement program, including cost and potential benefits.

Chapter 2. The Software Process Improvement Framework

Chapter Highlights

STRUCTURE

Developers
Analysts
Support Staff

APPROACH

Understanding
Assessing
Packaging

DOMAINS

Scope of an organization to
which process improvement is
to be applied
Transfer of information across
domains
Improvement of software within
a domain

VARIOUS APPROACHES

NASA Approach
Capability Maturity Model
Emphasis on the need for continual,
sustained improvement of software

This chapter provides an overview of concepts pertaining to software process improvement within NASA. The first section discusses the key components of the software process improvement framework, specifically, the organizational structure needed to facilitate process improvement. The next sections cover the three-phase software process improvement approach and related concepts including "domain." (A domain is a classification scheme as it pertains to the application of the process improvement approach within a specific organization.) The last section discusses the unique aspects of this process improvement framework with

respect to other process improvement approaches. Some of the concepts introduced in this chapter are discussed in further detail in later chapters.

2.1 The Software Process Improvement Organization

Historically, software organizations have exhibited significant shortcomings in their ability to capitalize on the experiences gained from completed projects. Most of the insight has been passively obtained instead of aggressively pursued through specific plans and organizational infrastructures.

Software developers and managers, although well-meaning and interested, generally do not have the time or resources to focus on building corporate knowledge or organizational process improvements. (For this document, a "software developer" is defined to be any technical project personnel, including designers, development and maintenance programmers, technical managers, and any other technical contributors.) They have projects to run and software to deliver. Thus, collective learning and experience must become a corporate concern and be treated as a company asset. Reuse of experience and collective learning should be supported by an organizational infrastructure dedicated to developing, updating, and supplying upon request synthesized experiences and competencies. This infrastructure should emphasize achieving continual sustained improvement.

Software process improvement organizations within NASA are structures devoted to using lessons, data, and general experience from software projects to ensure that ongoing and ensuing efforts use the experiences gained to continually improve the associated organization's software products and processes.

Software process improvement organizations within NASA are dedicated to software process improvement and the reuse of experience. Each NASA software process improvement organization consists of

- *Developers,* who design, implement, and maintain software. They also provide project documentation and data gathered during development and operations.

- *Process Analysts* (hereafter referred to as analysts), who transform the data and information provided by the developers into reusable forms (e.g., standards, models, training) and supply them back to the developers. They provide specific support to the projects on the use of the analyzed and synthesized information, tailoring it to a format that is usable by and useful to a current software effort. In some programs, this element may be called the Software Engineering Process Group (SEPG).

- *Support Staff,* who provide services to the developers by supporting data collection and retrieval and to the analysts by managing the repository of information.

Although separate, these three components are intimately related to each other. Each component has its own goals, process, and plans, but together all three components have the mission of providing software that is continually improving in quality and cost effectiveness. Table 2-1 outlines the differences in focus among the three components comprising the software process improvement organization.

Table 2-1. Focus of Software Process Improvement Organization Components

Area	Developers	Analysts	Support Staff
Focus and Scope	Specific project	Multiple projects (specific domain)	Multiple projects (specific domain)
Goals	Produce, maintain software Satisfy user requirements	Analyze and package experience Support developers	Archive, maintain, and distribute development and maintenance experience
Approach	Use the most effective software engineering techniques	Assess the impact of specific technologies Package experience into models, standards, etc.	Maintain a repository of experiences, models, standards, etc.
Measure of Success	Delivery of quality software products on time and within budget	Reuse of empirical software experience by developers Improved products	Efficient collection, storage, and retrieval of information (data, models, reports, etc.)

The developers' goal is to deliver a software system. Their success is measured by delivering, on time and within budget, a software product that meets the needs of the user.

The analysts' goal is to analyze and package experiences into a form useful to the developers. They use information such as development environment profile, methods, characteristics, resources breakdown and utilization, error classes, and statistics to produce models of products and processes, evaluations, and refined development information. This set of products could include cost models, reliability models, domain-specific architectures and components, process models, policies, and tools. Every product of the analysts is derived from specific experiences of the developers. The success of the analysts is measured by their ability to provide to the developers, in a timely way, useful products, processes, and information. Ultimately, the success of the analysts is measured by improved software products.

The success of the support staff is measured by the efficiency of the information collection, storage, and retrieval system, and the degree to which it relieves the overall organization of unnecessary activities and waiting periods.

Figure 2-1 provides a high-level picture of the software process improvement organization and highlights activities and information flows among its three components.

The developers produce and maintain software but are not directly responsible for capturing the reusable experience. They provide the analysts with project and environment characteristics, development data, resource usage information, quality records, and process information. The
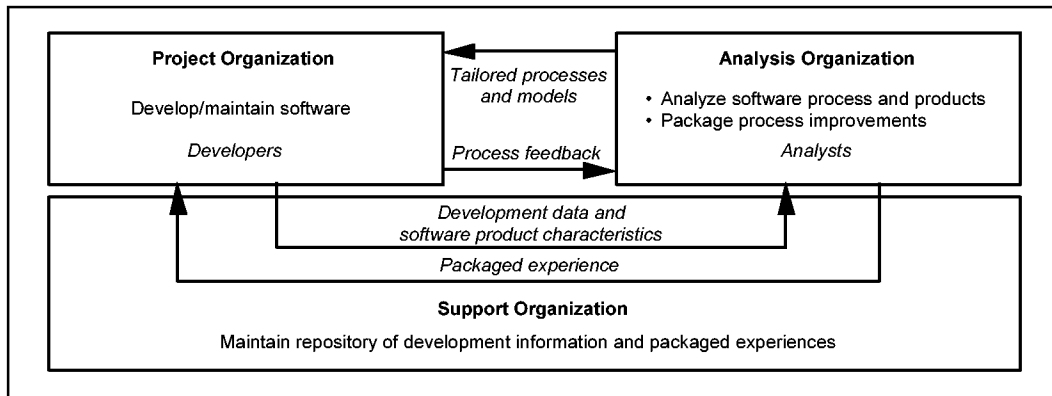
Figure 2-1. Software Process Improvement Organization

developers also provide feedback on the actual performance of the models produced by the analysts and used by the project. Therefore, with respect to software process improvement, the developers have the global responsibility for using, in the most effective way, the packaged experiences to deliver high-quality software.

The analysts, by processing the information received from the developers, produce models of products and processes and return direct feedback to each project. They also produce and provide baselines, tools, lessons learned, and data, parameterized in some form in order to be adapted to the characteristics of a project.

The support staff sustain and facilitate the interaction between developers and analysts by saving and maintaining the information, making it efficiently retrievable, and controlling and monitoring access to it. They use tools that assist in collecting, validating, and redistributing data and reusable experience.

The roles of the developers and support staff in software process improvement are easily understood. The role of the analysts is less clear; however, based on the information stated thus far, Table 2-2 summarizes what the role of the analysis organization is and is not.

Table 2-2. Role of the Analysis Organization

The analysis organization IS	*The analysis organization IS NOT*
An organization; it has people and structure	A quality assurance or independent verification and validation (IV&V) organization
A domain-specific infrastructure	
Separate from the developers but works closely with them	A research laboratory
	A management infrastructure
Variable in size (driven by the size of the development organization)	An audit organization

The ultimate goal of a software process improvement organization is to understand and repeat successes and to understand and avoid failures. Therefore, the software process improvement organization's processes and operations must be based on solid and objective development experience. Thus, a measurement-based approach is needed for project management, evaluation, and decision making. Software measures are applied to process, product, and resources. Measurement is one of the basic tools available to the software process improvement organization for performing its tasks and to management for controlling and improving the efficiency of the whole infrastructure.

2.2 The Software Process Improvement Approach

The goal of any NASA software process improvement program is continual process and product improvement. To attain this goal, the program uses a process approach consisting of three major phases: *Understanding*, *Assessing*, and *Packaging*. These phases are continually executed in any development environment within the organization. Figure 2-2 illustrates these three phases.
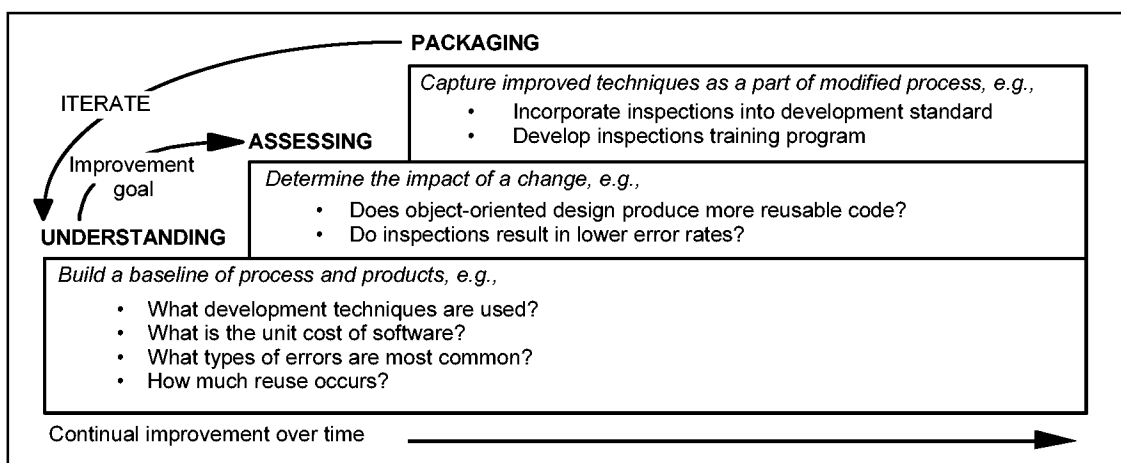
Figure 2-2. Three-Phase Approach to Software Process Improvement

The remainder of this section describes these three phases in more detail.

2.2.1 Phase 1 Understanding
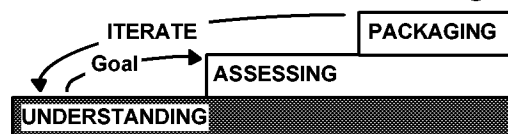
In the *Understanding Phase*, the organization's process and products are characterized and high-level goals for improvement are identified. The purpose of this phase is to continually capture the characteristics of the software process and products within the project organization and produce models, relationships, and general descriptions of the process and products. Understanding is the required starting point of the overall process improvement sequence, and it never ends, because changes must always be understood and characterized. Without this **baseline** of the process, products, and environment, no basis for change or improvement exists. A determination for change and improvement can be made and quantitative goals set only when the characteristics of the ongoing process and products are captured and understood.

Understanding is the phase in which the software baseline is established and is the most critical phase of the improvement approach; however, this phase is often ignored. The developers and analysts characterize the organizational environment, describing it in terms of relevant processes and models. The environment is characterized by using available data, both objective and subjective. The existing processes and products are characterized and modeled through measured experience. Based on the baseline findings, a specific organization can identify high-level goals for improvement (e.g., cut cost, improve reliability). Each organization must determine what types of improvement goals are most important in its local environment. Having a baseline allows the organization to set goals that are based on the need for specific self-improvements.

The baseline of the software organization is captured in the form of models (e.g., cost models, error models), relationships (e.g., relationship between testing time and error density), and characteristics (e.g., what standards are used, what techniques are used for performing specific activities). Although the type of information collected in the Understanding Phase is relatively generic and common across software organizations, specific characteristics that are derived from the particular goals and needs of the software organizations should always be considered. These specifics are product characteristics such as cost, size, and errors, and process characteristics such as effort distribution and resources usage. Understanding such environment-specific characteristics is necessary so that the organization can plan improvements in the context of local goals. For instance, if the organization's high-level goal is to improve productivity, it must understand (baseline) its current productivity rate and process and product characteristics. Using the baseline as the basis for improvement allows the organization to set specific, quantitative goals. For example, rather than striving to simply reduce the error rate, an organization can establish a more specific, measurable goal of reducing the error rate by 50 percent.

Figures 2-3 through 2-5 show some baseline information recently gathered for NASA as a whole (Reference 2). During the baseline period, NASA had developed more than 6 million source lines of code (MSLOC) and had over 160 MSLOC in operational usage. The baseline established that nearly 80 percent of NASA's software work is contracted to industry and educational institutions.

Figure 2-3 shows the distribution of NASA software domains for operational software. Mission ground support and general support software were found to be the largest and most prevalent software domains, accounting for almost 60 percent of all NASA software. Administrative/IRM software was the next largest domain, accounting for almost 20 percent of NASA software. The science analysis, research, and flight software domains were much smaller in size.

Figure 2-4 shows the amount of resources NASA invested in software. As indicated, more than 10 percent of NASA's workforce spent the majority of their time (i.e., more than half time) on software-related activities including software management, development, maintenance, quality assurance, and verification and validation. NASA invested a significant amount of manpower budgetary resources in software.
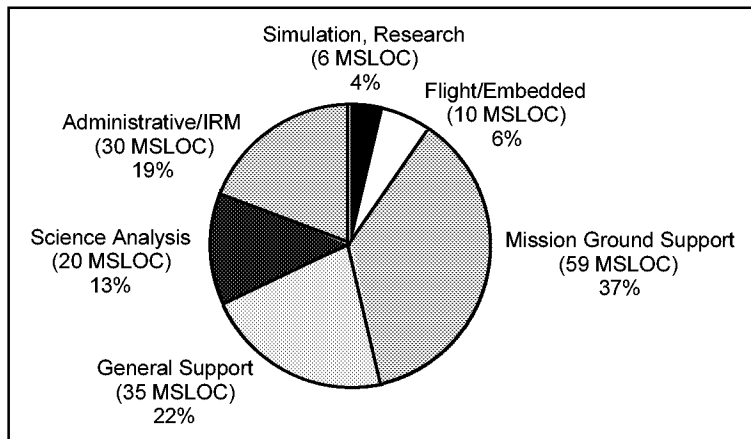
Figure 2-3. NASA Operational Software Domains
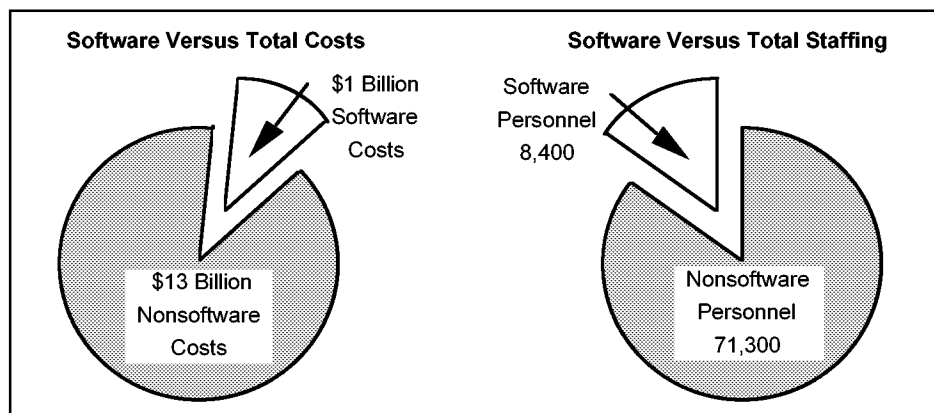
Figure 2-4. NASA Software Resources

Figure 2-5 shows the distribution of languages used for software in operations and under development. As shown, the use of FORTRAN, along with COBOL and other languages (e.g., Assembler, Pascal), has decreased significantly; presumably these languages are being replaced by C/C++. The use of both C/C++ and Ada has increased dramatically, though Ada use is not as widespread as C/C++. Nevertheless, FORTRAN development is still substantial, indicating its use will likely continue for some time.

For a global-level organization, such as NASA as a whole, the baseline is necessarily at a much more general level than in organizations locally implementing the software process improvement approach. Most models, for instance, would make sense only with respect to particular domains (e.g., flight software or administrative software), not for the Agency as a whole. Local models (e.g., cost and reliability models) can be developed for specific organizations to help engineer the process on ongoing and future projects.

NASA-GB-001-95
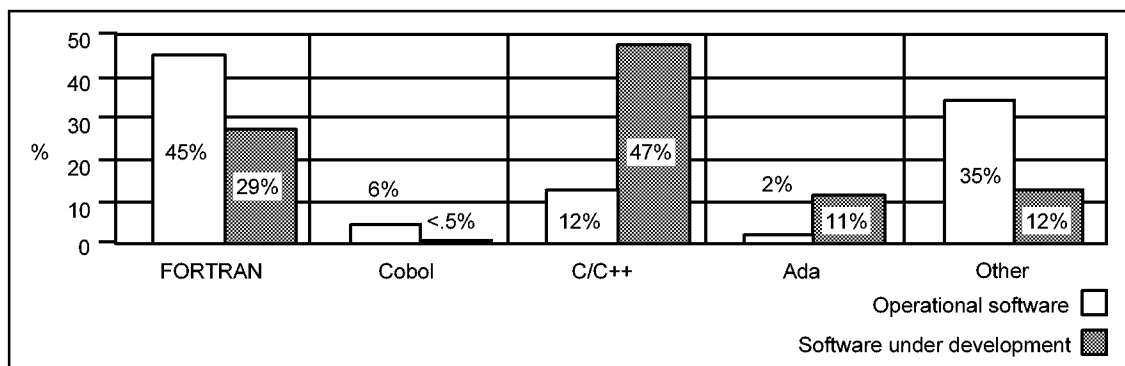
Figure 2-5. NASA Language Preferences and Trends

The following examples illustrate the Understanding Phase at a local organizational level. The examples use data from a major software organization at NASA that has continually collected and studied development data for general support systems. The data represent over 500 staff-years of effort and are from over 25 systems completed in the mid- and late- 1980s. These data were extracted and analyzed to build the basic understanding parameters including two of the most basic, yet often overlooked, characteristics of software: effort distribution and error profiles.

By collecting data readily available during the development process, the organization gained understanding of where the development effort for the software process was being expended—among design, coding, testing, and other activities (such as training, meetings, etc.). Although extremely easy to obtain, such basic information is often ignored.

Example 1: Understanding Effort Distribution

Figure 2-6 shows the distribution of effort by time. Typically, 26 percent of the total effort is spent in the design phase, that is, the period from onset of requirements analysis through critical design review (CDR); 37 percent in the code phase, that is, from CDR through code completion and unit testing; and the remaining 37 percent in the test phase, that is, integration, system, and acceptance testing.

Viewing effort distribution from a different perspective, Figure 2-7 breaks down specific development activities, showing the amount of time attributed to each as reported by the individual programmers rather than in a date-dependent manner. Throughout the project, programmers report hours spent in these categories. The analysts examine the developer-supplied information across many projects and then determine the typical effort distribution for this particular organization. As this figure shows, 23 percent of the developers' total effort is spent in design; 21 percent in code; 30 percent in test; and 26 percent in other activities including training, meetings, documentation (e.g., system descriptions and user's guides), and management. Such basic information can then be used to generate local models, such as a "cost by activity" model.

Date Dependent

Test 37%

Design 26%

Code 37%
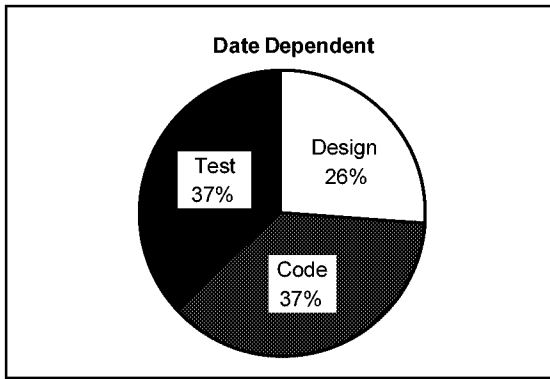
Figure 2-6. Effort Distribution by Time

Programmer Reporting

Other 26%

Design 23%

Code 21%

Test 30%
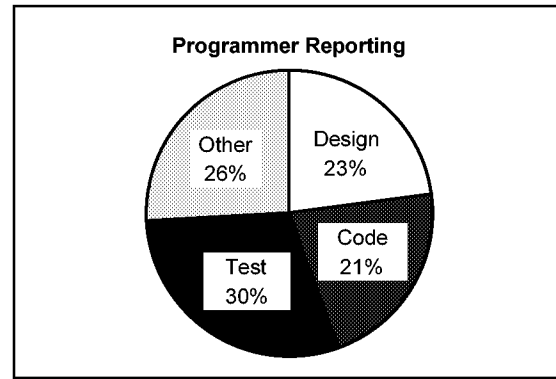
Figure 2-7. Effort Distribution by Activity

Example 2: Understanding Error Profiles

A second example of the characterization performed in the Understanding Phase is found in error characteristics. Based on the same projects for NASA ground systems, error profiles based on over 500 reported development errors are depicted in Figures 2-8 and 2-9. These data provide some initial insight into the error profiles, which in turn can lead to a more structured approach to addressing certain error characteristics in future systems.

Figure 2-8 shows the breakdown of all errors by class. This figure shows what types of errors exist and how they are distributed across classes, as classes are defined by the specific organization.

Figure 2-9 depicts how the software errors found in an environment are distributed into different classes based on their recognized origin. In this example, 50 percent of errors originate from requirements, 20 percent from design, 20 percent from coding, and 10 percent from clerical sources. The overall error rate for this organization was six errors per thousand source lines of code (KSLOC).

Computational

Initialization

13% 15%

Data 24% 30%

18% Logic/ Control

Interface
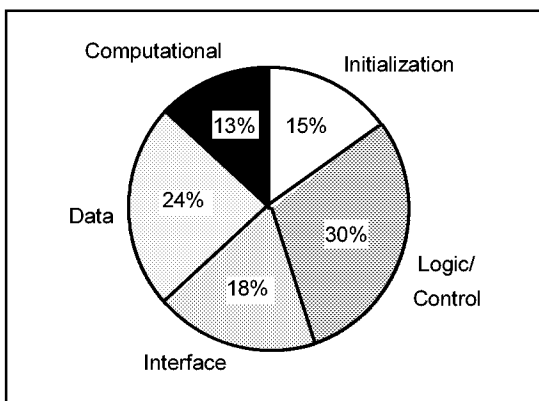
Figure 2-8. Error Distribution by Class

Implementation

Require- ments 20%

20% Design

50% 10%

Clerical
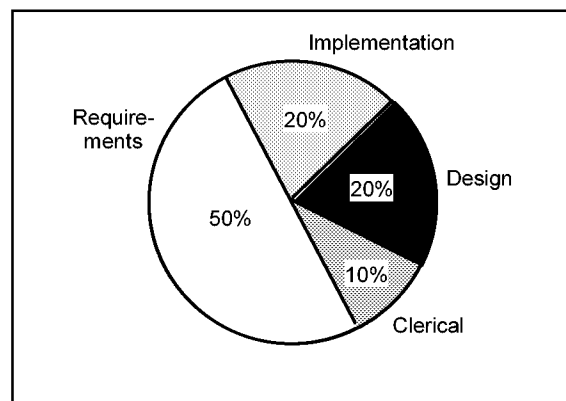
Figure 2-9. Error Distribution by Origin

Basic information about errors can lead to the development of error-related models, such as error detection rate. Figure 2-10 shows the error detection rate for five projects of similar complexity in the same environment. This organization typically sees the error rate cut in half each time the system progresses to the next life-cycle phase. This type of information leads to the important step of producing models that can be used on ensuing projects to better predict and manage the software quality within the different phases.
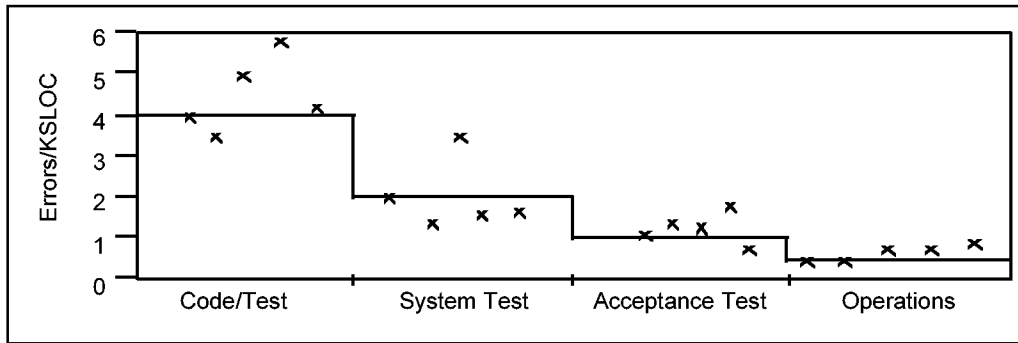
Figure 2-10. Error Detection Rate

Example 3: Understanding Other Models and Relationships

Figure 2-11 provides examples of other models and relationships that have been developed as useful profiles of a specific environment. Reference 3 discusses these and other relationships and how they can be applied.

Effort (in staff-months)	=	$1.48 * (KSLOC)^{0.98}$
Duration (in months)	=	$4.6 * (KSLOC)^{0.26}$
Pages of Documentation	=	$34.7 * (KSLOC)^{0.93}$
Annual Maintenance Cost	=	$0.12 * (Development\ Cost)$
Average Staff Size	=	$0.24 * (Effort)^{0.73}$

Figure 2-11. Sample Process Relationships

These examples are typical products of the characterization activity performed in the Understanding Phase. They are based on actual data collected in a production environment and represent the situation before (or at a specific milestone of) an improvement initiative. The examples are domain-specific and, although essential for understanding a specific environment and controlling improvement, they are not universally applicable to other domains. The concept of building the understanding (baseline) is applicable to all domains; however, the specific models produced and organizational goals set may not be.

2.2.2 Phase 2 Assessing
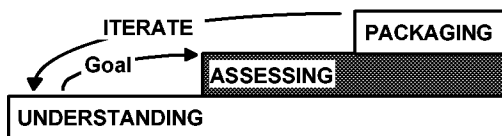
In the *Assessing Phase*[1], specific objectives for improvement are set, one or more changes are introduced into the current process, and the changes are then analyzed to assess their impact on both product and process. Change may include introducing a method, tool, or management approach. This phase generally is thought of as the experimental step in which some defined change to the

---

[1] Within the context of process improvement, the term "assessing" refers to an evaluation of the effect of introducing a change. It should not be confused with the usage of the term assessment as regards CMM organizational process capability evaluations.

process is evaluated against the baseline. The changes are studied through specific experiments conducted in selected projects.

Experimentation is defined as the steps taken to set objectives based on a project's software improvement goals for the specific assessment; introduce some change to the baseline process; collect detailed information as the changed process (i.e., the experiment) progresses; compare the results of the experiment against the baseline values established in the Understanding Phase; and, finally, determine if the change that was introduced met the objectives established for the assessment/experiment.

The choices made in this phase are driven by the characteristics and high-level organizational goals identified in the Understanding Phase. Experiment objectives are set and quantified according to the data collected in the baseline. Reference 4 provides information on a mechanism [the Goal/Question/Metric (GQM) paradigm] for defining and evaluating a set of operational goals using measurement on a specific project. The questions addressed in the Assessing Phase depend on both the overall goal and the chosen process change. For instance, if a cycle time of 2 years per system has been measured over the last 10 years, a possible goal might be to decrease the cycle time to 18 months over the course of the next several years. Another organization may wish to focus a specific experiment on improving reliability (i.e., reducing error rates) and might introduce software inspections to attain that goal. The inspections' assessment would answer the question "Does the use of software inspections result in the production of more reliable software?"

To carry out the assessment process, several basic requirements must be fulfilled. First, some kind of baseline or norm must exist against which the assessment data can be compared. This baseline is provided by the Understanding Phase. In some cases, however, further information is needed to provide a reasonable comparison base for the Assessing Phase, for example, experience level of the developers or level of receptiveness to change. Second, the organization must prioritize changes and select the ones on which it will focus. Although any organization would certainly aspire to attain all improvement goals at once, the assessment of changes through experimentation proceeds slowly. The effect of too many concurrent changes cannot be controlled, and the whole assessment would be compromised. Additional complexities exist within the Assessing Phase because essentially all measures of change are coupled to other measures. For instance, to improve reliability (i.e., reduce error rates), cost might be added to the development process. Third, the organization must state its goals (e.g., fewer errors, higher productivity) in a measurable way (e.g., the number of errors per KSLOC should be less than 4.0; the number of statements developed and tested per hour should be higher than 3.5). Although achieving the measured goal is a good indicator of success, the subjective experience of the developers must also be considered in the assessment.

Examples 4 and 5 illustrate specific assessments performed within NASA.

Example 4: Assessing Inspections

Formal inspections are technical reviews that focus on detecting and removing software defects as early in the development life cycle as possible. The goal of introducing inspections in one NASA organization, JPL, was to increase the quality of the following products of software systems: software requirements, architectural design, detailed design, source code, test plans, and test procedures. By doing so, the overall quality of the

software would be improved and cost would be reduced by detecting defects early (Reference 5). In the Understanding Phase, the cost to detect and fix a defect found in formal testing was determined to be between 5 and 17 hours (the defect has to be traced, found, fixed, and retested). In the Assessing Phase, inspections were introduced and their impact measured. Because inspections were introduced early in the development life cycle when most products are technical documents, the reporting metric was number of pages rather than estimated lines of code. As Figure 2-12 shows, a higher density of defects was detected in earlier life-cycle products than in later ones. Not only was the overall software quality improved by reducing rework during testing and maintenance, but costs were reduced by finding defects earlier in the life cycle. On average, finding and fixing defects found during inspections took 1.1 hours and 0.5 hours, respectively—a significant savings compared to the previous range of 5 to 17 hours for both. Inspections have subsequently been introduced at other NASA centers, and their assessments have also been favorable.

The importance of the Understanding Phase cannot be overemphasized. In this example, without the established baseline, the assessment would have been purely subjective, relying solely on opinions of people within the organization as to whether inspections helped, did nothing, or perhaps even hindered the development process. Changes introduced by the software organization to address some goal of improvement will always have multiple impacts that must be considered, such as the added cost and overhead of making a change. In this example, several measures, such as additional cost, development time, and final defect rates, must be analyzed to ensure that the full impact of the change is understood.

Not all assessments will be positive. The assessment results might show positive impact, no change, or even negative impact on the factors being examined.
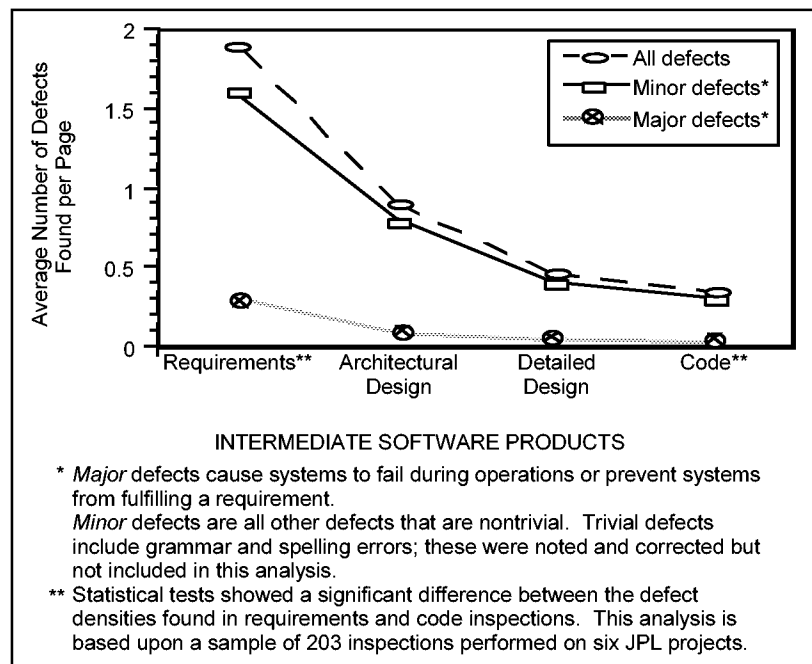
Figure 2-12. Assessing the Impact of Inspections

Example 5: Assessing Cleanroom

Another sample assessment, the use of the Cleanroom process (Reference 6), is also provided. Cleanroom is a software process developed by Harlan Mills (International Business Machines) that focuses on producing error-free software and results in a product with certifiable reliability. The Cleanroom process was selected to attain the goal of improving the reliability of delivered software without penalty to the overall development cost. Significant process changes included using formal code inspections, applying the formal design concept of box structures, using rigorous testing approaches driven by statistical methods, and providing extended training in software engineering disciplines such as design by abstraction.

In 1987, the first Cleanroom project was selected, the team trained, the experiment plan written, and the development process and product meticulously measured. Process impacts were observed at several levels, including increased effort spent in design and a different coding activity profile. Figure 2-13 illustrates these impacts.

This first experiment (Reference 7) resulted in impressive product gains in both reliability (38 percent) and productivity (54 percent) when compared with existing baselines (Figure 2-14). However, because this first project was small [40,000 developed lines of code (DLOC)], two additional projects were selected using a refined set of Cleanroom processes derived from the first project's experiences (Reference 8). These later projects provided additional evidence that components of the Cleanroom process were effective in reducing error rates while maintaining productivity for smaller projects, but the larger project had a smaller reliability improvement (14 percent) with a 23 percent reduction in productivity.
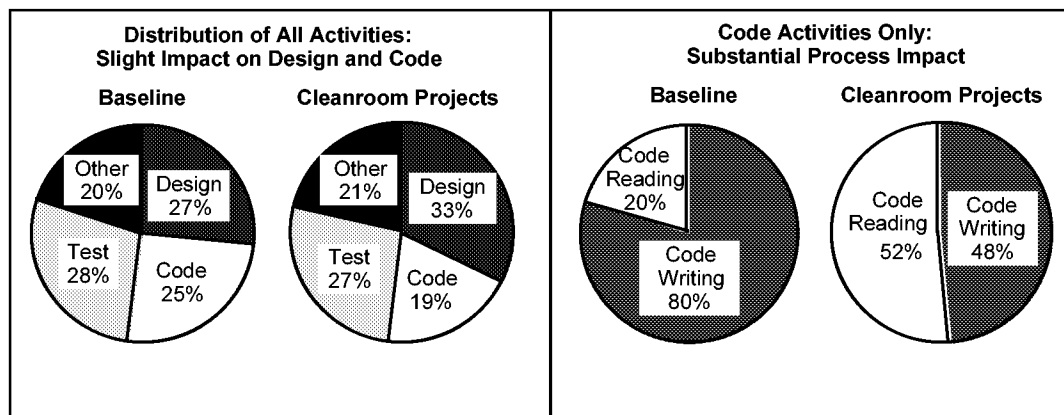
Figure 2-13. Assessing the Impact of Cleanroom on Process

Figure 2-14 illustrates the impact of the Cleanroom process on the product. As a result, key Cleanroom concepts, such as focused inspections and process training, have been infused into the standard organizational process, but other aspects are undergoing further analysis until the cost differences are more fully explained.
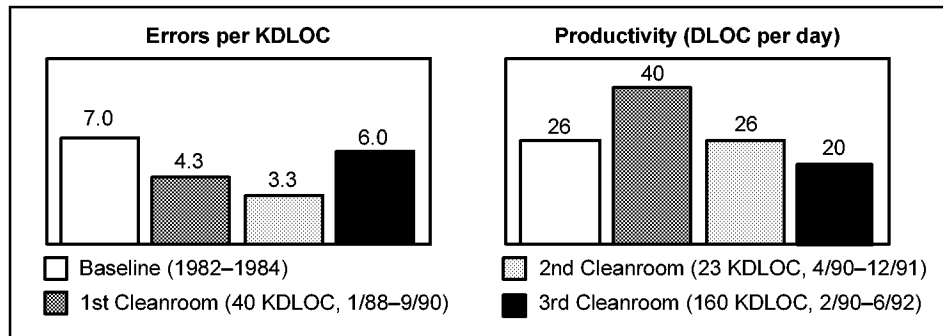
Figure 2-14. Assessing the Impact of Cleanroom on Product

Again, even though the change might result in the achievement of the original goal, other parameters must be analyzed to ensure that the full impact of the change is understood. If the assessment of Cleanroom showed that the goal of improved reliability was met but other factors suffered significantly (e.g., productivity drastically decreased and cost increased), the overall assessment might not have been favorable. It might be totally acceptable to one organization to increase cost significantly to achieve improved reliability; for another, the same circumstances might be unacceptable. It is important to understand the full impact of any change within the constraints of the specific organization.

Many other assessments have been performed within NASA. References 9 through 12 detail other sample assessments involving Ada, object-oriented technology (OOT), software modularization, and testing techniques, respectively.

Assessing is the second step of the improvement paradigm. Whether results are favorable or unfavorable, each assessment must be followed by the third step, some form of packaging. The next section describes the Packaging Phase.

2.2.3 Phase 3   Packaging
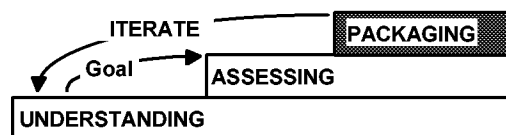
In the *Packaging Phase*, changes that have produced satisfactory results and shown measurable improvement are institutionalized and incorporated into the mainstream of the organization. During this phase, the analysts develop new models, standards, and training materials based on what has been learned during the Assessing Phase. The products developed by the analysts are stored by the support staff into a repository (i.e., an experience base) and are provided to the developers upon request. Packaging typically includes standards, policies, and handbooks; training; and tools. For methodologies or techniques that do not show any favorable impact during the Assessing Phase, results must still be captured and archived (i.e., packaged) so the corporate memory is continually enhanced. This packaging may include reports or papers that are maintained in the corporate repository. The results of the packaging phase are fed back to those individuals involved with baselining prior to the next related project or experiment. Thus a particular technology can be assessed through multiple experiments, each one building on the packaged results of the previous experiment(s).

Packaging requires a clear understanding of the impact of a specific technology on the software process and products. The ultimate measure of success is, in general, an improved software

product. Therefore, institutionalization of change must be substantiated by reliable data on the products resulting from the process. Standards, policies, process characteristics, and other "packages" are most effective when they reflect empirically derived evaluations of technologies and processes that are suitable and beneficial to the specific organization.

The major product of the packaging step is the organization's standards, policies, and training program. The software process must be developed to respond to the general needs of the organization and is driven primarily by the experiences and needs of the developers. Thus, every element of the standards need not have been assessed, but some rationale must exist for their inclusion.

Examples 6 and 7 illustrate how the experiences from the previous assessments (Examples 4 and 5) may be (or have been) packaged.

Example 6: Packaging Experiences With Inspections

Consider the example of inspections. In the Understanding Phase, the organization characterized the cost to detect and fix defects. In the Assessing Phase, inspections were introduced to increase software quality and to reduce cost by detecting defects early in the life cycle. The organization then assessed the impact of inspections and determined that the goals of the experiment had been achieved. In the Packaging Phase, the organization needs to determine how to package its favorable experience with inspections, perhaps by modifying its standards (e.g., development manual) to include inspections or to train its personnel to effectively use inspections. Figure 2-15 depicts the example of inspections with respect to the three-phase process improvement approach. Process changes occur, assessments are made, and improvements are identified. Organizational standards then need to be upgraded to reflect the improved process as part of the standard way of doing business within the organization.
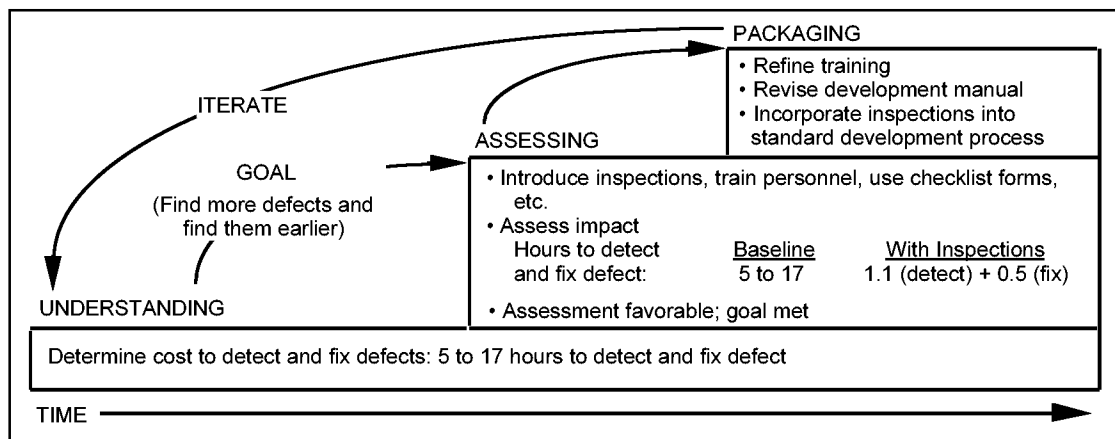
Figure 2-15. Packaging Experiences With Inspections

Example 7: Packaging Experiences With Cleanroom

Finally, consider the Cleanroom example. The goal of introducing Cleanroom was to improve reliability without incurring a cost penalty. In the Understanding Phase, cost and reliability rates were determined. In the Assessing Phase, the first Cleanroom experiment was performed, and impressive gains were achieved in both reliability and productivity. Experiences of this first experiment were packaged in the form of updated training

materials and a handbook detailing refined Cleanroom processes (Reference 13); these products were then used on subsequent Cleanroom experiments.

The three steps of the improvement approach were repeated for additional Cleanroom experiments. In the Understanding Phase, error and productivity rates from the organizational baseline and the early Cleanroom experiment were established for use in later comparisons. In the Assessing Phase, the results of the later experiments were evaluated against both the baseline and early Cleanroom experiences. Experiences from these later experiments were also incorporated into the tailored Cleanroom process handbook and training materials. Some key Cleanroom concepts, such as focused inspections and process training, have been packaged and infused into the standard organizational process. Other aspects of Cleanroom are undergoing further analysis until the cost differences exhibited in the larger project can be more fully explained. Figure 2-16 depicts the packaging of the experiences of the Cleanroom experiments with respect to the process improvement approach.
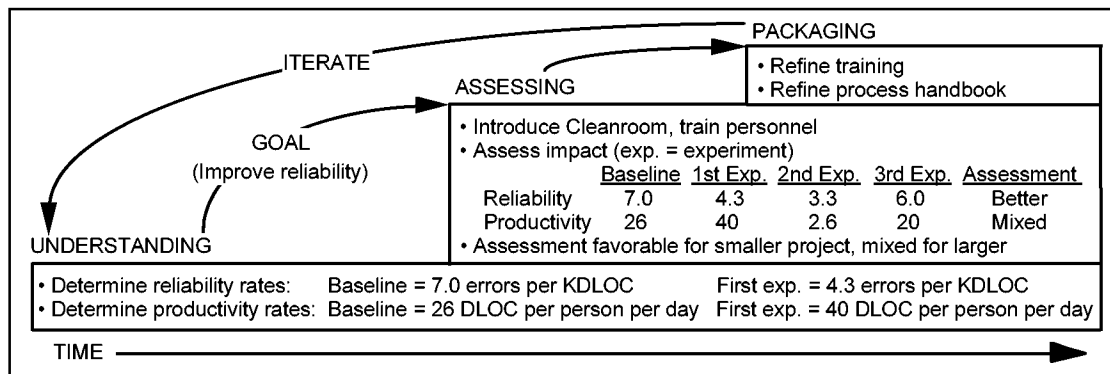
Figure 2-16. Packaging Experiences With Cleanroom

Even if later assessments of Cleanroom are favorable, the process change will not be mandated immediately to every project. A significant change like Cleanroom would be evolutionary, and additional projects would be identified as the experience base is broadened. Experience has shown that significant process changes cannot be adopted or mandated quickly; they must evolve. It is the task of the analysts and development managers to jointly plan the evolutionary process change for changes as significant as Cleanroom.

2.3 Domains for Software Process Improvement

This chapter has discussed the structure and three-phase approach needed for software process improvement. The concept of software domain is also critical to process improvement. The organization must know the scope to which process improvement is being applied. Is process improvement being applied across the entire organization or to a specific subset such as a division or department? Understanding domains is also important to facilitate sharing of experiences within and across domains.

The software process improvement organization gathers, synthesizes, and packages experiences from a particular domain for use within the same domain. Whatever the domain is, the

organization first develops an *understanding* of its processes and products. It then treats the domain as the "whole world" for applying process change and *assessing* the impact of this change. The results and lessons are then *packaged,* and the information is applied to ongoing and subsequent efforts within that specific domain to improve the quality of the software being produced. Transfer of information across domains may be achieved, especially when the domains have similar characteristics. However, the primary goal is to improve software within the specific domain.

Domain analysis, or domain definition, is usually described as the process of recognizing standard concepts, functionalities, and architectural characteristics within a software development application area. Domain definition is important to facilitate product reuse and improve both the productivity and quality of the final products (Reference 14). Often domain analysis pertains strictly to the reuse of code. For software process improvement, domain analysis is not limited to any one type of artifact (i.e., code); it facilitates the reuse of all experience, including that embodied in code. For software process improvement, the domain influences the breadth to which analysis applies; it influences the scope of packaged results (e.g., standards, policies, training, tools); and it strongly determines to what extent any information, experience, or data can be shared and reused. Additionally, it dictates how large a software process improvement organization can be.

No direct mechanism exists for defining domains or organizational characteristics. Historically, the definition of a software domain has most often been associated with an organizational structure: a single domain is either an entire organization (e.g., NASA) or a subset of an existing organization (e.g., NASA field center, branch, division, project, department). Domains are not, however, necessarily linked to organizational structures. They may be defined according to the factors that characterize the development processes, technologies, products, constraints, goals, and risks associated with the projects. Different classes of projects may exist within one organization (e.g., real time versus non-real time, flight versus ground systems) that might be treated as individual domains or together as a single domain. The development processes used and the overall process improvement structure are most often defined by the management structure that is in place.

Once domains are understood and identified, common processes, standards, and experience may be shared with confidence by various software organizations (e.g., individual NASA projects or field centers) within a broader organizational structure (e.g., the Agency as a whole). Because organizations can share data, information, and lessons learned, they can improve faster and further than they could in isolation.

Domains have no size or breadth limitations. For example, there is some commonality for "all of NASA" to be considered one domain; there is more commonality for "all of Johnson Space Center" to be considered one domain; and there are some differences between Johnson Space Center and Langley Research Center potentially resulting in their being considered different domains. The point is that any organization can be classified as one domain, but as the organization becomes broken down into more specific and smaller elements, more parameters are relevant to the more specific, smaller domains than the larger organizational domain. Figure 2-17 depicts some potential domains within NASA. Some domains might be tied to organizational structures (e.g., individual field centers) while others might pertain to application domains (e.g., mission support software). Within any domain is the potential for subdomains to exist. For

example, the mission support software domain might be broken down further into ground support software, flight software, and scientific software.

Reference 14 describes activities to date in analyzing and defining NASA domains for the reuse and sharing of experience.

The baseline has been established (Phase 1—Understanding), change has been introduced and improvements identified (Phase 2—Assessing), and the experiences have been packaged in a reusable manner (Phase 3—Packaging). Now the experiences have to be shared and incorporated throughout the organization. These experiences can be shared not only within specific domains, but occasionally even across domains. But with whom? Determining the domain-specific characteristics is necessary to identify who can share the information and experiences. For example, standards are one form of packaged experiences. If standards are to be adopted across domains, it is important to understand the characteristics those domains have in common. Domain-specific characteristics need to be identified to tailor standards for the needs of a particular domain. Like any form of packaged experiences, standards should evolve over time. To what domains do the standards apply? Answering this question extracts current information from the domains so that standards can be updated to reflect recent experiences.

DOMAIN: *NASA*		
DOMAIN: *Mission Support*	DOMAIN: *Administrative*	DOMAIN: *Individual Field Center*
DOMAIN: *Ground Support* [e.g., Deep Space Network (DSN), SEL]	DOMAIN: *Information Resources Management (IRM)*	DOMAIN: *Goddard Space Flight Center*
DOMAIN: *Flight* (e.g., Space Station, Shuttle flight software)	DOMAIN: (etc.)	DOMAIN: *Jet Propulsion Laboratory*
DOMAIN: *Scientific* [e.g., Earth Observing System (EOS)]		DOMAIN: *Johnson Space Center*
DOMAIN: (etc.)		DOMAIN: *Langley Research Center*
		DOMAIN: *Marshall Space Flight Center*
		DOMAIN: (etc.)

Figure 2-17. Examples of Potential Domains Within NASA

Understanding domains and domain-specific and domain-independent characteristics is important for sharing information and experiences. Without defining domains and understanding the characteristics that make them similar to and different from others, the full benefits of sharing experiences cannot be achieved.

2.4   Comparison of Two Software Process Improvement Approaches

This section addresses the unique aspects of NASA's software process improvement approach by comparing it with another particular approach, the Software Engineering Institute's (SEI's) Capability Maturity Model (CMM) (Reference 15). Both approaches share the underlying principle of continual, sustained software process improvement.

As discussed earlier, the NASA software process improvement framework consists of two elements:

- An organizational structure (Section 2.1) consisting of developers, analysts, and a support staff

- A three-phase approach to process improvement (Section 2.2, Figure 2-2), that is, the continual understanding, assessing, and packaging of organizational experiences

These two elements allow an organization to continually improve the quality of software products and processes within a specific domain.

The key points of NASA's process improvement approach are that

- Process improvement is driven by internal goals and local experiences.

- Each domain is dealt with in a different way according to its specificity.

- The environment is characterized according to organization-dependent measures.

- No assumptions are made about best practices in the process area.

- The ultimate measure of success is the improvement of the product or service delivered by the organization.

The CMM is a widely accepted benchmark for software process excellence. It provides a framework for grouping key software practices into five levels of maturity. A maturity level is an evolutionary plateau on the path toward becoming a mature software organization. The five-level model provides a defined sequence of steps for gradual improvement and prioritizes the actions for improving software practice.

Within the CMM, an organization strives to mature to a continually improving process. To do so, the organization must advance through the following maturity levels defined by the SEI:

Level 1, Initial. The software process is characterized as ad hoc and, occasionally, even chaotic. Few processes are defined, and success depends on the efforts of individuals.

Level 2, Repeatable. Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier success in projects with similar applications.

Level 3, Defined. The software process for both management and engineering activities is documented, standardized, and integrated into an organization-wide software process. All projects use a documented and approved version of the organization's process for developing and maintaining software.

Level 4, Managed. Detailed measures of the software process and product quality are collected. Both the process and products are quantitatively understood and controlled using detailed measures.

Level 5, Optimizing. Continual process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technologies.

Figure 2-18 depicts the CMM process improvement paradigm; the NASA software process improvement approach is also repeated in that figure.

Differences between the CMM and the three-phase NASA improvement approach are described in four areas: the goals, initial baseline, initial analysis, and improvement approach.

1. **Goals.** Each organization must set goals for what is to be improved.

   CMM: A generalized, domain-independent goal focuses on process. Every organization strives to improve the software process and, ultimately, evolve to a continually improving, optimizing process (Maturity Level 5). Organization A and Organization B both try to improve their processes and become Level 5 organizations. In progressing to higher levels, organizations expect to reduce risk and generate better products.

   NASA: Organizations focus on improving products. Specific goals, however, vary from organization to organization. Organization A may attempt to improve reliability by decreasing error rates. Organization B may strive to decrease the development cycle time. Goals are domain dependent. Within the framework of the CMM, organizations using the NASA approach may progress to higher maturity levels and eventually become a Level 5.

   The CMM goal is domain independent and generalized. The CMM focuses on improving the software process. NASA goals vary from organization to organization, that is, they are domain dependent. The underlying goal of the NASA approach, however, is to improve the software product.
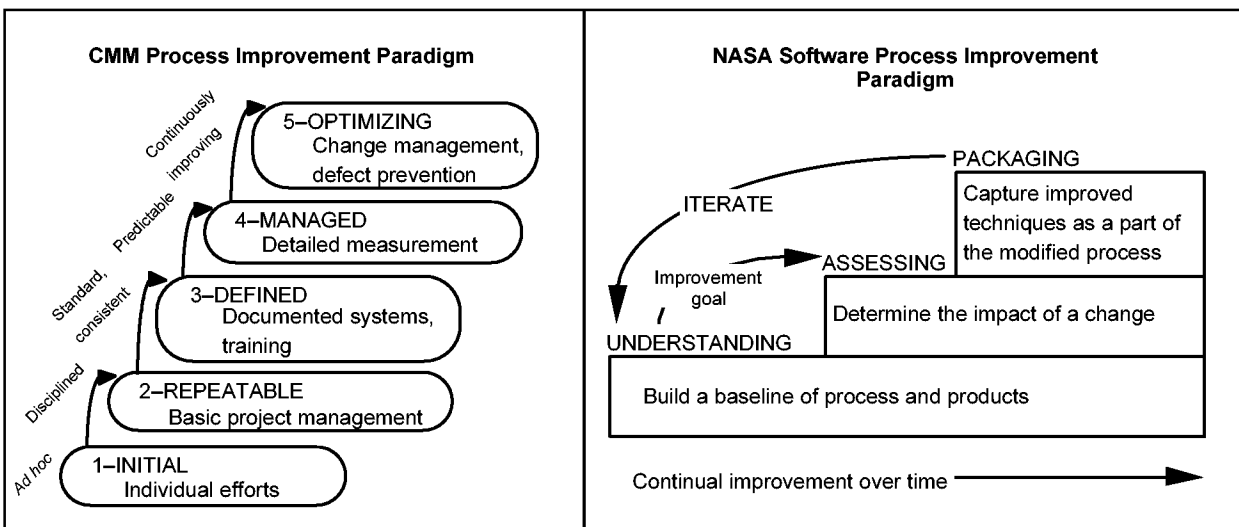
Figure 2-18. CMM and NASA Software Process Improvement Paradigms

2. **Initial Baseline.** Each organization must establish a basic understanding (baseline) of its current software product and process.

   CMM: Baselining is achieved by performing an assessment of the organization's process. This assessment is made against well-established criteria, and the organization is baselined at a certain maturity level. These criteria enable comparisons across domains because each organization is assessed against the same criteria. The same elements are examined for every organization: e.g., Does it have good standards? What

is its training program like? How is its measurement program conducted? Based on the examination of these criteria, the organization is baselined at some maturity level.

NASA: Baselining involves understanding the process and product of each individual organization. This baseline is domain dependent. Unlike the CMM, no common yardstick exists enabling comparison across domains. Some factors need to be characterized (baselined) by all organizations, such as how much software exists, what process is followed, what standards are used, what is the distribution of effort across life-cycle phases. Other factors of interest depend on the goals of the organization. Organization A, for example, would want to baseline its error rates, whereas Organization B needs to determine its development cycle time.

The CMM baseline is process based and established against a common yardstick. The NASA baseline is domain dependent and is both process and product based.

3. *Initial Analysis.* Changes are introduced to make some improvement. An analysis (assessment) of the change must be made to determine if improvement has occurred.

CMM: Assessment of change is accomplished by reassessing the process. An organization is baselined at one level, makes changes to try to attain a higher level, and is then reassessed to determine if it has progressed to another level. Success is measured by process change. The ultimate success is changing the process until it continually improves. The organization then achieves the highest maturity level rating, a Level 5. The measure of success is domain independent, because all organizations are measured against the same criteria (i.e., "a common yardstick").

NASA: Assessment of change is domain dependent. An improvement goal is set, change to the process made, change to the process and product examined and verified, and the effect of change evaluated against the original goal. Success is measured by product improvement and is determined based on the goals of the individual organization. The organization attempting to improve its reliability would institute a change, such as the Cleanroom process, to try to reduce its error rates. It would then assess the result of the experiment based on its original goals.

CMM analyses and assessments are based on its common yardstick. NASA analyses and assessments are domain dependent, and are based on goals defined by the individual organization.

4. *Improvement Approach.* Approaches to improvement are defined and driven by different factors.

CMM: Changes made to the organization's process are driven by the CMM common yardstick. If an organization is baselined at some level, it will change elements necessary to get to the next maturity level. If an improved measurement program is needed to advance to another maturity level, the organization will focus on changing its measurement program to meet the CMM's criteria. The improvement approach is solely process based. The CMM's common yardstick enables a common roadmap toward continual improvement.

NASA: Organizational experiences and goals drive change. Changes to the process are made in an attempt to improve the product. Each domain must identify the most appropriate process changes to achieve its product goals.

The CMM s common yardstick drives change; its improvement approach is process based. NASA s organizational experiences and goals drive change; its improvement approach is product based.

Table 2-3 summarizes the differences between these two process improvement approaches.

Despite their differences, both approaches suggest that every software organization should deploy a program for the continual, sustained improvement of the overall quality of its products and processes. The main difference is that, whereas the typical process improvement programs are based on the assumption that improvements to the software process maturity will eventually elicit improvements to the product quality, NASA's software process improvement approach ultimately focuses on improvement in the product and service quality, although achieved through process improvements.

Table 2-3. The NASA Software Process Improvement Approach Versus the CMM

Area	NASA Approach	CMM Approach
Goals	Focus on improving product Goals vary across organizations Domain dependent Success = better product; specific measures of success vary from organization to organization	Focus on improving process Generalized goal (improve process, get to Level 5) Domain independent Success = better process and higher level; common measure of success
Initial Baseline	Understand process and product	Perform assessment of process Common yardstick is basis for initial baseline (what is the maturity level?)
Initial Analysis	Change process to improve product, reassess process and product Organization specific, no way to compare across organizations	Change process to advance to a higher level, reassess process (what is the maturity level now?) Can compare across organizations
Improvement Approach	Product based Organizational experience and goals drive change	Process based Common yardstick drives change

The NASA approach assumes that every development organization must first understand its process, products, software characteristics, and goals before selecting the set of changes that are meant to support software process improvement. The underlying principle is that "not all software is the same." An organization must understand its software business before determining that change must be made, and any change must be driven and guided by experience, not by a set of generalized practices. There indeed may be generalized process concepts, but the essentials of any process guiding development and process change must be driven by the knowledge of the development organization.

NASA-GB-001-95

The CMM can be viewed as a top-down approach with a generalized set of practices, whereas the NASA approach is bottom-up with practices specific to individual organizations depending on their product improvement goals. Neither approach can be effective if used in isolation. The CMM approach requires awareness of the product changes, and the NASA approach requires use of some model for selecting the process changes aimed at improving product characteristics. Both the top-down and bottom-up approaches play an important role in the goal of improving the software business. For NASA, the CMM defines an excellent model for assessing process and for selecting potential process changes that can support the goal of sustained improvement.

The CMM approach is designed as a framework that organizations may use to better understand their software process and to provide guidance toward lower risk in the way software is developed. It provides an excellent procedure for identifying potentially beneficial additions to the organization's software business practices. NASA capitalizes on this approach to guide efforts at characterizing the way software is developed and in what areas NASA may look for improvements in consistence and commonality. By complementing the CMM with specific approaches to assessing goals, products, and product attributes, a complete and effective program is defined.

Each of the approaches poses similar difficulties in defining exactly the scope or size of the local organization, but some judgment must be applied to determine what this single entity can be. The smaller the organization, the more detailed the process definition, as well as the process improvement definition, can be.

NASA-GB-001-95

Chapter 3. Structure and Operation of the Software Process Improvement Organization

---

Chapter Highlights
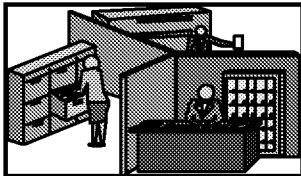
COMPONENTS
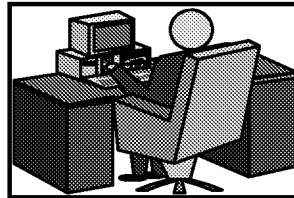
Developers
Analysts
Support Staff

DEVELOPERS

Produce software
Provide data
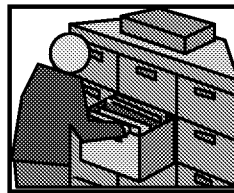Participate in studies
Use experience packages

ANALYSTS

Design studies
Analyze project data
Package results

SUPPORT STAFF

Process data
Maintain repository

This chapter takes a closer look at the structure of the software process improvement organization, its individual components, the resources they require, and their functions and presents details regarding the operation of the software process improvement program. It assumes the structure is already in place.

The chapter then describes the responsibilities, activities, and interaction of the developers, analysts, and support staff, detailing how each of these groups performs the activities associated with its process improvement responsibilities and how each operates on a daily basis for each of the following time periods associated with development projects: before or at project start, during the project, at or after project completion.

Many of the process-improvement-related responsibilities of these three organizational elements are associated with software measurement. Some details on measurement-related activities will be presented in this chapter; additional guidance on establishing, using, and maintaining a software measurement program can be found in Reference 16.

3.1   Components of the NASA Software Process Improvement Organization

Software Process Improvement Organization is a designation that refers to the whole organizational infrastructure whose components are the

- *Developers,* consisting of the developers and maintainers, whose primary objective is to produce software on time and within budget. Additionally, they must provide development information to the analysts. They receive experience packages from the analysts (e.g., standards, models) and reuse these packages in their activities.

- *Analysts,* whose focus and priority are to support project development by analyzing experience drawn from people, process, documents, and tools. They synthesize and package this information in the form of policies, standards, training materials, and, in general, models of the product and of the process (both formal and informal).

- *Support staff,* who serve as the focal point for all the archived information produced and used within the software process improvement organization. Additionally, this component validates and qualifies the data and the other information, making sure that the organization's information repository conforms to the needs of the analysts and developers.

Figure 3-1 shows the three components of the software process improvement organization and highlights some activities they perform.

The analysts and support staff exist solely because of the software process improvement activities; therefore, all their activities are related to software process improvement. However, process improvement activities are only a portion of the developers' responsibilities. Some of these activities are already part of the operation of the developers in a traditional environment, but some new activities have been added and some old ones changed. Table 3-1 presents a synopsis of the development activities pertaining to software process improvement, highlighting what has changed and what remains the same. The remainder of this chapter addresses only those activities of the developers associated with software process improvement. It does not discuss

any of the regular activities associated with developing software unless they are relevant to the process improvement activities.
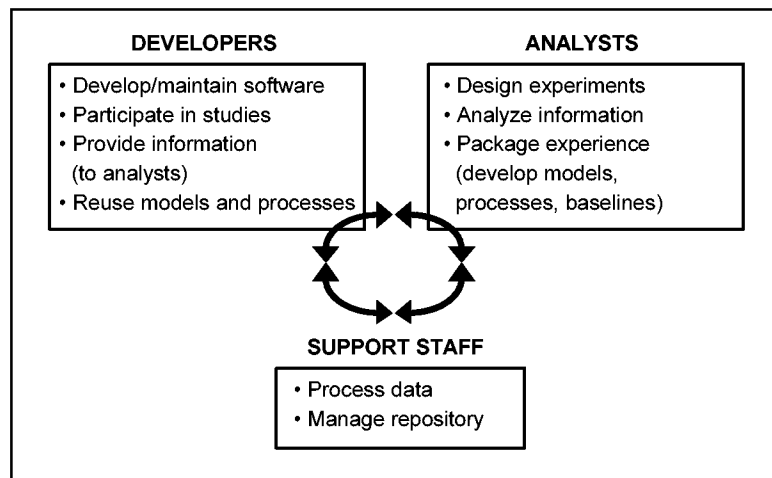
Figure 3-1. Activities of the Software Process Improvement Organization

Table 3-1. Activities of the Developers

Development Organization Component	Unchanged Activities	Changed Activities
Developers and Maintainers	Produce software on time and within budget (primary goal)	Interact with analysts for training, goal setting, and feedback
Management	Plan and control activities Use management tools (e.g., earned value) Act within management chain of command	Set up interfaces with analysts and support staff
Software Process	Adhere to process defined for development and maintenance	Interact with analysts for training Use tailored processes Use analyst-developed standards
Products Output	Generate software and related documentation Document lessons learned	Provide data to analysts Provide products, processes, and lessons learned to analysts

Tables 3-2 through 3-4 present an overview of the process improvement activities and the information exchanged between these groups during each time period. The remainder of this chapter provides details relating to these activities.

Table 3-2. Activities Before or at Project Start

From Developers To Analysts	From Analysts To Developers
• Understanding of project needs	• Experiment goals • Understanding of changes to the process • Training, as needed • Tailored processes • Refined process models • Experience-based policies and standards • Pertinent tools • Identification of project representative
From Developers To Support Staff (none)	**From Support Staff To Developers** • Clear descriptions of data to be provided • Clear and precise definition of terms • Identification of who is responsible for providing which data • Understanding of when and to whom data are to be provided
From Analysts To Support Staff • Set of forms to be used • Measures to be collected • Modified or technology-specific forms, as needed • Report formats • Reporting procedures	**From Support Staff To Analysts** (none)

Table 3-3. Activities During the Project

From Developers To Analysts	From Analysts To Developers
• Suggestions for process refinement • Project status and feedback at periodic meetings	• Processes and standards • Models and relationships for use in estimation and planning • Help in applying modified process • Training, as needed • Periodic status on the experiment, usually through the project representative
From Developers To Support Staff	**From Support Staff To Developers**
• Updated information (e.g., personnel changes) • Data (through data collection forms) • Feedback on data collection procedures • Documents and data for archival	• Reminders when forms are not submitted • Incorrect or incomplete forms • Periodic reports (usually to the manager) • Archived documents and reports, as needed
From Analysts To Support Staff	**From Support Staff To Analysts**
• Modified report formats • Modified reporting procedures • Documents, technical reports and training materials for archival	• Raw data from the repository • Periodic reports on the project • Problem reports on data collection procedures • Archived documents and reports, as needed

Table 3-4. Activities At or After Project Completion

From Developers To Analysts	From Analysts To Developers
• Final lessons learned at a project debriefing • Subjective assessment of the experiment	• Feedback on experiment results
From Developers To Support Staff	**From Support Staff To Developers**
• Project data, specifically for close-out • Documents and reports for archival • Feedback on data collection process	• Final reports on the project • Archived documents and reports, as needed
From Analysts To Support Staff	**From Support Staff To Analysts**
• Documents and reports for archival • Feedback on data collection process	• Raw data from the repository • Final reports on the project • Archived documents and reports, as needed

3.2 Developers

3.2.1 Overview

The developers comprise the largest element of the overall organization. They are dedicated to the development or support of software and may be involved with one or more projects. Their activities define the application domain for the software process improvement program, which is the "whole world" as far as the software process improvement organization is concerned.

The traditional role of the developers is not substantially changed by the fact that it is embedded in the software process improvement organization. The developers are the most critical part of the organization and absorb the majority of its resources. The developers are given a problem and have to solve it in the best possible way, within given time and budget constraints. As far as development and management activities are concerned, the major difference between a traditional environment and a software process improvement environment is the continual and consistent use of both the data collected in previous projects and models derived from those data by the analysts. In other words, data collection and analysis are emphasized more than in traditional environments.

The developers' management structure is not changed or affected by the software process improvement program. The analysts have no management responsibility or authority over the developers. However, to support the concept, the development management structure may need to take on some additional responsibilities. These could include interfacing with the other elements or utilizing specific training and policies provided by the analysts.

3.2.2 Resources

When the developers become part of a process improvement organization, their *staffing* is generally left unchanged. Some activities that may previously have been the developers' responsibility (e.g., the development of standards and training courses) are transferred to the analysts. Therefore, some project resources may be allocated to the analysts, possibly on a part-time basis.

The *management* of the development organization is not affected by the software process improvement program, but higher levels of management probably will be shared with the analysts and support staff. Developers are not subordinated to the analysts. The amount of management functions should not increase for the developers.

The developers' *budget* is not affected by the software process improvement program. The additional functions, which include interfacing with the analysts, providing project development data, and using models and processes developed by the analysts, should be carried out with no significant impact to the overall project cost. As an upper limit on cost impact, some organizations may assume that there is a 1 or 2 percent overhead determined by the additional training, meetings, and data gathering activities, but experience has shown that the additional overhead can be absorbed by the development budget. The developers' resources should not be appreciably impacted. Ideally, it would be beneficial to allocate an additional 1 to 2 percent to the developers to compensate for expenses related to activities such as meetings, data gathering, and training.

3.2.3 Activities

The primary responsibility of the developers is to develop or maintain software. They must not be burdened with software process improvement activities; therefore, their responsibilities for software process improvement are minimal. The development manager, however, is involved with some analysis activities as detailed in the remainder of this subsection.

The interface between developers and their customers is unchanged, although the use of data, defined models, and other products made available by the analysts will ease some aspects of this relationship. Those data and models should make the whole development process more controlled and predictable, even from the customer's point of view.

The developers perform the following functions to support software process improvement:

- Provide data

- Participate in studies

- Use experience packages

They also may be asked occasionally to *meet with the analysts* for feedback sessions to verify preliminary data analysis, for interviews to gather additional project characteristics data or subjective information, or for training sessions to reinforce the proper use of specific processes being applied by the developers.

Project personnel (developers) are responsible for providing project data. To do so, they complete data forms and submit them on a regular basis as agreed to by the managers and analysts. The forms are delivered to a specified, convenient location or handed to a designated individual. The developers simply provide the data; they assume no responsibility for analyzing them.

The developers may be asked to participate in a study of an experimental use of some process, technique, tool, or model that is not part of the organization's standard process. For projects undergoing significant process changes, the developers will need to attend briefings or training sessions on using the new process. At various stages in the experiment, the developers need to provide their insight regarding the value and relevance of interim results, the degree of success derived from the innovation, and the difficulties experienced in applying the new process. For the majority of projects, the only training needed is on data reporting agreements and the use of data collection forms.

Though the organizational experience is drawn directly from the developers, it is packaged by the analysts. The developers must then use these experience packages as part of their standard development process. Developers continually use analyst-provided packages, such as models, standards, handbooks, and training. Developers participating in experiments also use process models and processes refined by the analysts.

In support of these functions, the developers perform distinct activities corresponding to each phase of the project.

Before or at the start of a project, the developers and the development manager perform several activities to support process improvement. Together with the analysts, the development manager

- Defines the experiment associated with the project.

- Identifies measures to be collected.

- Identifies training needed for developers.

- Determines the experiment goals.

- Determines forms to be used to collect appropriate data.

- Decides what is to be altered for the project.

- Determines process and standards to be applied.

The manager provides project start-up information such as project name, preliminary start and phase dates, and estimates, to the support staff. They also provide the names of project personnel who will be supplying data. At this stage, the developers (technical staff) receive training, as needed, and instructions on data collection procedures.

For each project, an analyst is designated as a project representative to act as liaison between that development effort and the analysis organization. It is during this time frame that the analysis organization identifies the project representative. Most interaction between the developers and analysts takes place through this project representative.

During the project, the developers perform several activities to support process improvement. The developers (both management and technical staff) continually provide project data by completing data collection forms and submitting them (manually or electronically) to designated locations. The lead developer collects the forms from the development team and quality assures them to ensure that the numbers add up, the dates are correct, and the forms are filled in properly. The forms are then submitted to the support staff for processing. Project managers are responsible for periodically re-estimating size, schedule, and related information through appropriate data collection forms.

Throughout the project, developers and analysts interact to clarify process changes and provide feedback on the experiment. Most interaction between the developers and analysts takes place through the assigned project representative.

At or after project completion, the developers perform several activities to support process improvement. They provide project close-out data, including final system statistics (size, phase dates, etc.) and subjective information that might help characterize the problem, process, environment, resources, and product. The developers and analysts jointly generate a lessons-learned document. The majority of this document comes from the developers and focuses on the development effort as a whole, not specifically on the process improvement activities (the analysts capture project-specific lessons learned focusing on the process improvement activities and experiment(s) performed). The lessons-learned document is generated within 1 month of project completion.

3.3 Analysts

3.3.1 Overview

Because the analysis organization exists solely to support software process improvement, all the analysts' activities directly pertain to the organization's software process improvement program. The analysts are responsible for extracting information from the developers and then analyzing, synthesizing, and packaging the experience into reusable products for ongoing and future development and maintenance efforts. Their goal is the synthesis and packaging of reusable experience in the form of models, standards and policies, training materials, and lessons learned. The development of this information is based completely on lessons and data from past projects. The information is made available for use in the current projects of the same application domain. The analysis organization may be logical rather than physical, meaning that

- Personnel may be allocated to the analysts on a part-time basis.

- The analysts have their own levels of management but, as part of a larger parent organization, typically share the next higher level management with the developers.

- The analysts may include external consultants and researchers.

Most of the analysts' activities consist of defining and analyzing the information provided by the developers and feeding back the analyzed information to the developers. Because the information is contained in the experience base, or repository, the analysts must also regularly interact with the support staff to make sure that the information is appropriately collected, validated, and stored.

When compared with a traditional environment, the analysts' activities are new ones. Some activities, such as the development of standards, the development of training, and the production of models, may have existed but were previously carried out by the developers. Under a process improvement program, these activities would become the responsibility of the analysts.

3.3.2 Resources

Ideally, the *staff* of the analysis organization should include some experienced developers who have good field experience on processes and technologies used in the development organization, and researchers who are experienced in applying and assessing new concepts of software engineering technology. Given the particular role of the analysts, basic training in software engineering principles and techniques is desirable. The staff could also include experienced developers allocated to the analysis organization on a part-time or temporary basis.

The analysts have their own *management*, but higher levels of management are shared with the developers and support staff.

The analysts' *budget* and staffing levels are proportional to the budget and the size of the development organizations supported. On the basis of NASA experiences in typical organizations (where the development organization ranges in size from 100 to 500 people), the analysis element is typically between 5 and 10 percent of the overall size of the development organization (see the example in Figure 3-2).
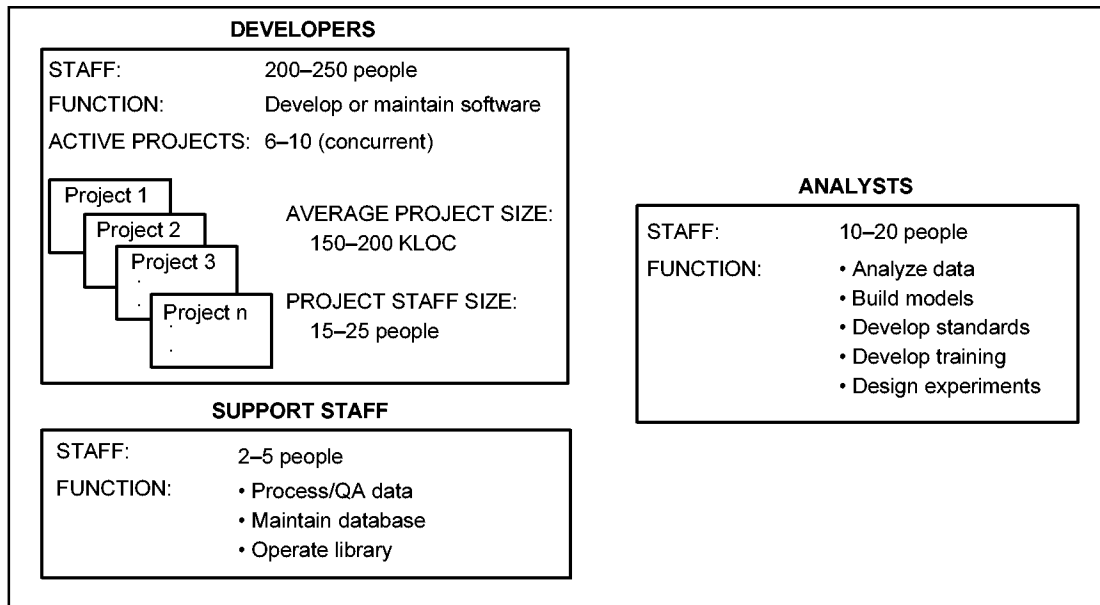
```
+-----------------------------------------------------------------------------------+
|                            DEVELOPERS                                             |
|  +----------------------------------------------------+                           |
|  | STAFF:            200–250 people                   |   +-------------------+   |
|  | FUNCTION:         Develop or maintain software     |   |    ANALYSTS       |   |
|  | ACTIVE PROJECTS:  6–10 (concurrent)                |   +-------------------+   |
|  |  +---------+                                       |                           |
|  |  |Project 1|       AVERAGE PROJECT SIZE:           |   STAFF:      10–20 people |
|  |  | Project 2|         150–200 KLOC                  |   FUNCTION:   • Analyze data|
|  |  |  Project 3|                                      |              • Build models |
|  |  |   .       PROJECT STAFF SIZE:                    |              • Develop standards|
|  |  |    Project n    15–25 people                     |              • Develop training |
|  |  |     .   |                                        |              • Design experiments|
|  |  +---------+                                        |                           |
|  +----------------------------------------------------+                           |
|                          SUPPORT STAFF                                            |
|  +----------------------------------------------------+                           |
|  | STAFF:       2–5 people                            |                           |
|  | FUNCTION:    • Process/QA data                     |                           |
|  |              • Maintain database                   |                           |
|  |              • Operate library                     |                           |
|  +----------------------------------------------------+                           |
+-----------------------------------------------------------------------------------+
```

Figure 3-2. Sample Process Improvement Organization

3.3.3 Activities

The heaviest burden of software process improvement activity falls on the analysts. They are entirely responsible for cultivating continual software process improvement within the organization. The primary operational responsibilities of the analysts are to

- Design studies

- Analyze project data

- Package results

The activities of the analysts can be associated with several development organizations or with several segments of the same development organization. All activities of the analysts are associated with software process improvement.

The analysts must first design studies supporting the organization's process improvement goals. They identify candidate process changes that address the organization's needs and that appear likely to improve the resultant product by reviewing literature and consulting developers who have insight into the problem area. Each development project is considered an experiment (i.e., a study of software engineering processes), and an experiment plan is written for each. These experiments can range in scope from validation of the current organizational models to controlled investigations of the impact of introducing a new methodology and involve developers as well as analysts. In addition to these individual experiment plans, the analysts (usually a lead analyst) work closely with the organization's managers to prepare higher level organizational plans coordinating the process improvement activities across all projects to ensure that all high-priority organizational goals are being addressed. They identify data to be collected and organize this collection based on the characteristics of the organization that is experimenting with the selected technologies.

The analysts are responsible for analyzing project data to develop and maintain organizational models (e.g., cost estimation models, resource models, error profiles) and to determine the impact of new technologies, such as object-oriented design, on the organization. They develop and update standards that incorporate into the normal organizational procedures the new technologies, the processes that are associated with them, and the models that support their use. They develop tailoring guidelines for the standards. They also develop training materials and programs to institutionalize the learning and use of new technologies.

Finally, the analysts must package the results and provide the derived information to the developers in useful forms, such as guidebooks, tools, and training courses. They are responsible for developing standards for the organization's process based on the experiences of that organization's developers. The analysts train the developers in activities such as using models and relationships that support the new technologies, planning and estimating using those models, controlling the execution of the new and updated processes, and tailoring mechanisms for standards. They fine-tune models and relationships to project-specific characteristics, possibly using parameters already provided with the models. They also provide feedback to the developers based on information obtained from them. This element of the analysts' responsibilities is a critical one. Information, results, and progress must be continually fed back to the developers. Ultimately, all items packaged by the analysts are for the developers' use.

In support of these functions, the analysts perform distinct activities corresponding to each phase of the project.

Before or at the start of a project, the analysts perform several process improvement activities. For each project, an analyst must be designated as a project representative to act as liaison between that development effort and the analysis organization. The project representative produces an experiment plan that defines the goals and approach of the experiment, provides a brief overview of the development effort, and describes the data to be collected. The analysts work with the development manager to define the experiment and to determine what is needed from and by the developers. They provide models and relationships for use in estimation and planning to the development manager. Process-specific training is given to the developers.

During the project, the analysts perform several process improvement activities. Throughout the project, they continually extract and analyze project information stored in the database, refine processes as needed based on feedback from the developers, and support the developers in applying the experience packages and refined processes. They continually interact with the support staff to ensure that data collection, processing, and reporting run smoothly.

At or after project completion, the analysts perform several process improvement activities.

For individual experiments, the analysts

- Extract and analyze project information stored in the database.

- Assess the experiment results.

- Package results (e.g., technical report, updated process guidebook).

- Jointly with the developers, generate a lessons-learned document. The developers provide most of this document, focusing on the development effort as a whole rather than specifically on the process improvement activities. The analysts (usually the

project representative) capture project-specific lessons learned focusing on the process improvement activities and experiment(s) performed. This document is generated within 1 month of project completion.

Based on results of multiple experiments across many projects, the analysts

- Tailor standards and guidebooks.

- Assess models.

- Update models, as necessary.

3.4   Support Staff

3.4.1 Overview

All support staff activities are directly related to the organization's software process improvement program. They are primarily responsible for processing data, including collecting, quality assuring, managing, and archiving all project data and for maintaining the information repository, which involves maintaining the organization's experience base. The actual experience base, or repository of information, consists of two basic components:

- *Projects database.* This component is usually a relational database, with associated data entry and data reporting functions. It contains the historical data from the projects, such as cost, schedule, and errors. A sample structure of such a database can be seen in Reference 17.

- *Library.* This second component is a document management and production infrastructure, possibly but not necessarily automated, that supports storage, retrieval, and distribution of project-related items (data collection forms, project-related documentation) and analyst-produced experience packages such as models (usually documented in reports and guidebooks), standards, policies, handbooks and guidebooks, and reports.

3.4.2 Resources

The support *staff* require a different set of skills. They are not necessarily experienced in software engineering, but they have practical experience with the tools used in the experience base (e.g., database and document management systems).

The support staff have their own *management* but, like the analysts, share the next level of management with the developers.

The support staff size and *budget* are smaller than those of the analysts. Based on experience, a reasonable ratio for a staff supporting environments of 200 to 500 developers is half of the overall budget of the analysis organization. These figures reflect a structure similar to the one presented in Figure 3-2. A reasonable estimate for the cost of this function is 5 percent of the development cost. For much larger organizations, experience has shown that the percentage decreases such that a support staff of 20 can carry out this function for an organization of several thousand.

3.4.3 Activities

The support staff exist solely for software process improvement; therefore, all their responsibilities are directly related to software process improvement within an organization. The primary operational responsibilities of the support staff are independent of specific projects. They are to

- Process data

- Maintain the information repository

The activities of the support staff are to

The support staff process, i.e., collect, store, quality assure, summarize, and export, the organization's project data. They manage the information provided by the developers to ensure that it is complete, consistent, and of adequate quality so the analysts can use it to develop the models and gain a general understanding of the software process. The support staff typically use a commercially available relational database management system (RDBMS) to store the project data. The support staff assign a database administrator (DBA) for the organization. The DBA coordinates data collection activities, gets appropriate information from the analysts (e.g., what is being monitored for specific projects), and serves as the interface to the developers. To ensure quality, the support staff monitor the regularity and completeness of the data collection process. They apply the data collection procedures provided by the analysts and report any problems encountered in their execution. They also manage the data collection forms, making sure that they are available in the current format to whoever needs them. As data become available, the support staff enter them into the projects database. They make sure that the data are formally consistent with the data collection standards. They also archive documents, technical reports, and other project-related information, making sure that the current versions are available and that the outdated versions are appropriately handled. The support staff are responsible for getting data from two sources:

- *Directly from project personnel.* Most project data are gathered directly from the developers through data collection forms. The support staff must make the data collection process as painless as possible for the developers. They must ensure that an interface is clearly established between themselves and the developers so that the developers can easily provide the project data. Developers must understand who is responsible for collecting and furnishing project data, how frequently the data will be collected, which portions of the software life cycle will be reflected, and what type of personnel (management, technical, or administrative) will be included. The support staff are responsible for managing the data collection forms; they must ensure that the forms are available to those who need them, clearly indicate where they are to be deposited, and promptly collect and process them. They must ensure that there is a consistent understanding of the software measurement terms and concepts and must supply concise, clear definitions to the developers. The analysts are responsible for writing definitions that are consistent with organizational goals and locally understood ideas; however, the support staff are responsible for furnishing the definitions to the data providers (the developers).

- *Automatically from the project.* Some information, such as source code growth rate or computer resources usage, is monitored and gathered electronically, without direct input from the developers. The support staff develop, run, and maintain procedures for this automatic data collection.

The support staff are responsible for maintaining the information repository, which includes maintaining, archiving, and distributing all output from the analysts such as archived reports, standards, training materials, and experimental studies. They are responsible for maintaining both components of the repository: the projects database and the library.

THE PROJECTS DATABASE. After collecting the data, the support staff store them in an on-line database, preferably a commercially available RDBMS. The quality of the stored data must then be considered. The support staff should quality assure the data using a two-step process:

1. *Verify the source data.* Support staff track discrepancies to the source and correct them. This step includes checking that the data forms have been submitted and are complete (i.e., all required values are provided); values are of the specified type (e.g., numeric fields do not contain non-number values); values are within specified ranges (e.g., number of hours of effort per day per person is never greater than 24); and values are reported on the prescribed schedule.

2. *Verify the data in the database.* After the data are entered into the database, support staff perform a second check to verify that the entries match the source value.

The support staff maintain and operate the database. They develop, execute, and maintain procedures for the operation of the database including start-up, shut-down, backups, restorations, reorganizations, and reconfigurations. Occasionally, changes to the data collection process will be introduced. The support staff are responsible for evaluating the effect of such changes on the database design, supporting application software, data collection procedures, and documentation. They must implement the changes and ensure that earlier data are not rendered obsolete or comparisons invalidated.

The support staff produce and distribute reports and data summaries to users in all of the software process improvement program's organizational components. Many reports are generated on a regular schedule. These include single project summaries that focus on a particular data type and multiple project roll-ups that provide high-level statistics facilitating project-to-project comparisons. These reports may be distributed to developers to provide feedback on project measures. Analysts also use these reports to identify projects and data to be used in studies and model generation. The support staff may also generate reports, such as low-level data dumps from the data verification process, on an ad hoc, as requested basis.

Occasionally the support staff are also responsible for preparing and exporting raw data to external organizations. Before sending the actual data, they need to sanitize them to preserve the confidentiality of data providers (e.g., removing names of individuals and substituting generic project names for actual ones).

THE LIBRARY. The support staff maintain the organization's library of products supplied by both developers (e.g., lessons learned) and analysts (e.g., experiment plans, technical reports, standards, policies, and handbooks). They organize and maintain a catalog of the library's contents, archive documents and technical reports, ensure that current versions of archived

documents and reports are available, remove outdated versions from the library, and prepare and execute document reproduction procedures.

In support of these functions, the support staff perform distinct activities corresponding to each phase of the project.

Before or at project start, the support staff perform several activities to support process improvement in addition to the project-independent activities (i.e., processing the data and maintaining the information repository). From the analysts, the support staff get the appropriate forms to be used, measures to be collected, and any information or instructions specific to that project. The support staff must ensure that communications have been established with the developers and that the data collection procedures are clearly understood. The DBA typically meets with the project leader to ensure that the developers understand what is expected of them and that the support staff understand the particular project and any unique changes being applied. The support staff obtain basic project information from the development manager including names of project personnel and initial estimates.

During the project, the support staff perform several activities to support process improvement in addition to the project-independent activities (i.e., processing the data and maintaining the information repository). Throughout the project, they interact with both developers and analysts and provide information (data, documents, reports, etc.) on a regular basis, including responding to special requests for information. They must also ensure the smooth operation of the organization's information repository.

At or after project completion, the support staff perform several activities to support process improvement in addition to the project-independent activities (i.e., processing the data and maintaining the information repository). They collect and process project close-out data and generate final reports. They process documents and reports for archival and respond to requests for information.

3.5   Summary

The success of the software process improvement program depends on the smooth operation of its components. The program, as a whole, is only as effective as the individual components. Aided by the support staff, the analysts are responsible for facilitating software process improvement within the organization. The developers are the source of experience and the cornerstone of the entire software process improvement program. Every effort should be taken to extract their experience in an unobtrusive manner. When the program is operating effectively, its activities are viewed by developers as the standard way of doing business, not as some "necessary evil." All members of the organization reap the benefits and become willing to support and advocate the process improvement program.

Chapter 4. Implementation of the Software Process Improvement Program
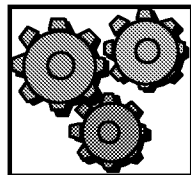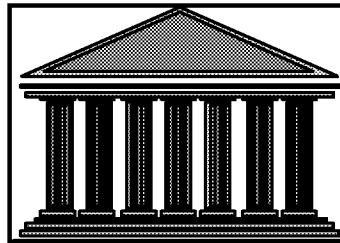

Chapter Highlights

OBTAIN COMMITMENT

Gain support of key individuals
Designate resources
Focus on first products
Produce software process improvement plan

ESTABLISH STRUCTURE

Determine affected elements of development organization
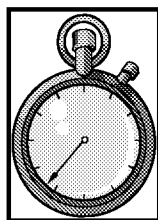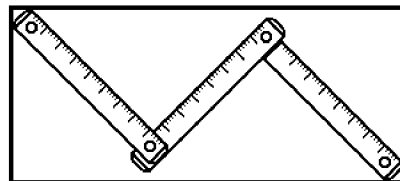Establish analysis organization
Establish support staff

ESTABLISH PROCESS

Define data to be collected
Define terminology to be used
Define data collection, quality assurance, and archival procedures
Define how to capture development lessons learned

PRODUCE BASELINE

Capture snapshot of organization s environment, process, and product characteristics

START OPERATION

Initiate normal day-to-day operations of the software process improvement organization

This chapter describes the five steps necessary to establish and implement a software process improvement program:

1. Obtain commitment from the organization.

2. Establish the basic structure of the software process improvement organization.

3. Establish the process and operational concepts for the organizational elements.

4. Produce the organizational baseline.

5. Start operation of the software process improvement program.

The ideas behind all the steps of the implementation process pertain to the *scope* and *focus* of the software process improvement program. The *scope* is delimited by a specific domain. The implementors, that is, those responsible for establishing the software process improvement program, should start with a small, but very reactive, initiative that represents limited overhead and provides visible benefits. The opportunity for widening the scope to other domains and organizations will exist if the original effort was mostly successful. The *focus* is the improvement of software in a specific domain using lessons learned from experimentation with technologies in real projects. The implementors should resist the temptation to introduce changes to the process that, although making sense, are not in tune with the overall goals of the organization that were set in the Understanding Phase after the baseline was established. The improvement must target the areas where it is most needed.

Starting small is also important. Limiting the number of projects affected, restricting the portions of the software life cycle to those with already-defined processes within the organization, and limiting staff involvement to essential personnel will all help to minimize resistance from, and impact on, managers and developers. The scope of the program will evolve, but the time to increase the size of the program is after it has become successful.

4.1   Obtain Commitment

Obtaining commitment is crucial to the success of the software process improvement program. At this point, the implementors need to

*   Inform all levels in the organization about the organizational goals and the changes implied by the software process improvement program.

*   Obtain support from some key individuals.

*   Prepare the ground for execution of the next steps.

The software process improvement program must be sold to the organization by showing its practical benefits for everybody. In particular, the concept must be sold to the key decision-making levels of the organization to obtain, and retain, their support during both implementation and operation of the program.

The major action items included in this step are to

- *Enlist an advocate.* The ideal advocate is an experienced senior software engineer who can dedicate at least half of his or her time to the software process improvement activities. This person must have significant insight into all of the organization's development efforts and general activities because he or she will be responsible for coordinating all the process improvement activities, making the final determination for experiments to be performed and the projects to which they will be assigned.

- *Increase awareness.* Awareness must be raised with two key groups: the developers and other support elements. The role and responsibilities of the developers must be clarified because they are the drivers of the improvement process. Support groups, such as Quality Assurance and Project Control, also must be made aware of the process improvement activities because they will be heavily affected by the changes in the organization and its standards.

- *Secure management support.* Management does not have to be a direct advocate of the initiative but must be aware of it, understand its goals, and support it explicitly and implicitly.

- *Designate resources.* The most important resource in the implementation of the software process improvement program is staff time. Management must be aware that the improvement program does not come free and must allocate enough resources to the effort. Some experienced members of the development teams must be allocated, at least part time, to software process improvement, because they are the current owners of the experience that will be packaged. Resources and time must also be allocated for the support staff.

- *Focus on the first products.* The goals of the process improvement program must be clearly defined in terms of the

  - Concept they implement

  - Needs they are going to satisfy

  - Expected baseline changes

  - Impact on existing software policy

- *Produce a software process improvement plan.* This plan will define in an operational way the role of the process improvement program by dealing with the following topics:

  - Discussion of the concept (Why does the organization want to do this?)

  - Outline of the program (Describe what it will and will not do.)

  - Definition of the scope of the overall program (Who will it affect? Which types of projects will participate?)

  - Identification of measurable goals and drivers (What will be the measure of success? How can adjustments be made?)

- Process description (How will changes be introduced? How will improvement be managed? How will new technology be introduced?)

- Resources assigned to the tasks (Who? How much time? What will it cost?)

4.2 Establish Structure

The structure established in this step was outlined in Chapter 3. It consists of three components: developers, analysts, and support staff. Based on the description of the components' roles given in Chapter 3, the major action items included in this step are to

- *Define the scope of the development organization.* Determine organizational units involved (which projects? departments? functions?). Determine the software to be included (what life-cycle phases will be addressed? what types/classes of software?). Specify roles and responsibilities (and points of contact, when appropriate) for interfacing with the analysts and support staff.

- *Establish the analysis organization.* Assign staff to the analysis organization, according to the established criteria. Set the focus on some early products.

- *Establish the support staff.* Assign staff and other resources (e.g., space for physical location) to the support organization according to the established criteria. Establish the data collection, quality assurance, and other procedures necessary for the organization to run efficiently. Determine tools to be used (e.g., RDBMS).

Figure 3-2 showed a sample process improvement organization. Figure 4-1 represents the same organization but shows some additional details pertaining to the structure of that particular organization.
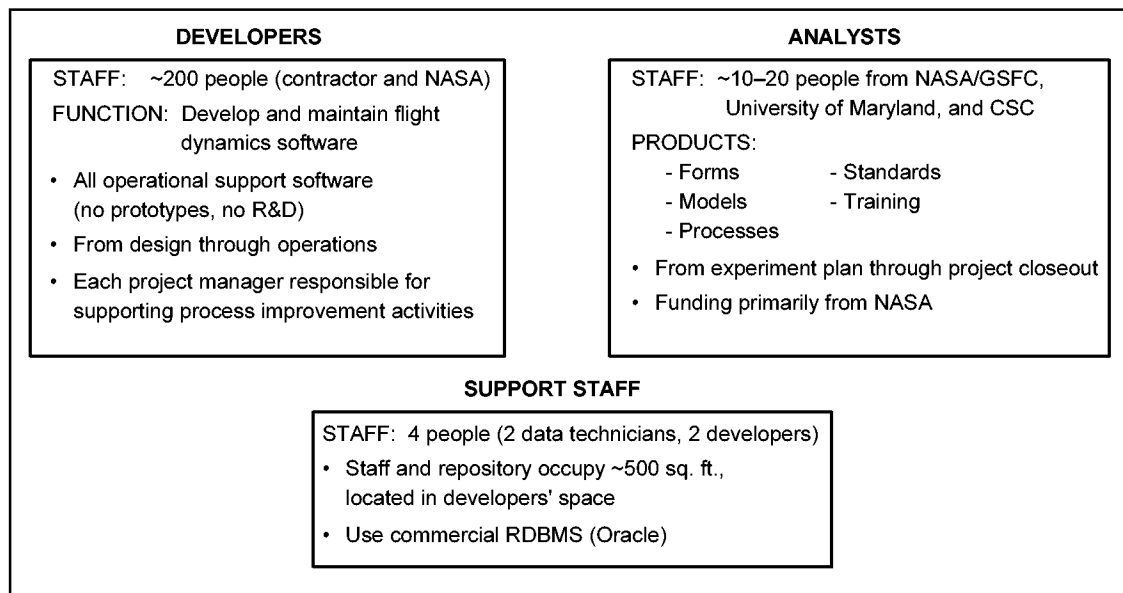
DEVELOPERS

STAFF: ~200 people (contractor and NASA)

FUNCTION: Develop and maintain flight dynamics software

- All operational support software (no prototypes, no R&D)
- From design through operations
- Each project manager responsible for supporting process improvement activities

ANALYSTS

STAFF: ~10–20 people from NASA/GSFC, University of Maryland, and CSC

PRODUCTS:
- Forms   - Standards
- Models   - Training
- Processes

- From experiment plan through project closeout
- Funding primarily from NASA

SUPPORT STAFF

STAFF: 4 people (2 data technicians, 2 developers)
- Staff and repository occupy ~500 sq. ft., located in developers' space
- Use commercial RDBMS (Oracle)

Figure 4-1. Sample Process Improvement Organizational Structure

A critical issue throughout the execution of this step is the clear specification of what is being done. At this stage, the tendency to broaden the scope is strong. The implementors must remember that a clear definition of scope and products and a rigorous specification of and compliance with the roles established at the beginning are the key to success. If adjustments need to be made, they must be explicitly planned and motivated.

4.3 Establish Process

The purpose of this step is the definition of the operational concept and the development of the necessary instruments for each component of the software process improvement organization. The basic operation of the software process improvement program has been presented in Chapter 3.

Based on that description of the operation, the major action items performed in this step are to

- *Define what data will be collected and archived.* What forms will be used?

- *Define terminology to be used.* Prepare a glossary defining terms commonly used and describing how they are used (e.g., What is a line of code? What is an error?).

- *Define how data will be collected, quality assured, and archived.* Established detailed data collection procedures. Define step-by-step timelines. What is the database organization?

- *Define how the development lessons will be captured.* Define report formats.

The organization must find the best way of operating the process improvement program in its specific context. It is not necessary to overplan and provide detailed specifications for the operation of the program. The process will evolve as the organization learns from its own experiences. Defining and documenting detailed data collection procedures is necessary, however, to guarantee the integrity and completeness of the data.

4.4 Produce a Baseline

Producing a baseline is, perhaps, the most critical element in the software process improvement approach. In this step, the organization captures a picture of itself as it exists at that time. The baseline is a characterization of the organization's software process and product. The baseline should include insight into the following areas:

- How much software exists within the organization?

- What are the characteristics of the organization's product?

- What are the characteristics of the organization's process?

- What are the perceived strengths and weaknesses from selected relevant perspectives (e.g., developers, customers and users, management)?

The instruments used in building a baseline include the usual data collection instruments: surveys, roundtables, interviews, and historical data. The questions that can be asked in the survey or during the interviews and roundtables are aimed at collecting information such as

- Resources (i.e., people, money, time) dedicated to software activities (i.e., development, management, maintenance)

- Amount of software being developed or maintained, as well as the domain and lifetime associated with the software

- Hardware and software environment (tools, languages, etc.)

- Methods and technologies used in the different phases of the software life cycle

- Major problem areas and sources of errors

During the development of the baseline, measures can be associated, where possible, with the information that is collected. Key measures will characterize the software process and products, emphasizing characteristics closely related to the overall goals of the organization. To identify these key measures, goals are identified that say why data are being collected. The goals can be refined into questions based on the phase they address and the viewpoint and characteristic they take into account. Measures are then associated with these questions to answer them in a quantitative way. Reference 4 provides additional information on establishing goals, refining them into questions, and identifying appropriate measures for data collection.

Other measures can be collected during the baselining effort that may have relevance to the improvement goals being set within the organization. Examples of such measures include

- Error density (e.g., errors per KSLOC)

- Staffing (e.g., number of people) and effort (e.g., staff months) per project

- Software measures (e.g., SLOC, complexity)

- People characteristics (e.g., education, experience)

The major action items performed in the baselining are to

- *Identify the major application domains in which the organization operates.* This activity can be accomplished by looking at the functionality of the software developed, general characteristics of the problem, organizational constraints, standards in use, platforms, and development environments.

- *Develop and adapt data-gathering mechanisms* suitable for the specific domain. Suggested mechanisms include administered surveys, informal roundtable discussions, data and documentation review, and one-on-one interviews.

- *Gather information and insight* by interviewing key individuals and groups. Use directed sampling. Start with senior managers to get an overview of the organization, to make them aware of the baseline efforts, and to identify software "pockets" within the organization. Sample these pockets. Be sure to get perspectives from throughout the organization (senior management, technical management, quality assurance, engineers, programmers, testers, etc.).

- *Analyze the data.* Cross-verify data collected.

- *Establish a baseline* that outlines the major distributions of relevant characteristics (effort per phase, errors per product, etc.) quantified by measurement.

- *Present results of baselining activities.* Present preliminary versions of the baseline to different organizational levels and incorporate feedback into the baseline.

Table 4-1 provides some guidance for the data collection involved with baselining. This guidance reflects lessons learned from the baselining of NASA software (References 2, 18, and 19).

Table 4-1. Key Lessons in Data Collection for Baselining

Do	Don t
Gather data in person.	Mail surveys.
Prototype and test survey vehicles.	Use descriptive entries.
Use quantities or checkmarks.	Use more than three people to collect data.
Use one person (or a small group) for data gathering.	Rely on someone outside the organization for baselining.
Use someone familiar with the organization for baselining activities.	Get wrapped up in details and statistics.
Look for trends and relative comparisons.	Expect quick results (baselining takes time).
Allocate time and resources for baselining.	

The baseline is meant to establish where an organization stands today. It is not a basis for judgment and should not be used to label the organization as good or bad. The baseline provides the organization with the basic understanding of its products and processes and enables the organization to measure and control change and progress. To support process improvement, the baseline must not remain static but must be maintained to reflect current data. See References 2, 18, and 19 for examples of completed reports of baselining activities within NASA.

4.5 Start Operation

The purpose of this step is to initiate the normal daily operation of the software process improvement organization. Based on the defined goals, a set of processes is initiated as described in the normal operational phases in Chapter 3.

The recommendations for the implementors in this phase are to

- Ensure that change is driven by the information contained in the baseline and by the perceptions of the developers.

- Pay attention to the day-by-day comments provided by the developers.

The normal operation of the software process improvement program begins according to the process described in Chapter 4. The operation follows the three-phase improvement process: understanding, assessing, and packaging, described in Chapter 2.

Chapter 5. Management of the Software Process Improvement Program

---

Chapter Highlights

COST
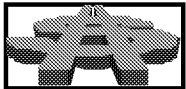
DEVELOPERS: No more than 2 percent overhead
ANALYSTS: Ranges from 5 to 15 percent
SUPPORT STAFF: Ranges from 3 to 7 percent

BENEFITS

Established improvement process
Repository of experience-based software processes and models
A process improvement infrastructure
Structured mechanism for introducing new technologies
A reuse-based software development process
Quantifiable benefits in specific organization

KEY MANAGEMENT GUIDELINES
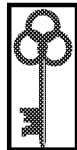
Limit scope
Clearly describe and assign roles
Keep analysts separate from developers
Ensure that developers drive change
Proceed slowly
Produce specific products

This chapter addresses key management issues associated with the implementation and operation of a software process improvement program. It discusses issues such as cost and staffing, as well as determining the payoff of having a process improvement program. The information is based on lessons learned and insight gained from having instituted such programs at GSFC's SEL, JPL's SORCE, LaRC's SEAL, and other places where a similar concept is in place and operating.

5.1 Cost Issues

The cost of process improvement is one of the most critical success factors for a software improvement initiative based on the approach presented in this guidebook. Besides the cost of the specific experimentation, whose goal is assessment and tailoring of software engineering technologies, an ongoing cost exists due to the presence of a measurement system supported by staff and tools. This section presents information available on the cost of software process improvement.

Process improvement is not free, but it can be tailored in size and cost to fit the goals and budgets of any software organization. A software process improvement program must be undertaken with the expectation that the return will be worth the investment. There will be a cost, however, and it must be estimated in the organization's budget; otherwise, there will be frustrations, attempts at shortcuts, and a failed program. Planning must take into account all the hidden elements of the proposed program—elements that often are more costly during start-up than they will be after the program becomes operational. The higher start-up cost is another reason to start small.

Planners often incorrectly assume that the highest cost of process improvement will be assigned to the developers. That part of the overhead expense, which includes completing forms, identifying project characteristics, and meeting with analysts, is actually the smallest portion of the three elements of the software process improvement program's cost:

- Cost to the software projects (overhead to the developers)

- Cost of quality assuring, storing, and archiving data and packaged experience (cost of support staff)

- Cost of analyzing and packaging data and experience (cost of analysts)

The cost of process improvement also depends on the following three scope considerations:

- The size of the organization

- The number of projects included in the program and supported by the software process improvement organization

- The extent of the software process improvement initiative (parts of the life cycle targeted by the initiative, number of pilot projects, breadth of the measurement program, etc.)

NASA experience shows that there will be a minimum cost associated with establishing and operating any effective process improvement program and its associated organization. The total cost will increase depending on the extent to which the organization wants, or can afford, to

expand the program to address additional projects, more comprehensive studies, and broader improvement activities.

The cost information available is based primarily on over 18 years of experience from organizations ranging in size from approximately 100 to 500 persons. Some additional information has been derived from process improvement programs in larger organizations of up to 5,000 persons. The number of projects active at any time has ranged from a low of 5 or 6 projects to a high of over 20 active projects, with the projects ranging in size from approximately 5 KSLOC to over 1 million SLOC. Because costs depend on a large number of parameters, a single definitive value cannot be cited that represents the cost of any organization's process improvement program. Based on experience, however, general suggestions can be provided that an organization can interpret in the context of its own goals and environment.

As a general rule, the overall cost of the program can be represented in terms of the cost to each of the three organizational elements:

- Overhead to the *developers* will not exceed 2 percent of the total project development cost and is more likely to be less than 1 percent (which implies that it is not actually measurable and is absorbed in the overhead).

- The *support staff* may reach a constant staff level of from one to five full-time personnel for data processing support. In addition, the cost of the database software will also be allocated to the support component.

- Several full time *analysts* will be required and may cost up to 10 or 15 percent of the total development budget. As an example, the SEL spends an average of about 7 percent of each project's total development budget on analysis and packaging.

Figure 5-1 illustrates the costs of the elements of a software process improvement program as percentages of the total organizational cost. The individual costs are discussed in more detail in the following subsections.

5.1.1 Overhead to Developers

> *The cost of software process improvement should never add more than 2 percent to the software development or maintenance effort.*

The smallest element of the cost of software process improvement is the overhead to the developers. This overhead includes the cost of completing forms, participating in interviews, attending training sessions describing measurement or technology experiments, and helping to characterize project development.

Although start-up costs may be as high as 5 percent of the development budget, the actual cost of operating an effective program will normally not exceed 1 or 2 percent regardless of the number of projects under way within the organization.

Some legitimate costs are associated with introducing the providers of data to a new program; however, part of the higher initial cost is attributable to the inefficiencies inherent in an inexperienced organization's program. Such new programs typically ask the developers to

complete unnecessary forms or require excruciating detail that is of little value or is not a part of the stated goal. A well-planned program will never impose a significant cost impact on the development or maintenance project.
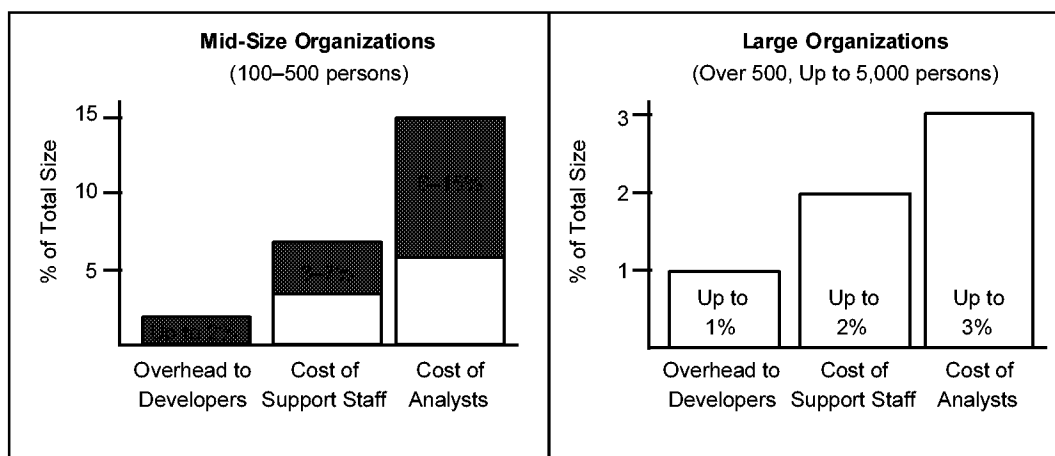

Figure 5-1. Cost of Software Process Improvement

5.1.2 Cost of Support Staff

> ### *The cost of the support staff may range from 3 to 7 percent of the total development budget.*

This element includes collecting, validating, and archiving data. It also includes database management, library maintenance, execution of support tools, and high-level reporting of summary measurement data. These essential activities must be planned, supported, and carefully executed. In addition to the cost of personnel supporting this activity, there will be the added cost of acquiring and maintaining database software, support tools, and other automated processing aids (e.g., code analyzers).

Within an organization of over 50 management, technical, and clerical personnel, any process improvement program will require three to five full-time staff members to handle the necessary support tasks. A smaller organization, with perhaps only one project and a pilot program, may wish to combine this data processing effort with the configuration management (CM) or independent quality assurance (QA) activities; implementation of a separate support element may not be cost effective. A large organization may benefit by creating separate, structural components to perform the three distinct roles. A small organization with a small project may simply assign the roles to individual personnel. In some cases, a single individual may perform multiple roles, as long as the amount of effort allocated to separate roles is clearly identified.

Experience within NASA has shown that the cost of the support staff supporting organizations of 100 to 200 software developers is approximately 7 percent of the total effort. That cost includes approximately five full-time support staff personnel (data technicians and database support personnel) plus the costs of the DBMS and associated software tools and equipment. For larger programs (250 to 600 software personnel), experience indicates that only one additional full-time support person is required. Thus, for organizations with 50 to 600 developers, the overhead cost

is approximately 6 percent of the project cost. For organizations with approximately 500 to 1,000 software personnel, the overhead cost approaches 3 percent of the project cost or about seven full-time personnel added to the cost of the tools and equipment.

The cost estimates are based on the assumption that an organization is actively working on 5 to 15 development or maintenance projects at any one time. The overall cost of the support staff will vary significantly depending on the number of projects participating in the program. An organization of 200 or 300 people actively working on a single large project will require much less support than the same organization with 20 active smaller projects. Limited experience with larger organizations of over 5,000 persons indicates that the support staff cost is essentially the same as that for an organization of 500. As its size increases, an organization tends to collect data at a less detailed level.

5.1.3 Cost of Analysts

> *The cost of analysis and packaging ranges from 6 to 15 percent of the total project budget.*

The analysis organization is the most critical part of the process improvement program and incurs the most cost of the three organizational elements. Without sufficient allocation of effort to the analysis and packaging function, the process improvement program will fail.

NASA experience shows that the cost of this element far exceeds the combined costs of the other two. A successful program demands that this cost be recognized and budgeted. For programs involving 50 to 250 software developers or maintainers, the cost of this activity has consistently run from approximately 7 to 12 percent of the organization's total budget. Costs include designing studies and developing new concepts; developing and writing standards; and analyzing, providing feedback, and developing improvement guidelines. The cost of this element depends on the number of active projects within the organization. The figures provided here assume at least 10 active projects and an archive of data from at least 15 projects available for analysis. The analysis cost would be smaller than indicated if there were fewer active projects.

NASA's historical data indicate that organizations spending between $20 million and $30 million for development and maintenance projects have spent between $1 million and $3 million for extensive and mature analysis efforts (in fiscal year 1993 dollars). For efforts on a much larger scale, the analysis must necessarily be conducted on a comparably higher level; consequently, the overhead percentage decreases significantly. An expenditure of an equivalent amount of analysis resources plus a modest increase due to the size of the organization need not exceed the lower range of cost for analysis activities. That is, for larger organizations, the cost of analysis and packaging activities need not exceed 3 percent.

Regardless of the size of an organization, adequate resources must be allocated for this critical program element.

5.2 Benefits Obtained

By implementing a process improvement program and establishing an organizational structure devoted to software process improvement, an organization can reap many benefits:

- An established improvement process for software, substantiated and controlled by quantitative data

- A repository of software processes and models that are empirically based on the everyday practice of the organization

- An infrastructure that requires a limited overhead and provides substantial cost and quality performance benefits

- A structured mechanism for identifying, assessing, and incorporating into the process new technologies that have proven to be valuable in similar contexts

- A reuse-based software development process including code, designs, processes, resources, models, lessons learned, and quality functions

The software process improvement program provides a corporate memory of software experiences that can be used in ongoing and future ventures. The organization gains the ability to learn from every project, constantly increase the maturity of the organization, and incorporate new technologies into the life cycle. In the long term, the process improvement program supports the overall evolution of the organization from a project-based one, where all activities are aimed at the successful execution of single projects, to a capability-based one, which utilizes the experience base across all projects.

Are there economic benefits in establishing and supporting a process improvement program? Identifying conclusive evidence of the economic benefits derived from process improvement programs is extremely difficult. The major reason for this difficulty is the relative immaturity of process improvement programs within the software industry whereby quantitative evidence could be derived.

Quantitative results can be obtained when measurement programs are mature enough to support the process improvement program. The goal of the process improvement approach detailed in this guidebook is to improve the products of an organization. Quantifiable benefits and improvements must be measured against the goals set by the specific organization to improve its products. Have error rates decreased and reliability improved? Has total system cost been reduced? Has productivity been improved? The results of process improvement should be quantifiable and should demonstrate positive return on investment depending on the goals of the organization.

Organizations that have been using the concepts of process and product improvement described in this guidebook can determine costs, benefits, and general impacts of such a program. Figures 5-2 through 5-6 show some tangible benefits obtained by one such NASA organization at GSFC. The data presented indicate the types of benefits that can be achieved by establishing a process improvement program, but the quantified results are specific to the one organization. The most important benefit demonstrated here is that an organization can quantify and qualify the impacts of change. Even if the impact is negative, the organization has gained experience and

benefited from the additional knowledge. Figures 5-2 through 5-5 focus on impact to product; they demonstrate change to the product over time as a result of process improvement activities.

Figure 5-2 shows the results of the organization's process improvement program on its product in the area of software reliability. As this figure illustrates, reliability has improved by 75 percent as the average error rate during software development has decreased from 4.5 to 1 error per KSLOC.
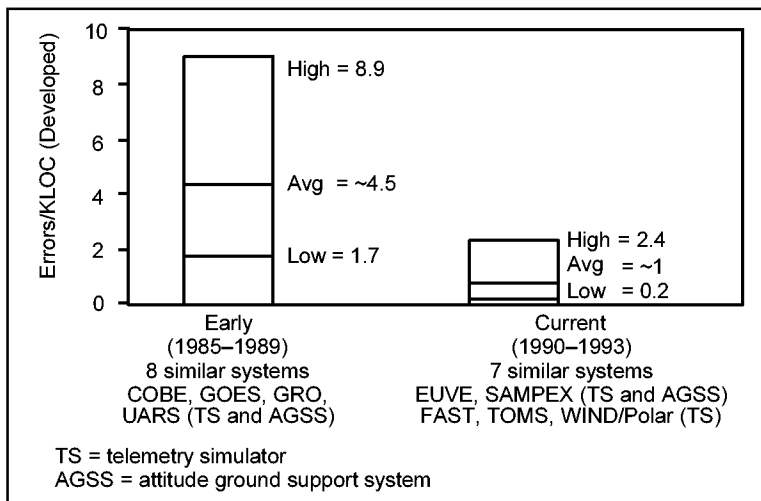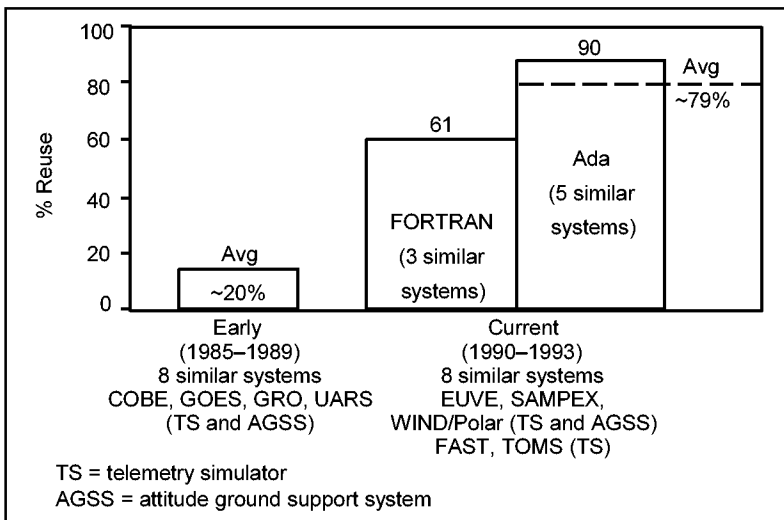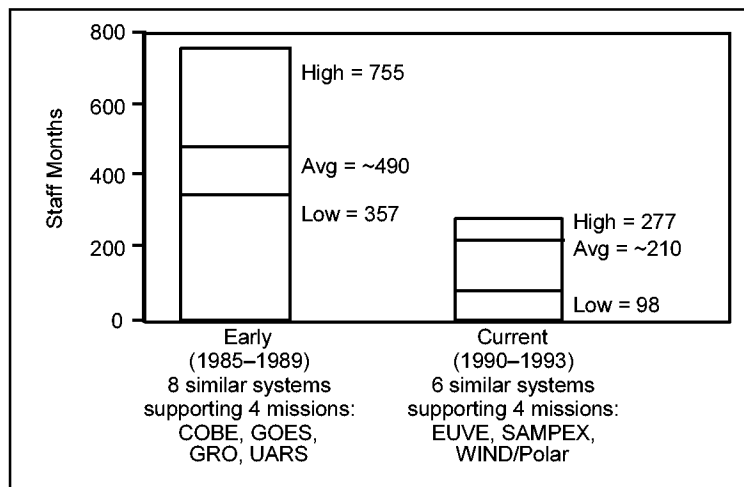

Figure 5-2. Improvements in Product Reliability

Figure 5-3 shows the results of the organization's process improvement program on its product in the area of reuse. This figure shows that the reuse rate has increased by 300 percent from about 20 percent to nearly 80 percent.

Figure 5-3. Improvements in Product Reuse

Figure 5-4 shows the results of the organization's process improvement program on its product in the area of software development cost. This figure shows that the typical mission cost (to deliver

several similar systems) has decreased by 55 percent from an average of about 490 staff-months to about 210 staff-months.

Figure 5-4. Improvements in Product Cost

Figure 5-5 shows the results of the organization's process improvement program on its product in the area of development cycle time (i.e., the amount of time required for development). As this figure illustrates, the average development cycle time has been reduced by 38 percent for both Ada and FORTRAN projects.
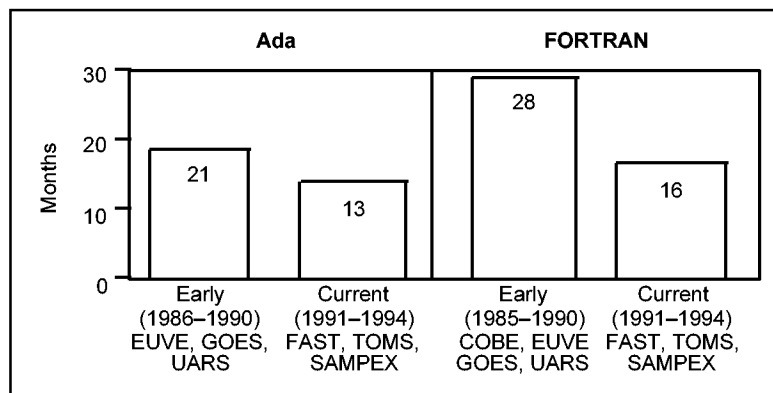
Figure 5-5. Improvements in Product Development Cycle Time

Table 5-1 focuses on the impact of the process improvement program on a specific organization's process. Within the past decade, many facets of the organization's process have evolved and matured as a direct result of process improvement activities. Software-related activities have been integrated. Training, standards and policies, measurement, and the adoption of new technologies are no longer performed in an ad hoc fashion; they fit into the three-phase improvement approach (understand, assess, package) and work together to meet the organization's needs. Developers become an integral part of the improvement process as they see their experience drawn upon and packaged for subsequent use. Software becomes process driven and less people driven. The three-phase approach also focuses the role of software engineering

research. This research becomes driven by the problems and goals of the organization, and a mechanism is in place for the experimentation, assessment, and adoption of new technologies.

Table 5-1. Impacts to Software Process

	Early to Mid 1980s	*Mid to Late 1980s*	*Early 1990s*
Process	**Good**	**Better**	**Even Better**
Standards	Top-down	Reflect environment	Experience driven; bottom-up
Training	By organization	For specific techniques	Full program based on local experience
Measurement	Overhead to process, data driven	Integrated, analysis driven	Organization's "way of doing business"
Life Cycle	Waterfall	Waterfall	Tailorable
Inspections	Code/design reading	Code/design reading plus peer reviews	Focused inspections, design inspections
Research	Ad hoc	Sometimes product driven	Product driven, based on improvement goals
Improvement Focus	Technology	Process	Improvement process part of standards; integrated experimentation and development

As these illustrations show, this organization has derived quantifiable benefits from its process improvement program in the areas of reliability, reuse, cost, and cycle time. Its process has also matured in many ways. Another factor considered by the organization is that the software being produced today is much more complex than in previous years. Despite the increased complexity, the organization has been able to produce the functionality needed for these more complex systems while improving reliability and reducing cost.

The use of a process improvement program to guide, manage, and improve processes will provide an action-feedback mechanism for recording the current performance of the organization and observing the actual impact of process changes. Control over change is a way to invest resources in changes that proved to be effective in achieving the overall goals of the organization. In general, a process improvement program should allow more control over what is happening in the organization. The models used by management to predict the behavior of the software processes, their cost, use of resources, compliance with schedule, and effectiveness, are based on internal data and experience. Thus, management should be better able to trust those models. Based on experience and information gained on previous projects, managers should also be better able to handle the cases in which things do not go according to the predictions. Figure 5-6 demonstrates some of these ideas. This figure shows the long-term effect of process improvement on reliability and demonstrates that the organization has gained a level of manageability and control. Over the years, not only has the average error rate decreased, but the band between the high and low values has narrowed. As a result, planning and controlling the quality of the software being produced have become much easier for managers within this organization.
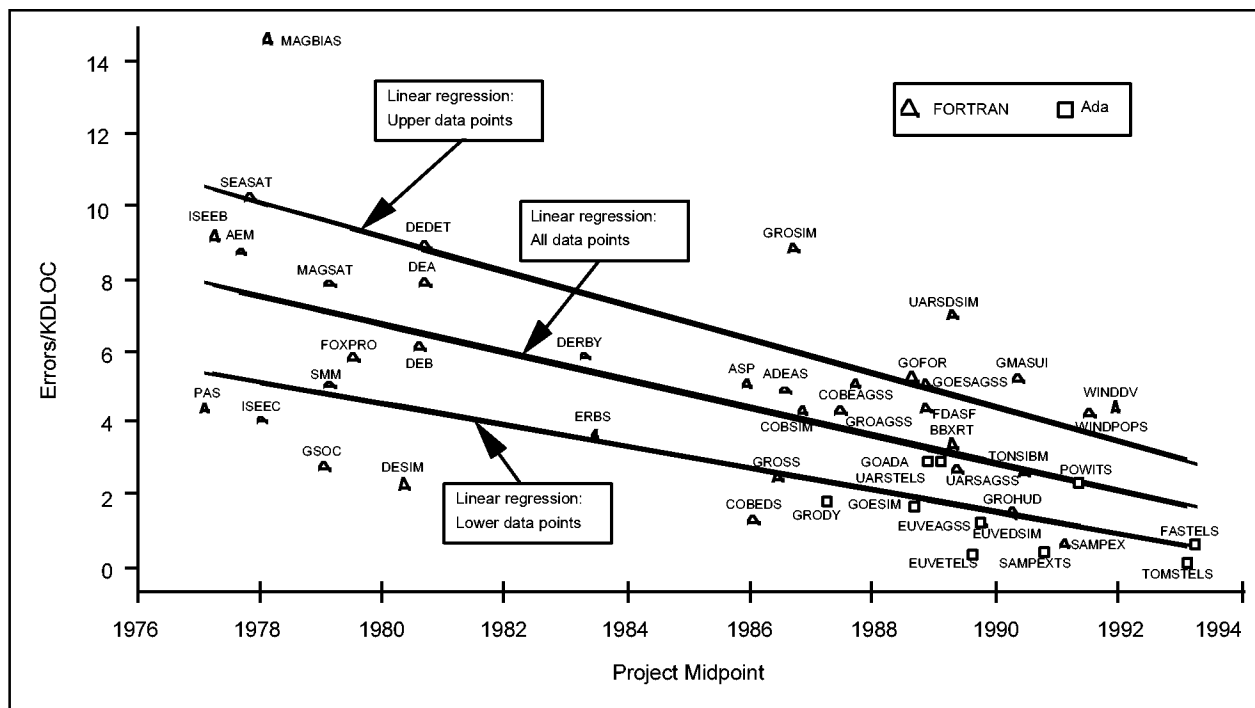
Figure 5-6. Long-Term Reliability Trends

Finally, especially in industry, there is a strong economic motivation for the organization to have process improvement programs and prove the maturity of its process. Within the CMM framework, for instance, continually improving (Maturity Level 5) organizations are much more competitive and likely to win Government contracts than those that operate in an ad hoc fashion (Maturity Level 1). In increasingly competitive markets, organizations without process improvement programs will be at a disadvantage to those with process improvement programs in place.

5.3   Key Management Guidelines

NASA's experience in implementing software process improvement programs has resulted in a list of recommendations for any organization that wants to create a software process improvement program. Some of those recommendations and lessons learned have already been introduced in Chapter 4. A short summary is provided in Table 5-2.

The issue of limiting the scope is based on two considerations. First, because process improvement represents, as shown in this chapter, a limited but significant overhead, quickly achieving the expected benefits is necessary to obtain the support needed to continue the program. Achieving success by applying corrections that can be effective very quickly proves easier in a smaller environment. For instance, suppose too many measures have been selected for

Table 5-2. Key Lessons in Starting a Process Improvement Program

Do	Don t
Limit scope (start small).	Assume bigger is better.
Specify who will analyze and package (separate from developers).	Assume developers can also do packaging.
Produce specific products the first year (concepts, baseline, perceived changes, software process handbook).	Focus on collecting data. Ignore experience or perception of developers. Promise more than you can deliver.
Assure developers that they drive change (analysts are helpers).	
Proceed slowly.	

implementation, and the data collection costs turn out to be larger than expected. Changing the standards and reorganizing the data collection process according to the original intentions are easier in a smaller environment. Second, delimiting is easier in a well-defined application and organizational domain. Because the predictive value of models and experiments is directly related to the appropriateness of the chosen domain, success is easier to achieve if this choice can be well delimited and characterized. Adhering to this principle will help keep the cost of the first phase (Understanding) low.

Developers cannot be expected to perform the analysis and packaging for the software process improvement organization. The packaging of experience is based on tenets and techniques that are different from the problem-solving approach used in software development. Developers design systems and solve problems by decomposing complex issues into simpler ones, based on a divide and conquer approach. Their goal is to deliver a system that satisfies the needs for which it has been designed within the time and budget constraints established at the beginning and corrected during the project lifetime. Experience packaging, on the other hand, is performed by unifying different solutions according to domain specificity and observed similarity. The goal is to extract from a project enough information to be able to effectively reuse the development experience in a different context within the same domain.

Therefore, the process improvement organization itself is composed (primarily) of two different suborganizations, each one with specific goals, products, and measures of success. The goal of the development organization is to deliver a software system, whereas the goal of the analysis organization is to analyze and package experiences into a form useful to the development group. The success of the development organization is measured by delivering on time and within budget a software product that meets the needs for which it has been designed. The success of the analysis organization is measured by its ability to provide and use in a timely way products, processes, and information that can be used by the developers. Every product from one side (the analysts) is derived from the specific experiences of the other side (the developers).

The separation between analysts and developers should not mean that the experience and perception of the developers are not taken into account. Their experience feeds process improvement, and their cooperation makes it useful and worth the investment. Although the analysts perform the major part of the work in collecting and packaging data and information, the developers are the driving force of change. They need to recognize the value of the support they receive from the analysts. If the perception exists that the activities of the analysts absorb

resources without providing real value, the whole initiative will likely fail. Therefore, it is important to get the developers' acceptance and support by aiming at their real problems, minimizing the effort on their side, and still giving them decisional power over change. The developers may consider measurement an annoyance, but this is not a big problem. The important thing is that measurement is not perceived as a threat. As long as a manager ensures that measurement will never be used to rate programmers, the developers will treat measurement responsibilities as just one more task that is part of their job.

The last issue associated with the successful development of a process improvement program is the need, on one hand, to proceed slowly and, on the other hand, to produce specific products (concepts, baseline, perceived changes, software process handbook) as soon as possible, perhaps even in the first year of the initiative. The personnel in charge of the initiative should be able to define a set of capabilities and products that will be ready and available for the organization at specific milestones in the program. These capabilities and products, such as providing a characterization of the organization's technical baseline or repeating the characterization process carried out in a specific domain in other contexts, should present in the most practical way the value that the process improvement program provides for the whole organization.

Appendix A. Glossary of Terms

This appendix contains definitions of terms used throughout this document and provides synonyms commonly used for these terms.

Analysis Organization: The organization whose focus and priority is to support process improvement by analyzing experience drawn from the development organization. The analysis organization synthesizes the information in the form of policies, standards, training materials and, in general, models of the product and of the process (both formal and informal).

Synonyms: Analysts, analysis and packaging element, analysis element.

Assessing Phase: The second of the three phases in the software process improvement paradigm where some change is introduced and the impact of that change on both software process and product is then determined. This phase is generally thought of as the experimental step in which some defined change to the process (e.g., the use of a new set of standards, the introduction of a new design technique) is evaluated against the baseline.

Synonym: Experimentation.

Development Organization: The organization, including all the developers and maintainers, whose primary objective is to produce software that meets the customer's needs on time and within budget. Additionally, this organization must provide development information to the analysts. It receives process and product information from the analysts and reuses this information in its activities.

Synonyms: Developers, maintainers, project element, project organization, experience source, source of data.

Packaging Phase: The last of the three phases in the software process improvement paradigm where changes that have produced satisfactory results are incorporated into the mainstream of the organization. The analysts develop new models, documents, standards, and training materials based on what has been learned during the Assessing Phase. The products developed by the analysts are stored by the support staff into the experience base and are provided to the developers upon request.

Repository: The "corporate knowledge" of an organization consisting of a projects database and a library. The projects database (usually a RDBMS) contains the historical project data (e.g., cost, schedule, and error data). The library consists of the data collection forms, project-related documentation from the developers, and the products of the analysts such as models, reports, standards, policies, and handbooks.

Synonym: Experience base.

Software Process Improvement: The continual and iterative improvement of both the software process and products through the use of project experiences.

Synonyms: Software product improvement, software process and product improvement.

Software Process Improvement Organization: An organizational structure devoted to continually using lessons, data, and general experience from software projects to ensure that ongoing and ensuing efforts use the experience to improve their software products and processes.

Synonyms: Experience factory, experience factory organization.

Support Organization: The focal point for all the archived information produced and used within the software process improvement program. This group is responsible for collecting, quality assuring, storing, retrieving, and archiving the data drawn from the developers. This organization maintains the repository of development information and packaged experiences.

Synonyms: Support staff, support element, technical support, repository component.

Understanding Phase: The first of the three phases in the software process improvement paradigm where characteristics of the software process and products are continually captured within the project organization. Models, relationships, and general descriptions of the process and products are generated. Understanding is the required starting point of the overall process improvement sequence, and it is unending because changes must be also understood and characterized.

Synonyms: Baselining, characterizing.

Abbreviations and Acronyms

AGSS	attitude ground support system
CDR	critical design review
CM	configuration management
CMM	Capability Maturity Model
COBE	Cosmic Background Explorer
DBA	database administrator
DBMS	database management system
DLOC	developed lines of code
DSN	Deep Space Network
EOS	Earth Observing System
EUVE	Extreme Ultraviolet Explorer
FAST	Fast Auroral Snapshot Explorer
GOES	Geostationary Operational Environmental Satellite
GQM	Goal/Question/Metric
GRO	Compton Gamma Ray Observatory
GSFC	Goddard Space Flight Center
IRM	Information Resources Management
IV&V	independent verification and validation
JPL	Jet Propulsion Laboratory
KDLOC	one thousand developed lines of code
KLOC	one thousand lines of code
KSLOC	one thousand source lines of code
LaRC	Langley Research Center
MSLOC	million source lines of code
NASA	National Aeronautics and Space Administration
OOT	object-oriented technology

Polar	Global Geospace Science Polar Spacecraft
QA	quality assurance
R&D	research and development
RDBMS	relational database management system
RTOP	Research Topic Operating Plan
SAMPEX	Solar, Anomalous, and Magnetospheric Particle Explorer
SEAL	Software Engineering and Analysis Laboratory
SEI	Software Engineering Institute
SEL	Software Engineering Laboratory
SEPG	Software Engineering Process Group
SLOC	source lines of code
SORCE	Software Resource Center
TOMS	Total Ozone Mapping Spectrometer
TS	telemetry simulator
UARS	Upper Atmosphere Research Satellite
WIND	Global Geospace Science Wind Spacecraft

References

1. "The Software Engineering Laboratory—An Operational Experience Factory," V. Basili, F. McGarry, et al., *Proceedings of the Fourteenth International Conference on Software Engineering*, Melbourne, Australia, May 1992

2. *Profile of Software at the National Aeronautics and Space Administration*, NASA Software Engineering Program, D. Hall, R. Pajerski, C. Sinclair, and B. Siegel, NASA-RPT-004-95, April 1995

3. *Software Engineering Laboratory Relationships, Models, and Management Rules*, W. Decker, R. Hendrick, and J. Valett, SEL-91-001, NASA/GSFC, February 1991

4. "A Methodology for Collecting Valid Software Engineering Data," V. R. Basili and D. M. Weiss, *IEEE Transactions on Software Engineering*, November 1984

5. "An Analysis of Defect Densities Found During Software Inspections," J. C. Kelly, J. S. Sherif, and J. Hops, *Journal of Systems and Software*, Volume 17, Number 2, February 1992

6. "Cleanroom Software Engineering," H. D. Mills, M. Dyer, and R. C. Linger, *IEEE Software*, September 1987, pp. 19-24

7. *The Cleanroom Case Study in the Software Engineering Laboratory: Project Description and Early Analysis*, S. Green, et al., SEL-90-002, NASA/GSFC, March 1990

8. "Software Process Evolution at the SEL," V. Basili and S. Green, *IEEE Software*, July 1994

9. "Impact of Ada in the Flight Dynamics Division: Excitement and Frustration," J. Bailey, S. Waligora, and M. Stark, *Proceedings of the Eighteenth Annual Software Engineering Workshop*, SEL-93-003, NASA/GSFC, December 1993

10. "Impacts of Object-Oriented Technologies: Seven Years of SEL Studies," M. Stark, *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, September 1993

11. "Criteria for Software Modularization," D. N. Card, G. Page, and F. E. McGarry, *Proceedings of the Eighth International Conference on Software Engineering*, New York: IEEE Computer Society Press, 1985

12. "Comparing the Effectiveness of Testing Strategies," V. R. Basili and R. W. Selby, *IEEE Transactions on Software Engineering*, December 1987

13. *Software Engineering Laboratory (SEL) Cleanroom Process Model*, S. Green, SEL-91-004, NASA/GSFC, November 1991

14. "Domain Identification for Optimizing Software Reuse," V. Basili, L. Briand, and W. Thomas, *Proceedings of the Nineteenth Annual Software Engineering Workshop*, SEL-94-006, NASA/GSFC, December 1994

15. *Capability Maturity Model for Software, Version 1.1*, M. Paulk, B. Curtis, M. Chrissis, and C. Weber, Software Engineering Institute, Carnegie Mellon University, CMU/SEI-93-TR-24, February 1993

16. *Software Measurement Guidebook*, NASA Software Engineering Program, M. Bassman, F. McGarry, R. Pajerski, et al., NASA-GB-001-94, August 1995

17. *Software Engineering Laboratory Database Organization and User s Guide (Revision 2)*, L. Morusiewicz and J. Bristow, SEL-89-201, NASA/GSFC, October 1992

18. *Profile of Software Within Code 500 at the GSFC*, NASA Software Engineering Program, D. Hall and F. McGarry, NASA-RPT-001-94, April 1994

19. *Profile of Software at the Goddard Space Flight Center*, NASA Software Engineering Program, D. Hall, F. McGarry, and C. Sinclair, NASA-RPT-002-94, June 1994

Index