Technical Report
CMU/SEI-95-TR-017
ESC-TR-95-017

Raytheon Electronic Systems Experience in Software Process Improvement

Tom Haley
Blake Ireland
Ed Wojtaszek
Dan Nash
Ray Dion

November 1995

Technical Report

CMU/SEI-95-TR-017
ESC-TR-95-017
November 1995

Raytheon Electronic Systems Experience in Software Process Improvement

Tom Haley
Blake Ireland
Ed Wojtaszek
Dan Nash
Ray Dion

Raytheon Electronic Systems

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

(signature on file)

Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

This document is available through Research Access, Inc., 800 Vinial Street, Pittsburgh, PA 15212.
Phone: 1-800-685-6510. FAX: (412) 321-2994. RAI also maintains a World Wide Web home page. The URL is http://www.rai.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center / 8725 John J. Kingman Road / Suite 0944 / Ft. Belvoir, VA 22060-6218. Phone: (703) 767-8222 or 1-800 225-3842.]

Table of Contents

List of Figures

List of Tables

Foreword

This report describes the work of the second winner of the IEEE Computer Society Software Process Achievement Award. This award was jointly established by the Software Engineering Institute (SEI) and the IEEE Computer Society to recognize outstanding achievements in software process improvement. It is given annually, if suitable nominations are received at the SEI on or before November 1 of any year. To obtain further information about the award, contact the award coordinator at the SEI.

For the 1995 award, the nominations were received and evaluated by a review committee consisting of Vic Basil, Barry Boehm, Manny Lehman, Bill Riddle, and myself.

As a result of the review, the committee selected the Software Engineering Process Group of the Raytheon Equipment Division for an on-site visit. Based on the professional presentation and their comprehensive improvement data, Raytheon was selected as the 1995 award winner. As a condition of the award, one or more representatives of the winning organization write an SEI technical report on the achievement. This is that report.

Many organizations have found that the lack of adequate data on the costs and benefits of software process improvement is a significant deterrent to their progress. This award thus emphasizes both quantitative measures of process improvements as well as their significance and potential impact. While no single improvement approach will be appropriate for every organization and while process improvement methods will evolve, the broad availability of such explicit improvement information should be of broad and general value.

The granting of this award does not imply endorsement of any one improvement approach by the IEEE Computer Society or the SEI. The award committee does, however, endorse the excellence of the work described in this technical report.

Watts S. Humphrey

Chairman, Award Committee

Preface

Raytheon began a coordinated effort to improve the way software was being developed in the fall of 1987, driven primarily by the need to overcome problems with programs overrunning schedules and budgets and the turmoil brought about by key personnel being moved in crisis priority to these programs. We recognized that the Software Engineering Institute (SEI) process maturity framework provided a feasible road map of gradual improvements which seemed to address our needs, while also lending guidance as to the sequence in which these improvement steps could be taken. As with any complicated endeavor, the initial planning required to solidify an approach took longer than we expected. Not until August 1988 were we ready to start the program, which we called the Software Engineering Initiative, and refocus mostly existing (discretionary) funds to carry us through the end of the calendar year. Subsequently, the initiative's achievements and updated plans were presented to senior management on an annual basis, justifying their ongoing sponsorship, which continues today at a rate of approximately $1 million per year of discretionary funding.

The Software Engineering Initiative has been a significant factor in Raytheon's success in delivering quality software and systems on budget and on schedule over the last several years. We also believe that these successes have had a direct effect in making the company more competitive and helping to get contract awards on new work that involves significant amounts of software. For these reasons, we see the IEEE Computer Society Award for Software Process Achievement not as a pinnacle, but as another milestone in our quest to be the "best of the best" simply because it makes good business sense.

We have measured the effects of improved process achieved by the Software Engineering Initiative in very concrete terms that can be used to make hard business decisions. In addition, the most important initial effect of the initiative has been to make software development a predictable process, thereby expediting overall program successful completion with subsequent system delivery and operation. Over the lifetime of the initiative, rework involved in building software has undergone a reduction from about 40% of the development cost to about 10%. During this same period, productivity of the development staff has increased by a factor of almost 2.8, and predictability of their development budget and schedule have been reduced to a range of +/- 3%. Our ability and willingness to analyze the impact of the initiative in these business-oriented terms has greatly influenced our success in maintaining the ongoing sponsorship of senior management.

This report provides a brief history of the initiative, the organization within which it operates, and the tailoring of the Capability Maturity Model (CMM) to make it work within the Raytheon company culture. It describes some key elements of the process improvement model that we evolved over the lifetime of the initiative, and discusses our approaches to counter traditional risks associated with process improvement programs. The report covers some of the leverage

points that contributed to the initiative's success and some of the challenges for the future. It also describes in detail the quantitative results of measuring the impact of process improvement in the terms described above: reduction in rework, increased productivity, and improved program predictability.

Additional information about this report can be obtained by writing to

Dan Nash or Gary Wolf
Software Engineering Initiative Manager
Raytheon Electronic Systems
528 Boston Post Road
Sudbury, MA 01776
e-mail: raysei@raytheon.com

Raytheon Electronic Systems Experience in Software Process Improvement

Abstract: The Software Engineering Process Group (SEPG) of Raytheon Electronic Systems (RES) is responsible for defining and implementing the Software Engineering Initiative, which outlines the policies, practices, and procedures to be followed in developing complex software for large-scale commercial and defense projects. To accomplish these objectives, the SEPG has had to develop the organizational structure and techniques to meet the growing challenges of developing, maintaining, and improving its software engineering process in significant and measurable ways, including quantifying return on investment (ROI) and increasing the quality of the deliverable product.

1 Background

1.1 History

Raytheon Company is an international, high technology company that operates in four business areas: commercial and defense electronics, engineering and construction, business aviation, and major appliances. With 1994 sales in excess of 12 billion dollars, Raytheon ranks 52nd in sales and 30th in profits on the *Fortune 500* list. Raytheon Electronic Systems (RES), focuses on the commercial and defense electronics, and is responsible for roughly 30% of the company's sales. RES is a recently consolidated organizational entity that emerged from the restructuring of Raytheon's defense business in 1995.

The recipient of this software process achievement award is our former Equipment Division, which is now a major component of the new RES organization. The software development organization within the Equipment Division was known as the Software Systems Laboratory (SSL) and is now a major component (approximately half of the 1200 software engineers) of the Software Engineering Laboratory (SEL) within RES. Throughout this report, all references to Equipment Division and SSL now reflect the recently consolidated organization as RES and SEL respectively.

Software is a major element of virtually all complex electronic systems and products we develop. These include air traffic control, vessel traffic management and transportation systems, digital communications systems, ground-based and shipboard radar systems, satellite communications systems and terminals, undersea warfare systems, command control systems, and combat training systems. The software itself tends to be real time in nature; much of it is performance critical and is tightly coupled to hardware.

As with most system houses like Raytheon, software came on with a rush during the 80s. A good part of the functionality of our complex systems that had historically been implemented in special-purpose hardware was inexorably moving into software. By the latter part of the decade, the software component had become so pervasive within RES's products that software problems quickly translated into contract performance issues, some of them of significant proportions. In the fall of 1987, prompted by the lack of success in delivering software projects on schedule and within budget, the software organization performed an assessment of its own software development process using the Software Engineering Institute's (SEI's) capability assessment questionnaire. The Level 1 results led the SEL manager to initiate a process improvement effort known as the Software Engineering Initiative. The initiative has since been the focal point for improving RES's software engineering process and a channel for institutionalizing knowledge of software engineering methods and technology as well as the policies, practices, and procedures that document the process.

1.2 The Business Case for Software Process Improvement

Funding to support the Initiative began in the fall of 1988 with a refocus of mostly existing discretionary funds. It is an important lesson that funding was not new dollars, but an improved use of existing funds. The credibility of the SEI process maturity structure with its road map of evolutionary improvements was sufficient rationale to justify continuation of the software process improvement program for about a year. At that time, many of our customers were beginning to show an interest in using the SEI process maturity framework as part of their source selection criteria, and our actual performance to contract seemed to be improving. In the one-two year time frame, new software projects began showing increased predictability. Software requirements were completed before software architecture and top-level design began, and peer-level design reviews and code work-throughs were yielding early positive results. In addition, we continued to be concerned about whether our $1 million annual expenditure of discretionary funds was *really* achieving a return sufficient to justify not spending that money some other way.

Beyond the improved predictability of software, we wanted quantitative measurements of improvements (as fits the Raytheon company culture). To address this challenge, we initially selected an approach for measuring return on investment (ROI) based on a solid business goal of reducing the amount of rework involved in developing software. The approach was based on an adaptation of the work done by Phil Crosby [Crosby 84] in the area of cost of quality. We later supplemented this approach with analyses of software productivity (a secondary, but growing benefit) on projects, and used cost at completion/budget (CAC/Budget) to measure the predictability of our software development effort on program performance. For the measurement of overall software product quality we used defect density analysis. We continue to use all four of these measures (cost of quality, productivity or cost, predictability, and overall product quality) today to monitor the impact of the software process improvement (SPI) program.

1.3 Process Improvement Strategy

We developed an overall strategy for improving the software engineering process to be used on projects within RES early in the history of the initiative, and the strategy remains in force today. This process is illustrated in Figure 1. Referring to the figure, the organization's standard software engineering process is defined by an underlying RES software policy describing the set of common software engineering practices (created by selecting our best practices across projects) describing the "whats" of developing software, detailed procedures describing the "how" of critical aspects of software development, along with the tools and the training needed to make the developers productive. A process database is an integral part of the equation, providing both metrics and a baseline for comparing future accomplishments, along with a repository for project-related products, such as lessons learned, that can be applied to future developments. Key to Raytheon's approach is overlaying process management and improvement teams with project engineering teams.

Any particular project (the dashed box in the figure) uses the organization's process, consciously tailored to its particular needs and constraints (Figure 1, step a) along with its own project software development plan, the key document binding the project to the process. The plan is typically constrained by the contract, the statement of work, and the requirements of the system as specified by the customer or developed early on during the system definition. As the project software engineering occurs and the specific process is applied (Figure 1, step b), two types of project feedback take place (Figure 1, step c). At the project level, the software development plan is refined to reflect lessons learned in early phases of the development, and at the organizational level, these lessons learned will have an impact on the process improvement activities and eventually lead to the creation of generic solutions to be added to the organization's standards (Figure 1, step d). In the meantime, the process improvement activities being conducted by the initiative, as illustrated in the lower left box in the figure, are benefiting from the real time application of these solutions on projects. The project feedback along with outside influences such as technology drivers, the marketplace, corporate business decisions, and customer initiatives, all have an impact on the direction in which process improvement will occur.

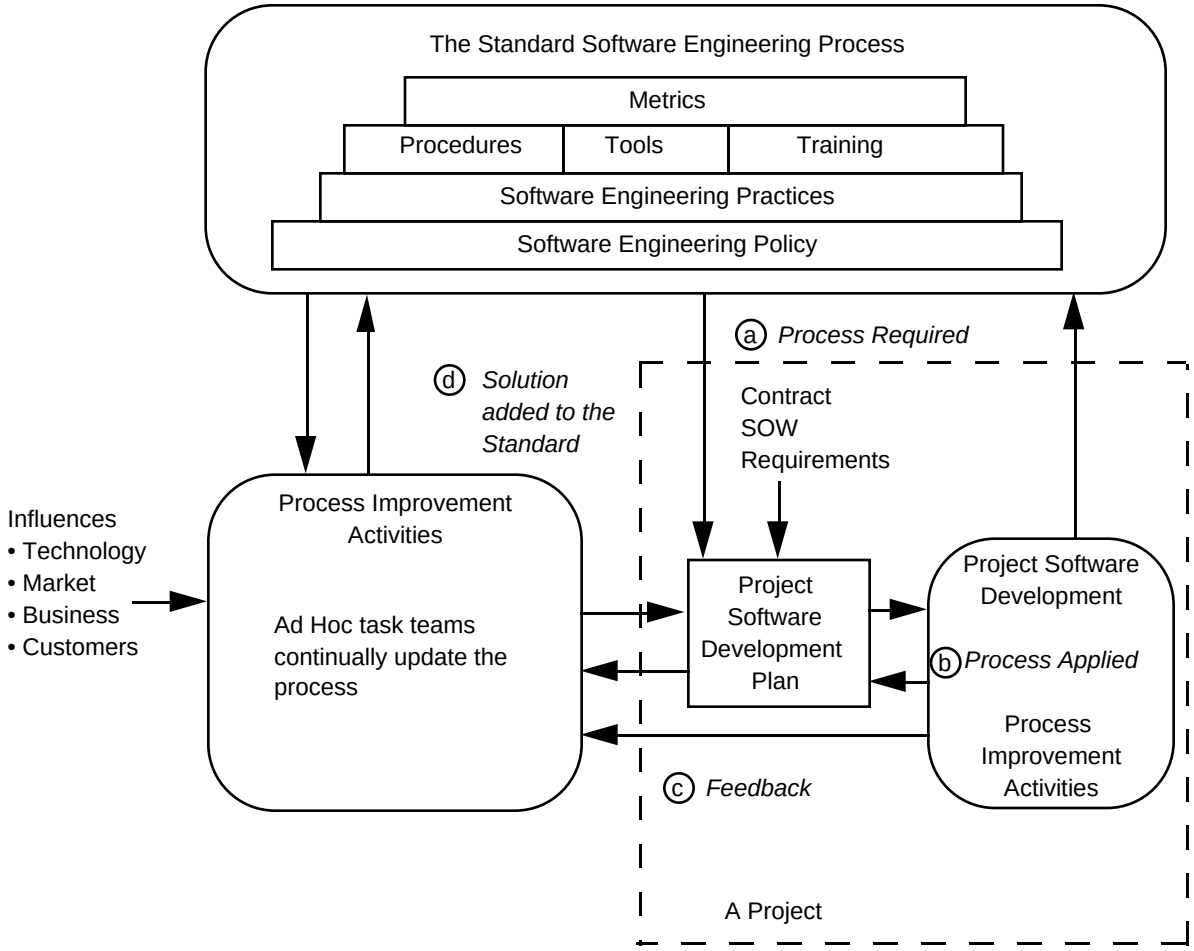


Figure 1: Process Improvement - The Process

At a higher level of abstraction the strategy involves elements that span the entire corporate structure. Organizations other than RES within Raytheon that deliver software participate in the process and gain leverage from its results.

2 The Raytheon Electronic Systems (RES) Organization

Raytheon Electronic Systems (RES) is a relatively new entity (January 1995), having been formed from the consolidation of its three Government Group divisions (Equipment, Electromagnetic, and Missile Systems). The software engineering organization within RES is made up of over 1200 software engineers. RES is a matrix organization made up of various business areas (or Directorates), which have the task of getting new business in a specific application domain, and a number of engineering functions (or Laboratories), which have the task of developing these new systems.

The Directorates provide project management and system engineering functions, while the laboratories act as subcontractors in their own areas of specialization. From the standpoint of software, one of the more challenging aspects of the Directorates is the diversity of the applications that must be supported. They include air traffic control, vessel traffic management and transportation systems, digital communications systems, ground-based and shipboard radar systems, satellite communications systems and terminals, undersea warfare systems, command control systems, and Top Gun Training systems. RES also includes missile product lines, most notably the PATRIOT missile, which was successfully deployed in the Gulf War.

Software to support the above applications is engineered by the staff of about 1200 software engineers that make up the Software Engineering Laboratory (SEL). Other laboratories which provide engineering services in their own area of specialization are Communication Systems, Computer and Displays, Mechanical, and Radar Systems. A typical development project involves most, if not all, of the above laboratories, as well as such support functions as product assurance, and engineering resources, and, of course, manufacturing.

The list of software projects that are active at any time might typically contain 200 individual ongoing projects. Of the total, half of the labor is on large projects (500K delivered source instructions [DSIs] to 2-3 million DSIs). The computer languages used for software development are predominantly “C” and Ada, with limited FORTRAN and assembly language. The target hardware (on which the developed code operates) ranges from workstations and microprocessors to personal computers (PCs) and a dwindling number of mainframes. The duration of projects varies from a few months to 2-3 years to 10+ years product line evolution. Staffing ranges from as few as 4 on some small projects to as many as 50 people on the largest projects to 300 people for product-line support, including mission software for the missile and radar, communication and training systems. The application software domains align with the business areas identified above.

3 The Process Improvement Model

Our model of process improvement embodies three critical activities: (1) establish a strong and effective infrastructure and maintain its enthusiasm over time, (2) identify the risks and develop a mitigation strategy for each, and (3) measure and analyze project data in order to determine the benefits of software process improvement. Each of these three areas is described below.

3.1 The Infrastructure

The general form of the infrastructure that we originally envisioned for our SEPG is shown in Figure 2 and consisted of four entities: an executive committee to provide direction and oversight, working groups specializing in each of the major disciplines involved in process improvement, task teams to develop the actual process changes that achieve the improvements, and an SEPG manager to monitor and coordinate day-to-day progress. The organizational structure has stood the test of time and continues today in this form.

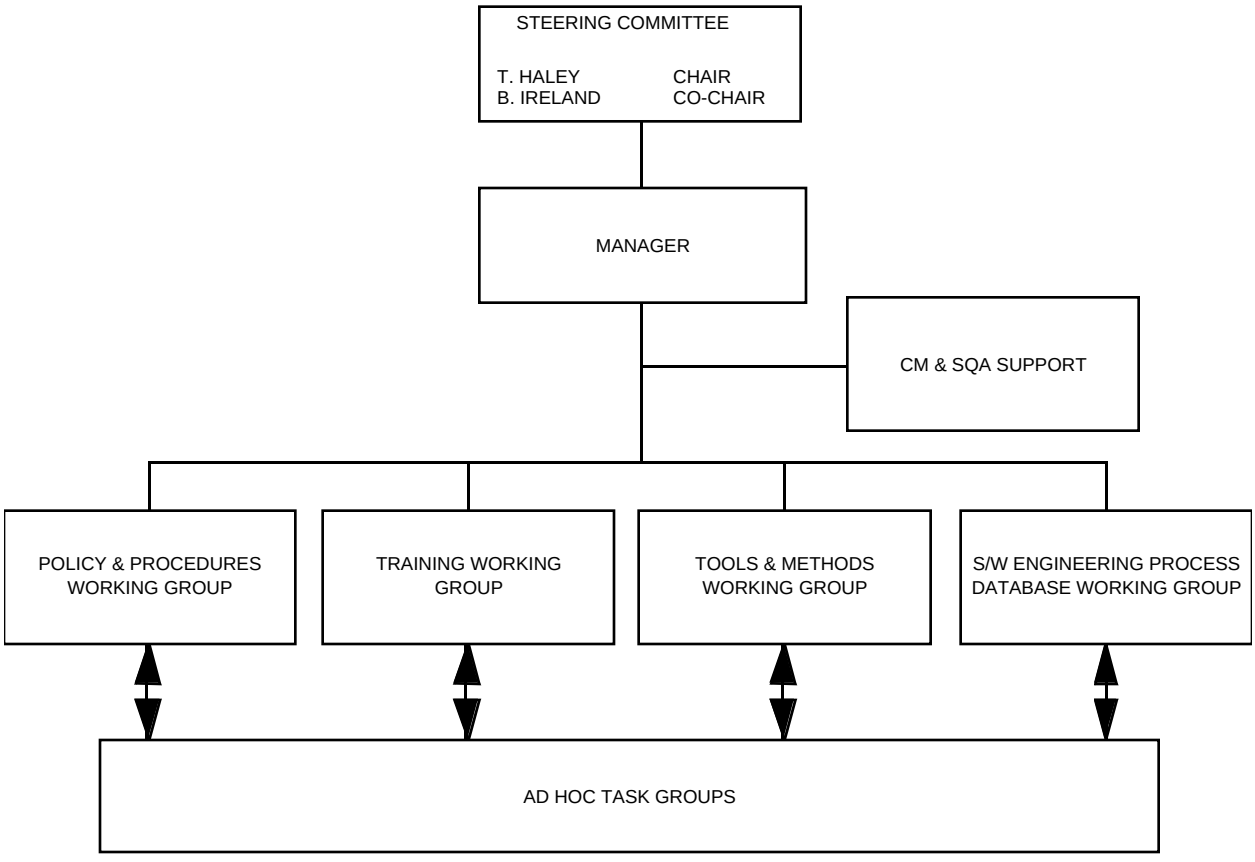


Figure 2: Software Engineering Process Group

3.1.1 The Organization

Executive committee - The composition of this committee is critical to the success of the initiative. The chairperson is the software engineering manager responsible for all software development, with the senior software individual on RES staff as his deputy. The co-chairpersons of the working groups are members of the executive committee, as are the SEPG manager and senior representatives from the software quality assurance (SQA) and configuration management (CM) organizations. The committee meets at least once a month and reviews the status and plans of each working group in the context of the long-term improvement of the organization's standard process and short term individual projects' tailored process. Perhaps one of the most important tasks performed by the committee is the adjustment of working group task priorities and reallocation of budgets based on process improvement needs of specific projects. In addition, the committee reviews any initiative risks that might be recognized by the members as needing management action.

Working groups - The Co-chairpersons of each of the four working groups (policy and procedures, training, tools and methods, and process database) were recruited from the line management organization based on their skills and interest in the specific discipline that the individual groups required. General membership in the ad hoc task groups (typically 12 - 15 people) was determined by the following selection criteria: (1) opinion leaders who were respected by their peers, (2) skills and interest, not mere availability, and (3) representation from diverse parts of the organization. Although membership was achieved primarily by recruiting, volunteers were sought and placed into positions where they could provide help where it was needed. Descriptions of the four working groups are provided below.

The primary function of each working group is to define process improvement tasks needed in its own area, and to create task teams to perform the tasks. This typically involves developing a written task definition, scope, desired schedule and funding requirement. This written task description generally provides enough specifics to allow prioritization, permit the identification of who might be best qualified (and available) to perform the task, and define the requirements that the task team will use to get started. The working group next needs to recruit the individuals who have been identified to perform the task since they are generally not members of the working group. Since most of the line managers are involved in the initiative, this is not as hard a job as it may seem. Once the task team is underway, the working group monitors their performance, provides direction, and removes any obstacles to progress.

Task teams - Each task team consists of a group of people with a team leader working on a specific software process improvement activity with a schedule and budget that have been defined by a working group. Their participation in the activity is generally on a part-time basis, in parallel with their normal project responsibilities. The team members generally have diverse backgrounds and experience, and come from different parts of the software organization. The benefit of this diversity is that the outcome typically does not represent a point solution, but one that will satisfy the diverse needs of the many parts of the software organization.

During the interval that a task is being actively worked, the members are considered to be part of the SEPG and are paid by the discretionary funds available to the initiative. The work is not expected to be *gratis*, and is in fact the major expenditure the initiative has. Working group meetings (which are generally weekly) and steering committee meetings (mostly monthly) are typically held at lunch time and attendance is expected to be *pro bono*. Once the task team has completed the activity for which they were responsible, they disband and lose their membership in the SEPG. Since there are usually 10 to 15 tasks going on in parallel, the typical size of the SEPG at any time might approach 100 people. As Raytheon adopted Total Quality Management (TQM) in the early 1990s, the task teams and working groups adopted TQM action approaches to their operations.

SEPG manager - When the initiative began, the SEPG manager was the only full-time position in the SPI organization. The function required is that of managing the day-to-day activities and individual funding accounts allocated to each task, coordinating the working groups, and facilitating the many meetings involved. As the initiative matured, two full-time staff were added: one to lead the extensive training program that evolved, and one to manage the data repository.

3.1.2 The SEPG Working Groups

The Policy and Procedures (P&P) Working Group developed the documents describing our process. Starting from the individual best practices employed by the various projects underway in all parts of the organization, the P&P Working Group extracted a set of “best-of-the-best” practices to become the starting point for our organization-wide standard practice. We also leveraged on existing standards like DoD-STD-2167A and on the process-related work being done at the SEI, using the 1987 SEI report initially [Humphrey 87] and later, the Capability Maturity Model[sm] (CMM[sm][1]) for Software [Paulk 93].

Over time, the P&P Working Group developed a three-tiered set of documents (referred to as the “blue books”) which define our process as follows:

- at the highest level (8 pages) - Software Engineering Policy
- at the intermediate level (200 pages) - Software Engineering Standards
- at the lowest level (550 pages) - Detailed Procedures and Guidelines

These documents are described in more detail below in Section 3.1.3. The P&P Working Group’s responsibility now is to continue to update these documents as needed (reflecting their use on projects) and to maintain configuration control over the documents.

1. CMM and Capability Maturity Model are service marks of Carnegie Mellon University.

During 1994, the P&P Working Group reviewed over 180 software trouble reports (STRs) written against these Blue Books. They were reviewed and the ones approved incorporated in Revisions D and E. Our software development process applies to all programs, but is tailored for each program. When tailoring is required, it is done through a methodical waiver process under the direct approval of the Software Systems Laboratory manager on an exception-by-exception basis. Although specific elements of our process have contributed substantially to our success (for example, CASE tools, requirements solidification, pathfinding, inspections or peer review), the key element is the full support, or "buy-in," to our improvement process by all software engineers. This buy-in represents a commitment at all levels of the organization.

The Training Working Group (TWG) developed a comprehensive training program. We have trained over 4200 students since 1988 including approximately 800 in 1994. All courses are given during working hours, which promotes the feeling of company support and enhances morale. Trainers are recruited from the engineering staff. The recruits take a "train-the-trainers" course and sit in on a course before they begin to teach it. Overview courses provide general knowledge about some technical or management area and detailed courses focus on a tool or technique (see "The Training Program," Section 3.1.4). A detailed feedback questionnaire is completed by the students at the completion of the course. Further, during the transition phase, process improvement discussions (PIDs) examine the effectiveness of the training provided.

The Tools and Methods Working Group implemented a comprehensive set of CASE tools available to the projects' software development/management team. In addition to supporting development, the CASE tools are used to capture the software requirements and design data and automatically publish the resulting documents. By policy, training courses are available for tool users prior to the time they are needed on the project. In addition, this working group established the Raytheon Advanced Prototyping, Integration, and Demonstration (RAPID) Lab and supported prototyping and demonstration efforts on many programs. The RAPID Lab also provides us our commercial off the shelf (COTS) and nondevelopmental item (NDI) evaluations.

The Process Database Working Group established the Process Data Center, a repository for project and process data and a source for root cause analysis, specific recommendations for local process improvement, and recommendations for general improvements to the standard process. As shown in Figure 3, the Process Data Center is the repository for both project data and metrics used in root cause analysis followed by improvement recommendations to both on going projects and the overall process.

It is interesting to note that the original decision to form four working groups with the charters described above was based on our analysis of the SEI's "Method for Assessing Software Engineering Capabilities of Contractors." Having identified the key practices at SEI Levels 2 and 3, where we believed that additional emphasis was needed, we formulated tasks to address the deficiencies and then grouped them into categories in order to assign responsibility. We found that four categories were sufficient: (1) policy and procedure development and docu-

mentation; (2) training development and conduct; (3) tool and method evaluation and transition; and (4) process data collection and analysis. Until just recently, we maintained the original identity of our four working groups and were able to comfortably fit all new tasks that we identified into this infrastructure. At the present time, the maturation of our process brought on by the changing environment (see Chapter 6) is causing us to take a fresh look at that infrastructure.

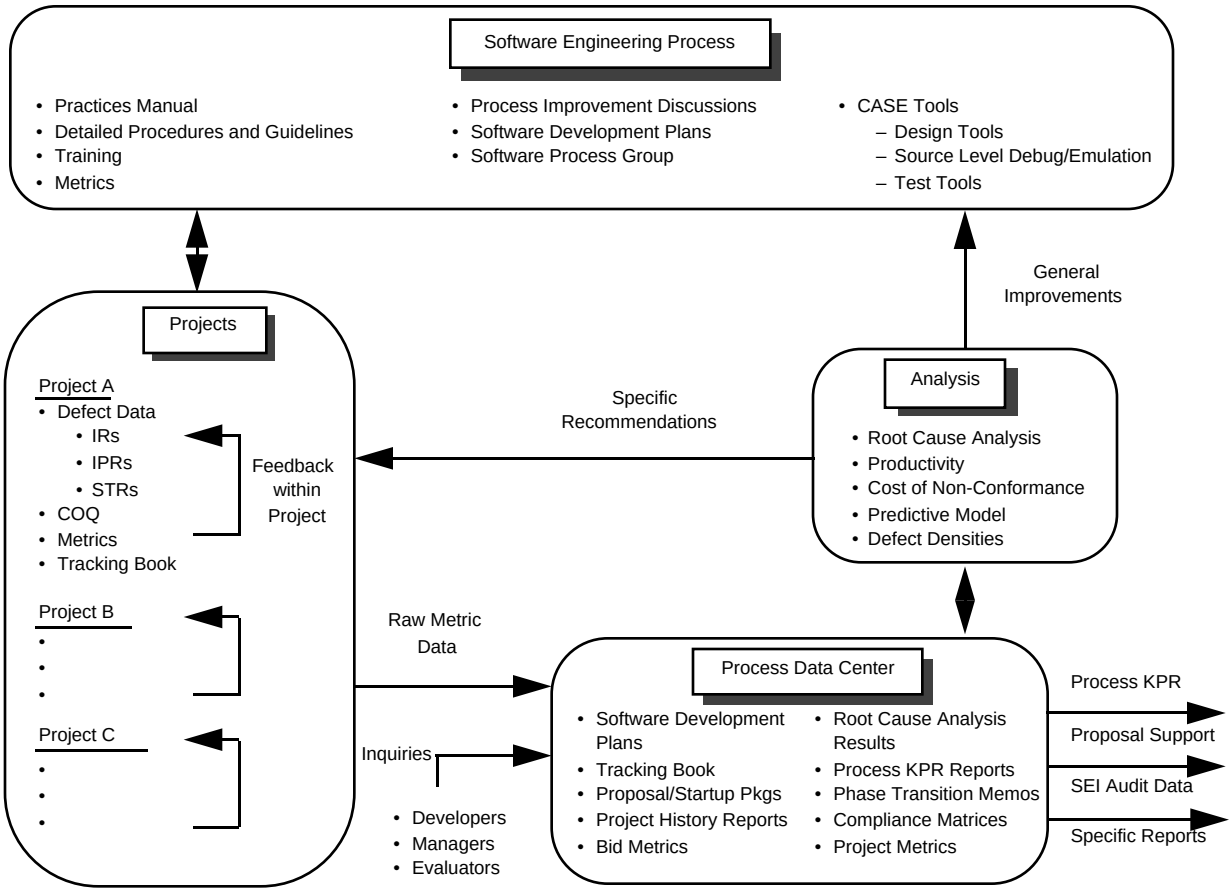


Figure 3: Metrics - Collection, Analysis, and Feedback

3.1.3 The Documented Process

The organization's standard process is documented at three levels of detail.

At the highest level, the *Software Engineering Policy* defines the objectives of the process, the role of the SEPG, the classes of software to which the policy applies, the mechanism for obtaining waivers, and the responsibilities of all organizations with which the software development organization must interface. Procedurally, the policy specifies such requirements as tailoring and review of software development plans by the SEPG, establishing a requirements baseline prior to proceeding with design, training in process as well as the tools and methods

to be employed on the job, the collection and reporting of metrics data, participation of software personnel in the systems engineering process, and selection of the host and target environments according to very specific guidelines. Although the document is only eight pages long, it has a tremendous impact on every project, because any proposed deviation is reviewed by the SEPG prior to starting the project. Recommendations for or against the proposed deviation are forwarded by the SEPG to the RES general manager for final approval.

At the intermediate level, the *Software Engineering Practices* provide a wealth of practical, easy-to-find information on the “whats” of developing software in RES. It covers all life-cycle phases of development, from pre-proposal to delivery and vaulting, specifying the inputs, process steps, and outputs of each phase. It covers the software task management elements of the work, as well as the software configuration management and quality assurance aspects. It also covers the engineering support practices that every software development project requires in order to ensure that

- Suitable resources for software development are provided.
- Training in tools and methods is available.
- Tools and techniques are investigated before their use is required.
- Potential risks and problems are identified early via prototyping.
- Process improvement reviews are held at the project phase transition points as a means of achieving continuous process improvement.
- Project requirements are traced from proposal stage through testing.
- Existing software is reused when possible, and mature COTS tools are evaluated to satisfy requirements to the maximum extent possible.
- Qualified tools and development methods are incorporated to enhance productivity.
- Quantified statements of project status, process, and product quality are maintained, and quantified data are periodically analyzed.
- A method of problem reporting and resolution is established.
- The Process Data Center is provided with all the relevant material called out in the Practices Manual.

At the lowest level, a series of *Detailed Procedures and Guidelines* is provided to address the ”hows” of developing software in RES, covering not all aspects of the process, but those which have been found to be needed in areas that were problematic, or were significant leverage points in terms of productivity or quality enhancements. The list of currently available detailed procedures and guidelines includes

- Requirements Traceability
- Software Costing and Scheduling
- Software Sizing and Timing
- Thin Specifications

- Metrics
- Program Design Language (PDL)
- Detailed Design and Coding Inspection
- Software Development Files
- Critical Design Reviews
- Ada Coding Standards
- C Coding Standards
- FORTRAN Coding Standards
- Unit Testing
- Software Integration
- Regression Testing
- Software Maintenance
- Firmware for Hardware CIs
- Use of Commercial Off-the-Shelf Software
- Prototyping and Internal Development Program (IDP) Software

3.1.4 The Training Program

The Training Working Group (TWG**)** is responsible for the development of training, student selection, course conduct, course quality, and feedback of results. The student selection process is initiated by students or their supervisors based upon project needs, individual requests, and our curriculum plan tied to various job levels. Both task and line management are involved in endorsing student participation, either for a project-specific, “just-in-time” class, or for a more general, regularly scheduled class to support professional development. At the end of each course, students complete a course evaluation form measuring the content of the course, the instructor’s presentation, and how well the course objectives were met. The instructor’s viewpoint is captured via a separate evaluation form. Modification of course material or improvements to the instructor’s classroom skills are made where necessary. Each employee’s annual review contains a process improvement section to assess how training has helped them in their work. If quality improvements are necessary in a course to benefit an employee’s performance, their supervisor reports this to the TWG. Task managers can evaluate their personnel who use training “just in time” to support their progress. Training benefits are being included as an agenda item for our process improvement discussions.

Overview courses include the following:

- *Introduction to Software-Development Methods*, software engineers, 12 hours: Lecture/discussion course on software-engineering methodology. Using our “Blue Book” (The Software Engineering Standards -Practices Manual) as a textbook, the life-cycle phases are taught in the context of our policy and practices document.
- *Software Testing Overview*, engineers and managers, 8 hours: Testing activities in all phases; compares testing practices of DOD-STD-2167A, RES’ standard process, and the SEI’s Capability Maturity Model.
- *Management Overview of Ada*, program/task managers, 8 hours: Ada structure and terminology, lessons learned.
- *Fundamentals of Ada*, middle-level task managers, group and project managers, 20 hours: Lecture course on how to read, but not write, Ada source code and how to participate in code reviews.
- *Introduction to Database Techniques*, engineers and managers, 4 hours: Introduction to database concepts and techniques.

Detailed courses include the following:

- *Formal Software Inspections*, software engineers, 14 hours: Lecture, discussion, and a practicum which involves conducting a project-related software product inspection (requirements, top-level design, detailed design, or code).
- *SRS/IRS Generation Using CASE tools*, senior engineers, 32 hours: Structured analysis for requirements, using CASE tools; generating associated DOD-STD-2167A documents, such as software-requirements specifications and interface-requirements specifications.
- *Design Techniques Using CASE Tools*, senior engineers, 32 hours: Teaches preliminary and detailed design using 2167A approaches and implementing their design using a CASE tool.
- *Software Configuration Management*, engineers, 12 hours: Standard policy and procedures, tools, utilities, and resources.
- *Software Unit Testing*, engineers, 12 hours: Covers when and how to perform unit testing, techniques and tools, preparation of unit-test plan and procedure, conducting analysis of testing, and test-exit criteria.
- *Software CSCI/System Testing*, senior engineers, 16 hours: Teaches those responsible for planning, managing, and performing CSCI and system-level testing the methods for preparing for and conducting structured system testing.
- *Software Project Management*, development managers, 40 hours: Skills and techniques to develop software within budget and on schedule. Also included are lessons learned from previous projects.

- *Software Sizing and Timing*, senior engineers, 8 hours: Estimation techniques used for proposals and in early development, emphasizing prototypes; tracking techniques, with examples.
- *Software Engineering with Ada*, engineers, 32 hours: Lecture-lab course on fundamental data structures and how to use Ada packages.

The working group is currently developing courses which cover a broader range of software engineering topics including object-oriented methodology, costing and scheduling, measurement and root-cause analysis, and software integration.

3.1.5 Tools and Methods

The Tools and Methods Working Group (TMWG) is responsible for the selection and evaluation of software development methods and tools. The working group initiates efforts to investigate new methods or tools based on project needs—whenever possible well in advance of actual project usage.

The emphasis of the TMWG has been pathfinding new technology areas. New tools and methods offer opportunities for increases in software development productivity. New approaches also present potential development risks to a project. For this reason, Raytheon's software development process relies on tools and methods pathfinding.

Pathfinding is the process of identifying and testing tools, methods, and software components prior to their required use on a project in both the development and test environments. Pathfinding is a key component of risk management because problems can be identified early, and mitigation steps can be taken before there is schedule or cost impact to the project. Pathfinding is also important for establishing and evaluating the effectiveness of new tools and methods.

A major accomplishment of the TMWG was the pathfinding and development of methods and tools for using structured analysis and structured design. The TMWG developed specialized templates and scripts to integrate a CASE tool with the document processing tool for automated production of project documentation including the system/segment specification (SSS), system/segment design document (SSDD), software requirements specification (SRS), interface requirements specification (IRS), software design document (SDD), and interface design document (IDD). This approach is now part of our standard development process and is supported by in-house training courses.

More recently the TMWG has extended this approach to support object-oriented analysis and design using commercial CASE tools.

Other early efforts of the TMWG developed specialized tools in areas where commercial products were not available. Examples of these tools include configuration management tools, a defect tracking tool, and a requirements traceability tool. All these tools are in widespread project use. Today there is a strong emphasis on commercial off-the-shelf (COTS) tools. Recent pathfinding efforts have resulted in the use of commercial products for configuration management and automated regression testing.

The importance of COTS technology goes beyond the traditional software development tools. COTS application generators and embedded COTS products are key technologies for many of our current and future projects. The Raytheon Advanced Prototyping, Integration, and Demonstration (RAPID) Lab was established to evaluate and pathfind these kinds of COTS technology and to develop methods for rapidly building COTS-based applications. COTS tools evaluated by the RAPID Lab include graphical user interface (GUI) builders, geographic information systems, database management systems and related tools, expert system shells, and networking technology such as DCE and CORBA. In addition, a COTS-based rapid prototyping method, originally developed by Raytheon under the PRISM Program at Hanscom Air Force Base, has been enhanced and applied to several new and proposed programs that require incremental, evolutionary software development processes.

3.1.6 The Process Data Center

The Process Data Center (see Figure 3) provides three very important services for the organization: management of the project data in the repository, specialized processing of the data in response to inquiries from various users, and support for root-cause analysis. Each of these services is described below.

Projects provide the following types of project-related software data to the Process Data Center:

- *thin specs* - a summary version of the software requirements. This documentation is required at proposal time, to ensure that we know enough about the project to put together a sensible bid.
- *thin SSDD* - a summary version of the system/segment design document. This document is required at proposal time, to capture the functional allocation to hardware configuration items (HWCIs) and computer software configuration items (CSCIs) as well as the system and software architectures.
- *final approved cost forms and basis of estimate* - provides a record of what the proposed software costs were and what the basis was for arriving at those costs.
- *final proposal review packages* - provide a permanent record of the changes made to the bid package as it made its way through the various levels of management review.
- *startup packages* - prepared after contract award and provides an opportunity to factor in any changes which may have occurred in negotiations with the customer.

- *policy compliance data* - provides a record of any tailoring or waivers of requirements of the Software Engineering Policy which are ordinarily applied in their entirety to all projects.
- *software development plan* - provides a detailed set of plans for developing the software.
- *task work statements* - provide detailed task descriptions, serving as the contract between the developing organization and the project's program office.
- *practices compliance data* - provide an annotated checklist indicating tailoring or waivers of the detailed requirements of the Software Engineering Practices Manual which are ordinarily applied in their entirety to all projects.
- *process metrics* - prepared monthly by individual projects; these data consist of software productivity, cost of quality, cost performance index, milestone performance index, software reuse, software trouble report statistics, and requirements trouble report statistics.
- *performance metrics* - prepared monthly by individual projects; these data describe the current measured/estimated utilization of software system target resources such as memory storage capability, data processing capability, response timing limits, communications data link, and random access shared data bus.
- *progress metrics* - prepared monthly by individual projects; these data are used by software task managers to gauge the progress or current status of their project. Included are such metrics as: module development progress, design complexity, design progress, incremental release content, staffing, schedule progress, testing progress, software volatility, software size, and inspection action item statistics.
- *greensheets* - project-specific process tailoring information prepared and distributed during the course of the project to provide all team members with the latest information on a variety of subjects. Examples are: workarounds and results of prototyping efforts.
- *process improvement bulletins* - results of project specific process improvement discussions (PIDs) which are conducted at the times of transition between development phases. The PIDs are used to review the activities of the current phase (what went well and why, what parts of the process need to be improved) and those of the next planned phase (any just-in-time training needed, any process tailoring needed based on the results of the previous phase).
- *software project history report* - prepared by the software task manager at the termination of the project to document lessons learned.
- *SEI questionnaire and backup materials* - prepared by the software task manager at the termination of the project in anticipation of any future customer software capability evaluations (SCEs) or similar audits.

Examples of the types of specialized processing of repository data provided by the Process Data Center are described below.

RES has a legacy of quarterly key program reviews (KPRs) of the major projects in each business area, chaired by the RES general manager. The concept has been extended to the review of *process improvements* accomplished by each of the functional areas within RES in a format now called “Process KPR.” With the broad availability of the project data described above, and the tools in place to perform these analyses, it is not difficult for the Process Data Center to provide this service. In fact, the availability of the analytical results in the software area spawned RES-wide process KPR in the first place.

Since there are great similarities between many of the software projects undertaken within RES, it is only logical that the data from previous projects have a high utility as a source of material for proposals and startup planning on new projects. By searching the cataloged data for similar-to comparisons, pertinent information can be extracted and furnished to the proposal writer or software task manager involved in planning a new project. Although this effort is mainly manual at present, an electronically accessible index is being implemented, along with planning for storing much of the data in electronic form to provide on-line access.

As evident from the relevant project data listed above, much of the data collected are pertinent to software engineering capability internal assessments or evaluations by external agencies. The Process Data Center can supply copies of this material in preparation for the evaluation, and also during the actual conduct of the evaluation, should that prove necessary.

Finally, the Process Data Center is facilitated to support projects in producing specific reports based on user inquiries. Software developers, managers, and members of the process improvement team can request special data analyses or queries of the database that might be helpful to their tasks. Process Data Center personnel then utilize the resources available to them to respond to that request.

We believe that the major leverage point for attaining the Managed level and progressing towards the Optimizing level of process maturity is the routine and continuous analysis of the data from multiple projects. To make this happen, we are providing the personnel, funding, communications and computer resources, and necessary tools. There are two major objectives of performing these analyses: namely, to gain insight into the current process in order to make general improvements, and to support individual projects in terms of specific recommendations. Further discussion of the use of metrics is provided in Chapter 5, Quantitative Process and Quality Management.

3.1.7 Project Support

Raytheon’s standard process is tailored for specific projects, which also benefit from ”just-in-time” training, metrics, and technology evaluation and insertion. The software task managers have access to much of the raw project data that should be collected in an organization-wide software process database. Support is provided to minimize the effort required in the data collection activity. First, tools are provided to automate the collection and presentation of data.

Support is provided for using the tools, and even to gather and validate the data. Second, we are very selective about the type of data collected. It has to be helpful to the software task manager in the day-to-day management of the project, or already required to be provided for higher level management reviews.

Another potential hindrance to good data collection is the natural tendency to declare project-specific data as *private,* for fear that the data will be used against individuals on the team. People get concerned when they see how easy it would be to use the data to measure individuals and how misleading it could be unless all the circumstances were known.

To counter this tendency, feedback is provided to the projects supplying data on the results of using the data for process improvement. This makes it clear to the providers that the data are, in fact, being used to better understand and to improve the process rather than as a finger-pointing exercise.

At Raytheon, we require that all software task managers take the Software Project Management course prior to their being assigned to a project. This provides an ideal mechanism for ingraining the concepts of using the data collection tools and support that are available to help manage the project. We also solicit the participation of the entire development staff in our process improvement activities, so that everyone is aware of the focus on metrics for process improvement, not “finger pointing.”

3.1.8 Process Improvement Planning

In planning for improvements in the software development process, we adhere to the philosophy of continuous software process improvement consistent with the ideas of Deming and Juran. We view the software development activities as steps in a process which can be consistently maintained by rigorously applying a documented set of procedures in which the practitioners are trained. The effectiveness of these procedures can then be measured and analyzed with the results of the analyses continuously fed back into the process improvement effort. Consistent with our charter, the broad objectives are reviewed annually during the update of RES’s five-year plan, and a specific implementation plan for the upcoming fiscal year is put in place.

There are a number of steps to the planning and budgeting process, which begin with a set of informal meetings where the chairpersons of the SEPG Working Group construct a list of key problem areas and key successes. This step culminates with a list of possible activities for the coming year, sorted on the basis of affected working group(s). In the second step, each working group reviews the ongoing activities in its area to determine if additional support will be needed in the coming year. If so, a determination is made whether that support can be allocated to the various projects that benefit, or whether the initiative must continue to support the activity.

The third step involves the review, by each working group, of the key practices of the CMM key process areas (KPAs) for which it has primary or support responsibility. The objective of the review is to determine what SPI activities the working group needs to plan for the coming year in order to ensure that all projects can perform the specific key practices not already in place, but judged as necessary.

In the final step, each working group then combines the required activities established in the three steps described above to derive an implementation plan for the upcoming year consistent with the long-range objectives of the initiative and satisfying the needs that have been identified. The individual plans are presented to the steering committee and adjustments made based on known project priorities and budgetary constraints. This is the final plan presented for endorsement to the engineering manager, RES general manager, and vice president for engineering.

A key ingredient of the planning process is involving a large number of the engineering and management staff in the process improvement tasks. The plan benefits from the very practical experience and lessons learned on a variety of projects expressed from a number of different viewpoints. In addition, since the individuals who will implement the process changes are, by and large, the same individuals who recommended the change, the likelihood of acceptance is much higher and the conventional problems with technology transition become more tractable. The individual projects also benefit from the resulting greater level of awareness and concern with the process. Recently, our focus has been on determining, developing, pathfinding, and implementing process improvements to successfully satisfy the two key process areas (Quantitative Program Management and Software Quality Management) required to satisfy the CMM Level 4 maturity.

3.2 Organization/Process Binding

Over the course of the initiative, especially during the early days, we encountered a number of obstacles which represented potential risks to the success of the program. These were mitigated where possible, and continue to be managed in a variety of ways which bear recording as a possible benefit to future efforts.

Sponsorship and Commitment - Since the software engineering (SEL) manager was the person who drove the entire initiative from the very beginning, getting initial sponsorship and commitment at that level was not a problem. The discretionary funds available at the SEL level, although adequate to support the size of program that was envisioned, had to be reallocated in a major way. It was necessary to seek authorizing sponsorship at the next two levels (engineering manager and general manager) in order to get the necessary support for this reallocation. This was achieved by getting both levels intimately involved in the initiative planning process and by presenting a solid business case for why the SPI initiative would be effective. The draft initial plans were presented and refined iteratively at the software engineering manager level in a series of presentations, until there was agreement that we had a solid plan. The three-hour presentation at the general manager level was really more of a working session,

where the details of the SEI questionnaire responses were discussed individually and considerable feedback was provided to the initial plan. In this process, the initiative plan became a plan that was mutually “owned” by both levels of management, thus providing the necessary authorizing sponsorship and commitment to get the program underway. This participation has continued throughout the life of the program, with annual two-three hour briefings at the general manager level, where progress during the completed period and plans for the upcoming period are presented in detail.

Because of the matrix organization structure, it was necessary to get reinforcing sponsorship from the various business directorates (described in Chapter 2, above). Since these directorates are the customers of the software engineering organization, they would naturally be affected by the initiative and their input was requested. This was done by making the same two-three hour presentation to each of the business directorate staffs and incorporating their feedback into the plan. As the initiative matured, the directorates were made aware of the progress and thus were able to see the benefits to their own programs, thus perpetuating the reinforcing sponsorship and commitment.

Recruiting the right personnel - Populating the lead positions in the initiative infrastructure was made easier by the fact that the program was being driven by the SEL manager. He was able to recruit his direct-report line managers to fill most of these critical slots, based on his knowledge of their skills, interest, and their current project commitments. By making it known throughout his organization that process improvement was a high priority on his agenda and that he expected everyone to think about process improvement as part of their normal job, he was able to help the recruiting effort tremendously.

As the need for additional people increased in direct proportion with the amount of SPI activity, the fact that the line managers were part of the initiative was a critical factor. They were able to convey to the individual contributors who worked for them the importance of process improvement for the company’s future. Not only was it acceptable to spend time on process improvement activities, it was considered part of their job. The line managers were also very cognizant of which individual contributors were already temporarily over-committed and therefore knew where to draw the line in asking for SPI support. They also knew how to steer volunteers to the areas where their contributions would be most beneficial.

Resources - The issue of funding a SPI effort of the size needed to support an organization of 1200+ software engineers is clearly a make-or-break proposition. Unless the improvement program can attain a certain momentum, interest can be lost because things just take too long to change. Therefore the level of funding required can be substantial. The probability of supporting a large program on an ongoing basis with *pro bono* support alone is negligible. It is necessary to identify a source of funding that can be allocated to the SPI effort and then managed like any other project.

Typically, when a SPI effort is just being started, the funding needs are looked upon as completely unprecedented, and a source of new money is looked for. However, the organization may already have discretionary funds being expended in other areas; by redirecting these funds at least a portion of the SPI program can be financed. Such was our own experience. The SEL was able to identify discretionary funds under its own control and redirect those funds to the initiative. Senior management was then much more receptive to our requests for some additional funding to ”round out” the program. The total amount of funding for process improvement at Raytheon RES since 1988 has been approximately $1 million per year. This represents between two-three % of the chargeable labor dollars being dedicated to process improvement. Coincidentally, this is approximately the same percentage that the SEI states to be necessary to achieve a successful program [Humphrey 89].

Cultural Issues - The corporate culture (the values, behaviors, and unwritten rules that a company exhibits) has a great deal of influence on the direction that SPI programs take. In the Raytheon culture, there were significant factors that on the one hand hindered progress, but on the other hand were a tremendous help in making SPI a success. On the positive side, we were well served by a culture that contained a strong quality ethic, heavy emphasis on planning and tracking mechanisms, a strong engineering background throughout all levels of senior management, and organizational precepts that promoted authority along with responsibility while also fostering accountability. We were able to gain leverage from these strong cultural traits in putting together a SPI program that focused on these characteristics as important elements of the plan. For example, the early SPI planning stressed the need for improved quality of the software product; planning and tracking of the SPI program just like any other project; feedback from senior management on the engineering approach; and an initiative infrastructure that gave responsibility for the success of the program to many of the middle managers, while giving them the necessary authority, but also holding them accountable.

In those few instances where we encountered cultural resistance, we were able to effect changes over the long term by gradually building a convincing base of evidence that linked the desired change to the core competencies and strategic direction that provided RES with a competitive advantage. For example, we began our training program by giving courses after hours, on the individuals’ own time, and we paid for our instructors out of discretionary (overhead) funds. Over time, we were able to convince senior management that project-specific training had an immediate payback and, thus, we eventually effected a change in the culture. Our standard practices now dictate that training needs be defined on a per-person, project-specific basis; that well-structured courses be provided on a just-in-time basis; that they be given during normal working hours; and that the cost be borne by the project.

Consolidating linked efforts - One way in which SPI efforts get diluted is by the lack of a focused and coordinated effort. Many times, this is caused by turf-guarding efforts, where a case is made that some particular improvement activity is not clearly related to the ”main” SPI Initiative and therefore is kept separate. In the Raytheon initiative, the purview of the SPI initiative was occasionally extended to ensure that this would not occur. All software-related ac-

tivities that might consume discretionary dollars were brought under the umbrella of the initiative, by linking them with the ongoing efforts. This consolidation had the effect of increasing the importance of the initiative to the organization and thus solidifying management support.

3.3 Measurement and Analysis

Raytheon considers the collection and analysis of metrics to be of primary importance in managing and improving the software development process. To support the collection and analysis of metrics, we have established the Software Process Data Center (SPDC). In the SPDC, we have provided a living archive for all the process-relevant, hard-copy documentation we produce over the course of each project development effort. We also provide an electronic repository for much of the same material.

3.3.1 Data Measurement

The two categories of measurements that we have found most useful are project progress metrics and system technical performance metrics. Each category is prepared monthly by individual projects.

Progress metrics - these data are used by software task managers to gauge the progress or current status of their project. Included are such metrics as module development progress, design complexity, design progress, incremental release content, staffing, schedule progress, testing progress, software volatility, software size, and inspection action item statistics.

System technical performance metrics - these data describe the current measured or estimated utilization of software system resources such as memory storage capability, data processing capability, response timing limits, communications data link, and random access shared data bus.

3.3.2 Data Analysis

In the area of *progress metrics* the analysis varies with each of the characteristics measured. As an example of the type of analysis, we will use the first characteristic listed above. Module development progress is expressed in terms of earned value and schedule and budget efficiency.

These ratios indicate how well the program is performing in terms of schedule and budget. Raytheon uses metrics to determine the percentage completion of each of its development phases (requirements analysis, preliminary design, and formal testing). Great care is taken to ensure that we avoid the 90% syndrome (the software is always 90% complete). Where necessary, we use self-calibrating metrics. That is, the metric is the ratio of two measured quantities, the number of items completed and the total number of items.

The analysis of *system technical performance metrics* is more straightforward. At the start of each project, performance requirements are identified. These typically include maximums for software system target resources such as memory storage utilization, CPU utilization, response timing limits, communications data link capacity, and random access shared data bus capacity.

Raytheon uses a systematic method of collecting and presenting this information from all jobs with significant software content. The material is prepared by each project in RES monthly, and reviewed technically. Projects prepare a plan giving the margins by which they will manage technical performance parameters over the life of the program. Typical planned margins are shown in Table 1 and depicted graphically in Figure 4.

Table 1: Phase Dependent Margins

Milestone	Allowed Usage
Proposal submission	50%
SRS completion	56%
PDR	62%
CDR	71%
Start of Integration	83%
FQT	91%
System Test	95%

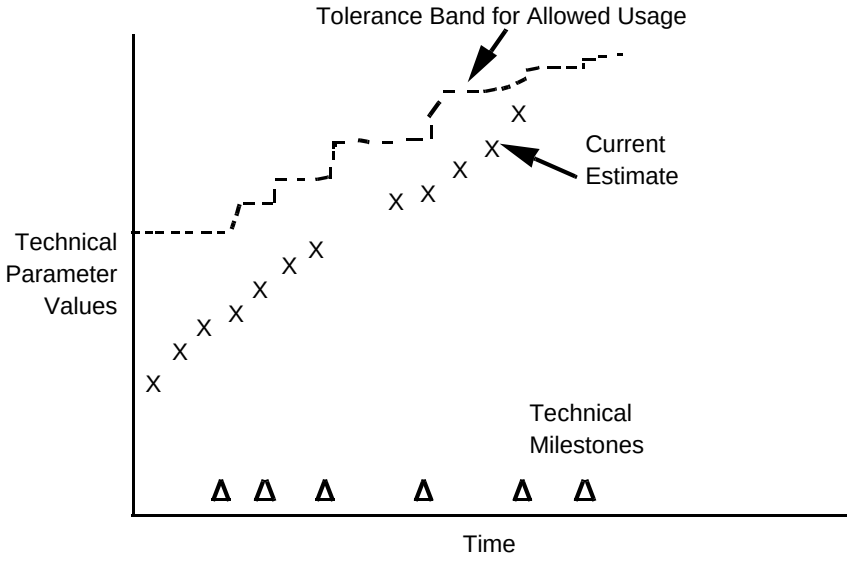


Figure 4: Performance Margins Versus Time

Projects report monthly the estimated value of each technical parameter. The estimated value of each parameter is initially modeled, and later measured. Since measurements are considered more reliable, the reporting distinguishes between these methods. The report also shows explicitly all reserves, both the development reserve and the customer's reserve. These estimates are reviewed monthly with RES management.

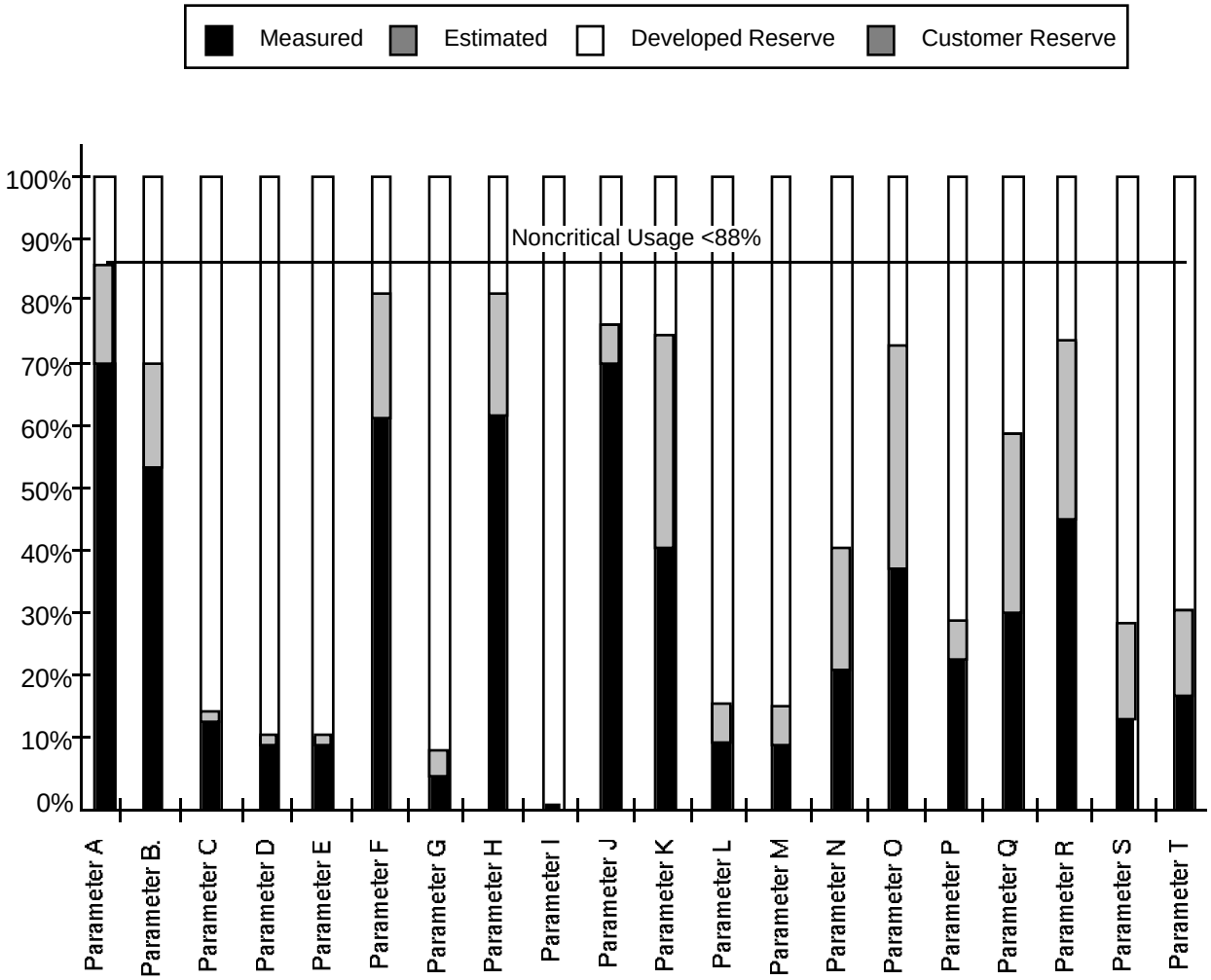


Figure 5: Sample Technical Performance Measurement Report

Figure 5 depicts a typical portion of the performance report. All parameters are normalized so that 100% represents the maximum contractual value. Parameters which exceed the currently allowable threshold are considered critical. Parameters which exceed 100% are failing.

The values of technical performance parameters are maintained in a relational database. Table 2 below shows the history of several critical parameters over a one-year period. Shaded parameters are (or were) failing. These parameters receive special management attention.

Table 2: History of Critical Parameters

Parameter Category	Parameter	1994										1993
		Dec	Oct	Sep	Aug	Jun	May	Apr	Mar	Feb	Jan	Dec
timing	A	97	97	97	133	92	217	217	217	217	217	200
comm	B	141	141	141	141	112	112	112	83	110	110	
cpu	C	77	101	101	125	91	91	91	91	121	121	
	D	77	101	101	125	92	92	92	92	121	122	
	E	91	91	91	110	71	71	71	¢66	88	88	
	F	87	87	87	95	71	71	71	14	19	19	
	G	91	91	91	110	71	71	71	66	88	88	
	H	87	87	87	95	71	71	71	14	19	19	

4 Leverage Points

In the course of implementing the Software Engineering Initiative at Raytheon, a number of key points have emerged in contributing to the sustained improvement of the software development process. These points have shown up repeatedly in different projects, specifically in the areas of product improvement and process improvement. By concentrating on these points, significant leveraging can be achieved in expediting the software development process.

Specifically, with respect to the product, the key leverage points include the areas of system requirements definition, software requirements definition (the relation between these two is of critical importance to successful system integration), in-process inspections, and integration and qualification testing. With respect to the process, the key leverage points are management control, training, and pathfinding. Each of these points is discussed in more detail in the following subsections.

4.1 Product Improvement

4.1.1 System Definition

Software engineering should play a major role in defining the system, recognizing that as our system solutions have become more and more software intensive, the overlap between system engineering and software engineering has become virtually complete. As a matter of RES policy, the group responsible for system engineering must be augmented by project personnel from the software organization at the outset of system definition.

Software engineering is typically an active participant in the early tradeoffs that precede the various partitioning steps from which a system design baseline eventually emerges. We expect our software systems people to have a working knowledge of what is happening outside their software domain; many of our people have found this to be a natural career progression path to system engineering.

The organization also expects the software engineering participants to assume a dominant role in setting the system's processor architecture. Since software is going to have to live with these architectural decisions for the life of the program, this serves to establish ownership in a timely and sharply focused way. It also causes the software participants to conceptualize a strawman system engineering architecture, which in turn provides a meaningful basis for sizing the job for planning and scheduling purposes.

Hand in hand with the definition of the target data processing environment comes the need to lay out the software development environment, in terms of both facilities and tools. For obvious reasons, software acts as the principal agent in this task, drawing heavily from the experience built up by the Tools Working Group of our Software Engineering Process Group.

Participation by software representatives also provides early and perceptive insight into the specifics of the applications domain to be implemented, thereby establishing the core around which an effective requirements generation team can be built, and enhancing the eventual software design.

4.1.2 Requirements Definition

We have learned from considerable experience, that definition of an inadequate and/or incomplete requirements baseline can bring even the best of software processes to its knees. Predictably, the software design team must then spend a disproportionate amount of its energy attempting to sort out and clarify the real requirements, and even despite their best efforts, the results at system integration time can turn out to be disappointing, at best, and extremely costly to fix at worst.

A fundamental precept of our software process initiative is to extend our process focus across the requirements definition phase, even though requirements analysis and specification at Raytheon are clearly within the purview of the system engineering function. We have found formal, structured decomposition methods well suited to our needs, particularly with the emergence and subsequent maturation of supporting CASE tools. Our methods baseline has been Yourdon/Demarco with Hatley/Pirbhai real-time extensions; our CASE tool set of choice is Software thru Pictures (StP), although our process permits the use of a variety of vendor products. With a well-honed training curriculum in place that is imparted to all members of the software requirements team (including systems people and, occasionally, the customer) we are finding it relatively straightforward to decompose software requirements in an orderly way to achieve a well-balanced functional performance baseline that is surprisingly resilient to moderate requirements change.

With the emergence of CASE technology, we have been able to achieve a convenient marriage with documentation technology, making our engineers more efficient, and their lives a lot easier, by attaining a level of tool integration that takes the onerous task of document generation out of the software engineers' hands.

Our practices require that we generate preliminary requirements (known within Raytheon as "thin specs") specifications during the proposal phase. These force the software and system engineers to describe the functional architecture well enough to allow the software to be sized and cost estimates generated in turn.

In closing out our requirements analysis phase, we do several things that appear to be unique. First of all, we recognize that there's a potential flaw in our partitioning process. That is, we allocate system requirements to individual configuration items (CI), proceed to take a vertical cut through each CI whereby requirements are decomposed and defined, and attempt to identify inter-CI interfaces as well as those external to the system. What our process adds is a final

step wherein we carry out an analysis of major system threads by taking a horizontal cut across all CIs, including both software and hardware, and, by ensuring end-to-end connectivity across each functional thread, we validate the partitioning that was put in place months earlier. If not uncovered here, the ”we forgots” will next show themselves in the system integration lab.

We also usually conduct a ”goodness” check on our software requirements specifications to satisfy phase exit criteria. We are not talking about functional correctness here; that is covered in great detail during underway inspections and reviews. Our specific focus is to have a small, independent team sample the full set of specifications, assessing such factors as the proper level of decomposition, consistency and clarity of the requirements statements, forward and backward traceability, and testability of individual requirements. This serves to put an important stake in the ground relative to expected specification quality.

4.1.3 Inspections

Surely one of the most important leverage points in our process derives from our adoption of in-process inspections. These are peer reviews that conform generally to the structure laid out by Michael Fagan at IBM in the mid-1970s. There is a formalism associated with each inspection in terms of the moderator’s role; the responsibilities of the two-three reviewers both in preparation for and as participants in the inspection; and the inspection support structure including recording, action item tracking and closure, and defect data gathering. Our process mandates 100% inspection coverage, so convinced are we of the importance of this scrutiny. We originally limited our inspections to the detailed design and coding phases, but have since extended this to embrace top-level design as well as requirements definition.

An inspection curriculum is an integral part of our training program. Course conduct is typically carried out on a project-by-project basis, with the course material tailored to a project-specific context.

Inspections have played the principal role in allowing us to identify defects at a stage where they are easy and relatively inexpensive to fix, thereby causing our process to avoid the heavy rework penalties associated with problem detection at later integration and test stages. We have dealt with the need to depersonalize this part of the process through careful schooling of moderators and reviewers in techniques for examining the product for defects, not for evaluating the author; however, objective peer review has the effect of causing the originator to place special emphasis on the correctness of his/her product.

In stepping up to the challenges of a Level 4 process, we have been quick to appreciate that inspections provide us with the sort of in-process windows needed to define the organization-wide behavior of our process model, as well as to measure an individual project’s performance against that model.

4.1.4 Integration & Qualification Testing

Our standard process separates software development from software integration, and in turn separates software integration from software qualification testing. We are also committed to the use of incremental builds and have been employing this integration strategy for well over 20 years; in many cases, our software builds' functional content and schedule are driven by a higher level build plan wherein hardware and software functionality is accumulated incrementally into a system string.

We devise a build strategy very early in our process. Our detailed development plans flow from that strategy. At any given point in time, functionality for several builds may be underway on parallel development paths. We find it effective to establish a formal hand-off point between the developers and the integrators; development teams normally pre-integrate their unit-tested code into functional sub-threads and effect a promotion to the integration baseline by discussing the extent of their testing with the build manager and demonstrating that their new code will run in the integrated build against a regression test suite. As an added benefit, we find that our developers are professionally motivated to bring code to the integration point that in no way affects the hard-won stability of the current build.

The role of the integrator is to ensure that the assembled code executes correctly and predictably; performance within the target environment receives special attention, while functional correctness is left to the test organization. The build team typically achieves a hand-off to the test team by demonstrating stable operation using a relevant set of qualification test procedures. The role of the test team is one of establishing via preordained verification methods that the software satisfies each and every performance requirement levied against it, and the subsequent demonstration of this to the customer.

There are several other facets to the integration and test portions of our process that warrant mention. Our use of incremental, and sometimes overlapping, builds has caused us to evolve rigorous, multi-version configuration control mechanisms that are serving us particularly well as we transition from one-of-a-kind government systems to systems that are productized. In addition, we find that our pathfinders usually have wrapped up their assigned tasks at about the time software integration starts, and that they provide perfect seed talent for the build teams. Finally, we have learned from painful experience that you cannot have too much in the way of facilities, especially when it comes to integration; therefore, rather than build a two-three shift operation into our usually tight schedules, we've found it cost- and schedule-effective to generously equip and instrument our integration and test laboratories.

4.2 Process Improvement

4.2.1 Management Control - Software Development Planning

The management component of our software process has its roots in a disciplined planning process. We insist on the preparation of a comprehensive draft software development plan at the proposal stage of a program that, to be process compliant, must include 20 discrete elements of planning detail as defined in our practices manual. Standard estimating methods are used, based in large measure on delivered lines of code, factored according to whether it is new, reused, COTS-based, etc. Once a project is underway, we obtain bottom-up estimates from individual staff members, thereby establishing the desired degree of accountability.

We carry out earned-value tracking on a regular basis: weekly at the project level, monthly at the engineering laboratory level, and quarterly at the RES level. We have learned that process compliance must be kept in full view and thus have made phase exit/next phase entry criteria compliance integral to laboratory-level tracking.

We regard the software quality assurance (SQA) function as a component of our management control process; however SQA is organizationally separate from the software organization, enjoying a separate reporting path to RES general manager. The SQA role is primarily focused on auditing the software process and its products; they carry out this role smoothly and effectively by using technically qualified people, having practical insight into the standard software process, and maintaining an unobtrusive presence. The result is complete acceptance by the performing organization undergoing audit.

Finally, on those occasions when it is appropriate to use software subcontractors, our fundamental management precepts call for the work to be done under our standard process and, ideally, for the work to be done on a co-located basis. We have found the SEI CMM to be a very useful vehicle for assessing the suitability of a potential subcontractor to our software needs.

4.2.2 Training

As our process has matured, we've defined a training curriculum that has stayed in lock step with it. We currently provide 18 courses covering the full range of process topics. This training baseline is subject to ongoing evaluation, and course elaboration and new course definition are a never-ending proposition.

Each new software program start is expected to lay out a training plan based on project-specific needs defined in the context of prior training accumulated by the proposed staff. Our philosophy is to present the courses on a just-in-time basis, recognizing that the material can quickly go stale if it is not soon reduced to actual practice. To this end we have a large staff of instructors, being able to draw from a cadre of five or six for each course. Our instructor posi-

tions are filled exclusively from within the software engineering ranks; our only full-time training person is the individual who has overall direction of the software training program, and who trains the trainers. We are certain that the training material is made much more meaningful by having committed practitioners present it.

We also find that training is the obvious vehicle for getting everyone on the same page; accordingly, our classes go well beyond software engineering personnel to include program managers, system engineers, SQA, and frequently, customer personnel.

4.2.3 Pathfinding

In examining how well our software engineers used their time in carrying out their assigned tasks, we discovered that their work environment frequently presented problems that slowed them down and/or called for them to use work-arounds (for example, buggy compilers and idiosyncratic real-time operating systems). To deal with this, we conceived the idea of pathfinding teams that would move out ahead of the developers to blaze the trail and cope with problems in advance. These are small teams staffed by those individuals with special talents who seem to thrive on this sort of relatively unstructured assignment, and whose success is directly proportional to the efficiency of the engineers who follow them down the various development paths.

Pathfinding concentrates on two sets of paths: those that comprise the software development environment, and those that wend their way through the target data processing environments. Work on the development side of things generally starts with getting the work stations/PCs and servers tied in with the network of choice and put through their paces. The full set of tools is then installed and checked for compatibility with the various platforms as well as with each other. Project-specific set-up is then carried out (for example, the software development library structure is established consistent with the particular configuration control strategy in use). All tools are exercised in the various ways they will be used by the developers. Tools that are new to Raytheon are given extra attention so as to define explicit procedures governing their use and identify peculiar behavior characteristics. Tools that are relatively new to the market are exercised extensively under all modes and stressing loads to benchmark performance and, perhaps more importantly, to put useful mileage on the tool set. Compilers are given special attention in this regard since their behavior has a direct impact on developer efficiency. We are also careful to accurately measure computer loadings associated with the use of specific tools; a development environment that chokes on a CPU-hungry compiler can impose schedule problems that are extremely difficult to deal with in real time.

The target environment has its own set of problems that lie in wait for the developers. Usually, the first pathfinding task is to set up the full target configuration, including the download paths from the development environment. Performance of the individual target elements is compared against what was anticipated. The tasking structure envisaged by the software architecture is put in place and exercised to verify its match to the hardware. A lot of attention is paid to the run-time operating system, especially if it is relatively new to Raytheon or is being ex-

posed for the first time to the rigors of real-time use that many of our systems demand. Pathfinding also finds itself caught up in the prototyping of performance-critical functions so as to be able to influence design decisions and make target hardware adjustments in a timely fashion.

Planning for pathfinding tends to be quite dynamic. It is necessary to put a software project plan in place at the outset that reflects an overall pathfinding strategy. As the team works its way down the various paths, new avenues of exploration are bound to open up and the team must have the flexibility to pursue them; however, because of the pathfinding team's relative independence, it is important to maintain visibility of their activities so as to ensure that adequate energy is being evenly applied across pathfinding areas.

The RAPID Lab, described in Section 3.1.2, has assumed an increasing role in pathfinding or qualifying COTS products.

5 Quantitative Process and Quality Management

An important part of the Software Engineering Initiative, and a key element in achieving the Software Engineering Institute's Level 4 rating, is the use of metrics, or quantitative methods, in measuring and evaluating the software engineering process. Once a software engineering process has been standardized, it is possible to gather and analyze data from projects using the standardized process to evaluate both the quality of software products and the efficiency of the methodology. Metrics gathered from a variety of projects employing a common methodology provide a valuable source of insight into the effectiveness of the methodology, as well as a means for providing an objective estimate of the effects of process improvements.

In order to gather and use such metrics effectively, it is necessary to develop an organizational framework and a tool set to support this activity. This includes the following steps: (1) defining which metrics are to be recorded; (2) specifying how the data are to be gathered, formatted, and organized; (3) establishing a common repository in which data from all projects can be stored and retrieved; and (4) defining procedures by which existing metrics can be analyzed to assist in process improvement.

This chapter summarizes Raytheon's approach to the quantitative measurement and analysis of its software engineering process, as well as of the quality of its software requirements as determined by conformance to requirements, and also summarizes some of the key results we have obtained to date in our progress towards achieving Level 4.

5.1 Approach

Our approach to institutionalizing quantitative management of both our software engineering process and its quality requirements involved the following activities: utilizing existing project data; determining the tools and resources necessary to expedite the collection, storage, retrieval, and analysis of data for efficient use within and across projects; establishing the organizational infrastructure necessary to support data collection, analysis, and dissemination; and implementing the proposed methods in our actual practices.

Project data collection is an integral part of Raytheon's culture. Cost, schedule, manpower, and many other project parameters are rigorously tracked and reported in a monthly tracking book, which is routinely reviewed by all levels of management. These data provided a useful starting point for the more detailed analyses that we projected as necessary to meet our goals. A large proportion of these data, however, were in hard-copy form, or else embedded in electronic word processing formats on different platforms, thus making it difficult to assimilate the data into a single common format suitable for systematic analysis.

One of the first steps in our approach, therefore, was to automate the collection of most of the data into a relational database using commercially available tools. This made it possible to perform searches, sorting, and general statistical processing uniformly within project-specific data, as well as across data from multiple projects. During this tooling effort, we also re-exam-

ined the detailed information previously collected on defects (as reported in action items generated during inspections, as well as from software trouble reports). As a result, we modified our data collection forms in ways that would enable us to perform the detailed analyses required for our root-cause studies.

In parallel with our data consolidation efforts, we augmented our Software Process Data Center's capabilities in anticipation of the more intensive activities in which it would be participating. These augmentations included additional staffing, more computer hardware and software tools, LAN connections to specific project servers, and additional physical space. We added a statistical process control (SPC) expert to our ranks to help set up the standard analyses we would be performing, and to prepare and teach a course in basic SPC techniques to team members involved in our Level 4 efforts.

Our approach to getting a higher level of involvement from software task managers also led to a change in our SPI infrastructure. We formed a Task Manager Working Group to facilitate project data collection, root cause analysis, and the feedback to enable process refinement on projects. The Task Groups that were formed to implement process change received common training in problem solving methods, and were given the title of Level 4 process improvement teams.

While the above developments were occurring, three projects agreed to pilot various elements of the approach. The success of these pilot efforts, specifically in the area of improved inspections through the use of statistical analyses and the area of process adjustments resulting from root cause analyses, prompted the software engineering manager to require that all new projects adopt this so called Level 4 approach.

Improvements in the inspection process resulted from a detailed study of the results from inspections in a number of projects. The analyses (described in Appendix A) concluded that, for the type of software developed in Raytheon Electronic Systems, the best way to operate inspection reviews under preferred conditions is to do the following whenever possible:

- Break review packages down into 1000 DSI or less.
- Allocate and schedule the review meeting length to be consistent with approximately 100 to 250 DSI per hour.
- Allow for adequate preparation time (minimum of 5 hours per KDSI) and ensure that sufficient peer reviewers are present.

In addition, the study was able to set control limits on the inspection defect-discovery density which were subsequently used by the pilot projects in conjunction with the root cause analysis efforts to achieve better inspection efficiencies and reduced project costs.

5.2 Results to Date

The gathering of metrics data and the initial analysis of the data gathered earlier in our initiative constituted the activities associated with the beginning of our Level 4 work. The next part was the root cause analysis of the information to determine the cause of problems identified by the data analysis. The root cause analysis was frequently done by teams of task managers associated with the projects, along with representatives from the Process Data Base Working Group. The final part, closing the loop by changing the process, was done by the task managers on their respective projects (where changes were project-specific) or by the SEPG, where the overall process was affected.

Key to these activities was the use of quantitative data for decision making. We deemed this to be the key identifying characteristic of Level 4 organizations. Collection and initial analysis were done earlier in our initiative, but the application of root cause analysis and the feedback loop of process change, based on unbiased data analysis, constituted our Level 4 advance. This was augmented by the addition of statistical process control techniques permitting us to focus our analysis and improvement activities on issues resulting from causes outside the normal process variations.

Table 3 provides some examples of the combined data analysis/root cause analysis/process change mechanism that we have applied.

Table 3: Examples of the Process Improvements Achieved by Root Cause Analysis

Weakness	Process Improvements Made
Erroneous interfaces during integration and test	- Increased the detail required for interface design during the requirements analysis phase and preliminary design phase - Increased thoroughness of inspections of interface specifications
Lack of regression test repeatability	- Automated testing - Standardized the tool set for automated testing - Increased frequency of regression testing
Inconsistent inspection process	- Established control limits that are monitored by project teams - Trained project teams in the use of statistical process control - Continually analyze the inspection data for trends at the organization level
Late requirements updates	- Improved the toolset for maintaining requirements traceability - Confirm the requirements mapping at each process phase
Unplanned growth of functionality during Requirements Analysis	- Improved the monitoring of the evolving specifications against the customer baseline - Continually map the requirements to the functional proposal baseline to identify changes in addition to the passive monitoring of code growth - Improved requirements, design, cost, and schedule tradeoffs to reduce impacts

Erroneous interfaces during integration and test - Analysis of software trouble report metrics, via use of Pareto analysis, showed that the largest contributor to the number of defects was interface defects. Further analysis by the task manager and process database team determined that the root cause could be attributed mostly to lack of sufficient detail during the initial design phases. As a result, we changed our process in two ways to alleviate this problem. First, we required that the level of detail of interface specification be enhanced up to and including adding actual code during the requirements and preliminary design phases of the life cycle. This was done to require more specificity in the interface definition earlier in the life cycle. Second, we modified all of our inspection checklists to place more emphasis and focus on the interface issues. Our Policy and Procedures Working Group modified the standards to reflect this and the Training Working Group helped change our requirements analysis course.

Lack of regression test repeatability - Pareto analysis of cost data for a large software project found that a major cost factor was the test area. The root cause of this was determined to be the cost of the associated set-up and execution time. Further investigation of the metrics data showed this was true across most of our product line software. Continued root-cause investigations revealed that because of this cost, regression tests were run less frequently so that when the product was tested, defects had accumulated and debugging was more difficult and costly. Again, the use of quality metrics data on defects gave quantitative measures to the extent of this problem. Process changes included automating test activities with commercial products, standardizing on a test tool set across projects, and increasing the frequency of regression testing.

Inconsistent inspection process - Our baseline data for detailed design and code inspections shows a wide dispersion of defect removal results among individual inspections. We wanted both to determine the guidelines for optimal reviews, consistent with our process, and to begin to shrink the deviation around the mean of these parameters for improved predictability. As a result, baseline numbers were established for these key variables and control limits established. A training program in statistical process control was developed and is now given to project teams (using actual project data) as their projects transition to the use of Level 4 techniques.

*Late requirements update***s** - Projects were exhibiting a large number of defects during integration and test, as revealed by analysis of defect density trends from software trouble reports. The root cause of these defects was traced to the failure of the software to fully satisfy requirements, resulting in rework to both the specifications and the software. An improved tool set was added to provide complete traceability from system level through test cases and support the confirmation of requirements mapping during inspections as each phase of development.

Unplanned growth of functionality during requirements analysis - We had been accustomed to assessing requirements growth in terms of code size. When unplanned requirements growth occurred on several projects, we examined the growth trends and traced the root cause not to misestimation of size (a common first assumption) but to growth in functions. We have changed our requirements process to now use customer specifications and the informa-

tion that is developed during the proposal process to continually monitor functional (in addition to code size) growth. Functional growth is now managed in two ways, once identified. First, a certain amount of growth is planned and tradeoffs are made to evaluate the growth in specific areas against shrinkage in others to manage the functional scope of the projects. Second, functional growth is managed as part of the program risk process, where cost and schedule impacts are identified and mitigated. These process steps are now codified in our software standards and promulgated in software management and requirements analysis courses.

As we continue our initiative, we have taken the step of requiring all new projects to operate as Level 4 projects. Within the context of Raytheon SEL's Software Engineering Initiative, we require of projects that

- appropriate metrics be collected and sent to the Process Data Center on a monthly basis
- standard analysis and reporting of these metrics be done regularly
- key quality metrics and parameters (from inspections and software trouble reports) be compared to baselines (mean and control limits) derived from the SEL metrics database
- task managers and line engineers be trained in the use of statistical process control techniques so that they can properly interpret the metrics data
- task managers be able to use SPC, along with other techniques, to be able to focus their resources on key issues, not variations within normal process limits
- project team members, in conjunction with other members of the SEPG, contribute to root cause analysis of both project and process-related problems
- project team members, in conjunction with other member of the SEPG, contribute to actions to change the SEL process

A key requirement is that the collection, analysis, and process change must have immediate benefit to the project (a goal is within a month). Raytheon's engineering culture emphasizes results showing direct, immediate benefits to projects. We have established this as a criterion for demonstrating the cost/benefit of achieving Level 4 throughout the entire software organization.

6 Changing a Mature Process

RES's Software Engineering Initiative has become the cornerstone of our future software business. Since our proposals are keyed to the software quality and productivity gains we have seen in the recent past, it is critical to our continuing success to ensure that software improvements are sustained, measured, and remain significant. This must be done in the ever-changing business environment in which we live. Some of the important developments which have dramatic effects on our developing process are (1) the migration of our process from defense-oriented programs to more commercial-based programs. This is consistent with Raytheon's evolving focus on commercial opportunities and conforms with DoD's downsizing initiatives, and (2) the migration of our process from the former Equipment Division to other organizations within and external to Raytheon (the newly organized Raytheon Electronic Systems, other Raytheon divisions, subsidiaries, new corporate entities, customers, subcontractors, and prime contractors).

In the DoD area, our process developed under the constraints of required compliance to military standards, close monitoring by the customer, and emphasis at the "software specification" level of requirements. The characteristics that we are seeing in today's environment are that reduced (or no) standards are imposed, the customer may not be actively involved during the development, and the emphasis has shifted to the "system specification" level of requirements. In short, there exist a number of drivers, in addition to our quest for institutionalizing Level 4 and achieving Level 5, that are already affecting and will continue to influence the direction in which our presently mature process will evolve.

6.1 Technology Drivers

The variety of issues that we contend with in this area is staggering. One challenge is the maturity of object-based techniques. While the use of these techniques may provide a competitive advantage, a complete transition to this approach would present significant risk to the organization. With a documented process, tool support, training programs, and metrics activities all centered around the use of functional decomposition approaches, the impact of evolving to an object approach would be significant, to say the least. On the other hand, we may be missing a great opportunity. The actions being taken to evaluate this tradeoff include the piloting use of object-based techniques on one project being funded as part of our internal research and development (IR&D) program. This presently includes training project personnel to use these techniques during the requirements phase, and the plan to extend the training in the near future to include software design.

Another technological thrust is the migration to Ada 95. With a large number of projects that have used Ada over the last several years, we have an existing base of programming guidelines, training materials, tools, and personnel that will require transition to the new standard. Having anticipated this need, we have been following the evolution carefully, and publishing periodic Ada Newsletters to keep people informed as to the changes to be expected. Plans for completing the transition are underway, so that the risk involved can be minimized.

The evolution to open system architectures is another technological challenge. Anticipating these developments over the last few years, we have been making efforts to develop our software to be platform independent to the greatest extent possible. This includes the use of open systems standards (such as SQL, POSIX, Cobra, and X-Windows) wherever practical, and the use of COTS software and hardware. To further enhance our platform independence, we are relying almost exclusively on commercial tools in areas such as test tools, file management software, and project management software. In the human/machine interface area, we are employing application builders to create graphical user interfaces which have a high degree of independence of target hardware.

Another issue that we must face is the rapid introduction of personal computers and the generally exponential growth in computing horsepower that is potentially available to our developers. Here again, this technology provides an opportunity to achieve competitive advantage, by leveraging the computing power to utilize more sophisticated tools or increase throughput. Our approach has been to carefully select management and engineering tools which are supported both by the workstations and the personal computers that are increasingly becoming our environments of choice, and by working closely with the hardware vendors. In this way, as the computer environment evolves, we stand a better chance of upgrading with minimum impact on our computational resources.

Of course, the whole area of tool and environment evolution presents a challenge to the stable process we have in place. Two approaches have been adopted: the first being the ongoing evaluation of new tools and methodology that has been the focus of our Tools and Methods Working Group since the very beginning of the initiative, and the second being the recent development of the Raytheon Advanced Prototyping, Integration, and Demonstration (RAPID) laboratory. Although the primary objective of RAPID is the prototyping of application software within a family of domain-specific software architectures, the facility has the capability of integrating COTS tools for purposes of qualification and evaluation as well.

Yet another technological innovation that has an impact on the current process is the almost limitless information access available through communication networks. Although these new communications channels (such as WAN electronic communications throughout the Raytheon organizations and Internet access to the world) have been made available to the staff, their potential for integration into the software development process is far from being fully realized.

6.2 Market Drivers

We see the marketplace in our business areas undergoing at least two distinct transitions that already have an impact on our ”mature” process and will undoubtedly continue to influence the direction that our process evolves. This new market is characterized by demands for quick and inexpensive solutions, and a strong emphasis on quality.

There are several reasons for the increasing emphasis on getting quick and inexpensive solutions:

- As competing companies continue to downsize into ”leaner and meaner” entities and scramble for the dwindling DoD market, the challenge of trimming costs to the bare bone becomes greater.
- As the computer hardware capability continues to grow at what seems to be an exponential rate, it becomes more and more important to build systems more quickly in order to take advantage of the latest capability available.
- The continuing trend toward extending the life of existing systems by upgrading rather than building from scratch drives customer’s expectations of “faster, better, cheaper.”
- As customers see the successful incorporation of non-developmental software (NDS) and COTS software and hardware into new systems, they are demanding more of the same.
- In the international arena, the softness of the dollar, the growing capabilities of off-shore software houses, and the frequent need to compete with foreign companies that are government subsidized all drive us to streamline our process to meet these cost challenges.

Some of the specific process-related actions that we’re taking to address these issues are as follows:

- Initiate IR&D projects oriented towards producing baseline capabilities.
- Develop standards for domain-specific software architectures and mechanisms for reusing large software NDS and COTS products.
- Organize resources to handle multiple projects working on similar systems for different customers during the same time frame.
- Develop heavily tailored processes for projects that have only small amounts of new software embedded within a very large NDS and COTS component.
- Gain leverage from the Raytheon Advanced Prototyping, Integration, and Demonstration (RAPID) laboratory capability in providing streamlined testing capability for these ”new” types of systems.

In fact, the system development paradigm that RAPID supports can be viewed as a specific process itself. This so-called “RAPID Process” is shown in Figure 6. The RAPID Lab’s software engineering process is composed of two distinct but intertwined activities. Domain engineering is a foundation activity in which software architectures for each principal application area are defined, developed, and documented through an integrated CASE tool environment. The attributes of the components within these architectures become the basis for periodic, methodical product evaluations conducted on nondevelopmental (COTS/GOTS) software products. All products that satisfy the attributes assigned to these components, as well as associated documentation, are captured and catalogued in a corporate reuse library for future project use. Project system engineering matches a specific set of user requirements to the ap-

propriate architecture component attributes and the products that have been evaluated and validated for use in building similar functionality. By reusing architectures and products captured in the reuse library, development teams are able to rapidly generate and deliver prototype or first-article capabilities to an end user.

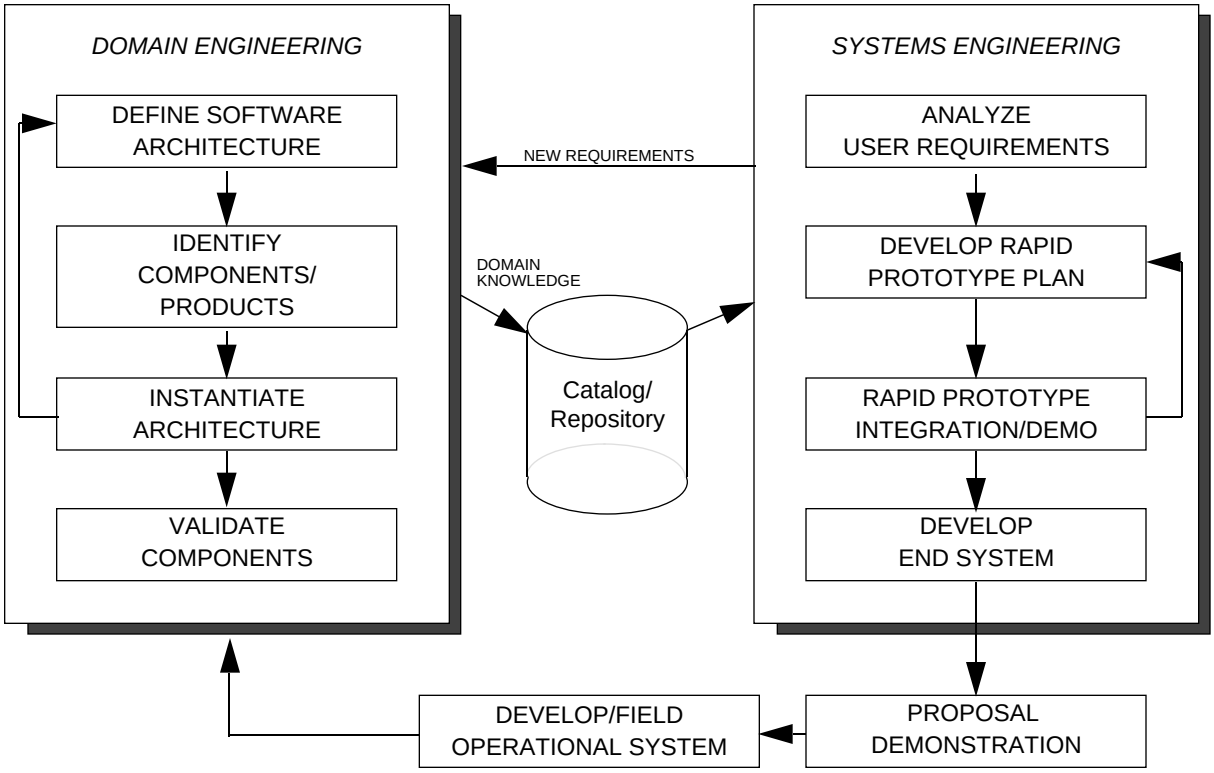


Figure 6: The RAPID Process

The heavy emphasis on quality that we see in the marketplace is characterized by requirements for quality certification, an emphasis on applying TQM techniques and, at least in the European community, ISO 9000 registration. The Raytheon quality management (RQM) approach that has been instituted across RES has led to the adoption of a uniform philosophy and mechanisms for applying problem-solving techniques. The Software Engineering Initiative has adopted the RQM approach using process improvement teams (PITs) instead of task teams for its CMM Level 4 activities. The advantage is that the PITs can benefit from the RES-provided training in different problem-solving techniques, including the use of statistical process control and approaches for addressing the root cause of problems.

In the area of ISO 9000 registration, the initiative has led the way to the achievement of the goal of ISO 9001 software certification within RES. A special training program was set up and conducted throughout the remaining software-intensive segments of RES in preparation for the audits, with the goal of achieving certification throughout the entire organization by year-end. We elected to pursue software certification separate from hardware. While the software development part of ISO registration was well served by our mature process, it was all the interfaces to the total RES structure, such as purchasing, vaulting, program management, and document preparation, that proved tricky.

6.3 Business Decisions

Some of the more recent business decisions that Raytheon has made are having an impact on the process direction that we had established. Specifically, these fall into two areas: corporate restructuring, and teaming and subcontracting.

Our recent merger of the former Equipment and Missile Systems Divisions to form Raytheon Electronic Systems, more than doubled the size of the software development organization. The standard practices of both of these major components must now be merged into a single set of ”best practices,” from which specific tailoring will be done on a project-by-project basis. Raytheon has also acquired major new business elements which have large software components. Again, the standard practices which have become part of our mature and stable process will need to be further adapted to the business areas of these newly acquired businesses.

Raytheon has recently embarked on programs with complex teaming arrangements involving multiple companies with varying degrees of software engineering capability maturity. In some cases, Raytheon is the prime contractor and in others a subcontractor. Some success has been achieved in working with team members in both environments to have them adopt the Raytheon process for the entire contract team. In other cases, the division of responsibility on the contract has been, in part, based on process maturity, with the development of a single standard process for that contract being performed specifically for the award.

6.4 Customer Initiatives

The evolution of our current process is influenced by the initiatives that some of our customers are taking. These initiatives include the Portable, Reusable, Integrated Software Modules (PRISM) work with the Air Force, the Perry initiative affecting all of our DoD customers, and the increased reliance on process maturity in the selection process being used by many of our customers in both the Government and commercial sectors.

Raytheon is one of the two main contractors for PRISM. The rapid prototyping development approach being emphasized by this customer initiative is influencing other work, creating a distinct shift in the processes needed to comply with the customer requirements. This shift has necessitated the development of new processes, thus helping us continue to evolve our process. The approach has such great potential that we have embraced the technology and transitioned many of the concepts to our own Raytheon Advanced Prototyping, Integration, and Demonstration (RAPID) laboratory, and to some of the programs we are currently bidding.

The Perry initiative, relaxing the use of military standards and emphasizing the use of best commercial practice, provides an opportunity to use more effective, less costly approaches for developing software while still maintaining the level of rigor necessary to ensure a quality product. This has the potential for further streamlining our process. Alternative solutions can now be proposed which involve greater use of commercial software products with resultant savings in both cost and time. The flexibility this initiative provides to our DoD customer base can now be leveraged by proposing highly innovative approaches.

6.5 Benchmarking

As Raytheon continues to strive to be the “best of the best,” we have increased our participation in benchmarking. By searching out those industry best practices that can complement our own, we expect to improve our overall process [Camp 89]. Our approach is to run some benchmarking pilots with several companies; in doing so, we have two goals in mind: finding process elements that we can incorporate, and developing our own internal benchmarking process that we can make part of the Raytheon standard.

7 The Impact

In the six years since the Software Engineering Initiative was put in place, Raytheon has been able to demonstrate sustained, significant, and measurable improvements to its software engineering process. Given the diversity and complexity of Raytheon's projects in both commercial and defense electronics, this is a noteworthy achievement. Our success has enabled us to remain competitive in the shrinking defense marketplace, to expand into some promising commercial markets, and to grow substantially.

The gathering and analysis of metrics instituted as part of the Software Engineering Initiative (see Chapter 5) have enabled us to monitor the impact of the Initiative. In particular, the impact has been assessed in the following areas: cost of quality, software productivity, cost performance index, overall product quality, benefit to other organizations, and benefits to personnel. Each of these areas is described below.

7.1 Cost of Quality

The cost of quality, as defined by Phil Crosby, is the extra cost incurred because a product or service was not done right the first time. Crosby defines the cost of quality as a sum of two components: the cost of nonconformance (CONC) (or rework) and the cost of conformance (COC), which is made up of appraisal costs and prevention costs. Nonconformance costs are all those direct and indirect costs associated with reworking a product or service because it was not done right the first time. Appraisal costs are associated with evaluating or testing the product or service to determine if it is faulty. Prevention costs are those derived from the proactive steps taken to reduce or eliminate rework.

In applying these concepts to software engineering for purposes of analyzing project costs, we found it necessary to add a fourth major category, which we call the performance costs. These are simply all those costs which are absolutely necessary in developing the software product even in an error-free environment. In other words, this is the cost of doing it right the first time. We felt that by using this fourth category, we would be able to better define the subcategories to which all project software costs would be allocated.

Defining the subcategories of each main category (performance, rework, appraisal, and prevention) for our software development environment was not an easy task. First of all, the work breakdown structure used on software projects did not map well to the cost-of-quality categories. Second, the definitions of each subcategory, which were rather brief for reasons of simplicity, were subject to misinterpretation.

We addressed the first problem by adopting both a short-term and a long-term solution. In the short term, project costs would continue to be collected using the conventional work breakdown structure, and project leads would, periodically, manually remap all costs to the cost-of-quality subcategories. In the long term, a common work breakdown structure would be developed to provide as close a mapping to the cost of quality as possible. This would also entail a revision of the cost accounting system, and possibly the time-card reporting system as well.

The second problem was addressed by refining the definitions as we gained experience in using them. This required five iterations of the initial data-gathering exercise before we obtained a satisfactory level of consistency. The result was the subcategorization shown in Figure 7.

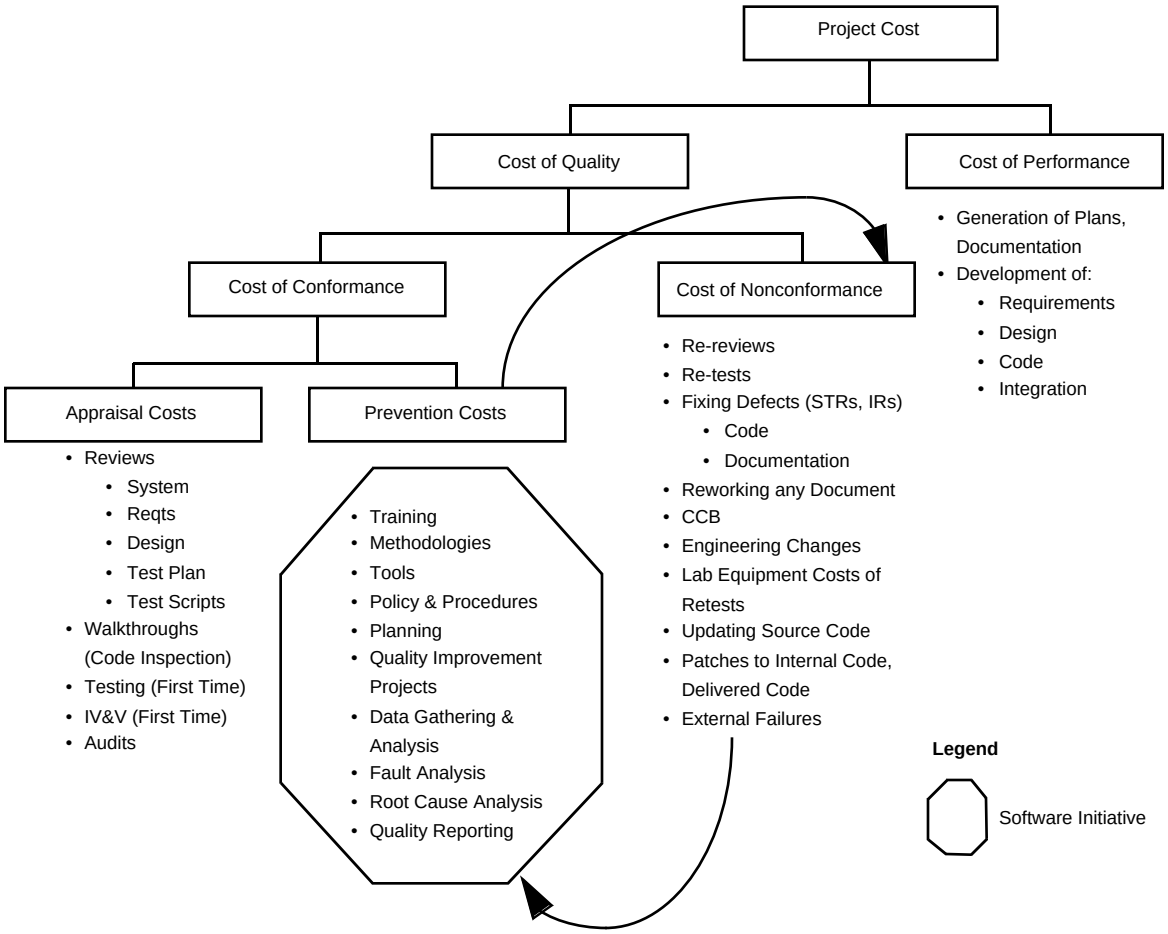


Figure 7: Cost-of-Quality Model

As shown in the cost-of-quality model the subcategories of the prevention cost are those associated with the process improvement program. The advantage of using this particular model is that it could embrace both process improvement costs and project costs. Thus we should be able to see interaction between the two and perhaps establish causal relationships.

The initial application of the cost-of-quality model used six large ongoing projects, mainly because the six projects employed 80 to 90% of the software engineers. The six project leaders used the model's cost subcategories to allocate their project's actual costs so that all costs were accounted for. As predicted, there were many questions as to which subcategory particular project costs were to be allocated, and quite a variation in the algorithms employed to break the actual costs down to the defined "buckets." These issues were resolved by refining the subcategory definitions, and by analyzing and comparing the suballocation algorithms employed by the six project leaders. In the latter case, this involved a search for anomalies in the cost data across projects, followed by some research to determine if the anomalies could be rationalized, or if they were caused simply by differences in the interpretation of definitions or in the suballocation algorithm employed.

One cause of difference that we uncovered dealt with the fact that two of the six projects were nearing completion, whereas the remainder were in various early stages of development. In order to compare data between projects, it was necessary to extrapolate all costs to project completion.

Another lesson that we learned during this initial exercise is that it is important to have the project leader generate the data rather than an administrator. In order to make an accurate sub-allocation of actual project costs, one must possess first-hand knowledge of project particulars as well as good engineering judgment. Of course, this should not be necessary in the long term when costs are automatically collected in the appropriate cost-of-quality subcategories.

Once the iterative process of analyzing and rationalizing project-variant data was complete, the individual project data were combined by weighting on the basis of project staffing and then normalizing. The composite results for the six projects were then expressible as average percentages of total project cost. For purposes of the analysis, process improvement program costs were factored by considering the initiative as a seventh project which had only prevention costs and no appraisal, rework, or performance costs.

The combined data showed that the average cost of rework or nonconformance had decreased following the start of the initiative (see Figure 8). In the two years prior to the initiative, the rework costs had averaged about 41% of project costs. In the two years following, that value had dropped to about 20% and the trend was continuing downward.

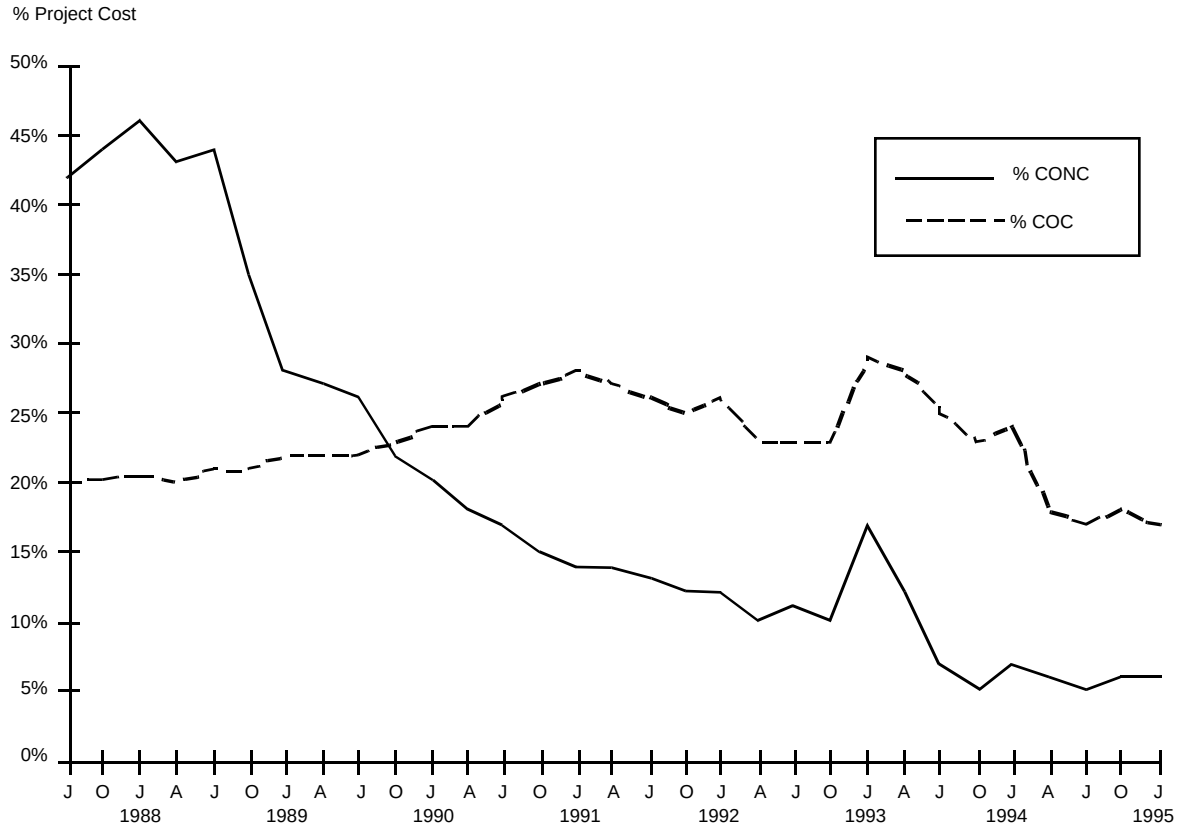


Figure 8: Cost of Quality Versus Time

In order to get a better understanding of possible causes of the rework reduction, we analyzed its subcomponents, as well as those of the appraisal and prevention costs. As expected, we found that rework savings were achieved at the expense of a small increase in appraisal or prevention costs. For example, appraisal costs rose when informal reviews were replaced by formal inspections and prevention costs rose when inspection training was instituted. Also, rework costs associated with fixing defects found during design rose from about 0.75% to about 2% of project cost and those associated with fixing defects found during coding rose from about 2.5% to about 4% of project cost.

The major reduction in rework costs was that associated with fixing source code problems found during integration, which dropped to about 20% of its original value. The second largest contributor to the rework reduction was the cost of retesting which decreased to about half its initial value. This clearly indicates that the additional costs of performing formal inspections and the training that must precede it are justified on the basis of finding problems earlier in the process, resulting in a more efficient integration.

This initial cost-of-quality analysis was indeed a learning experience. It was not easy in that many project people had to be diverted to this ”nonproject” task. It was also not inexpensive, costing about $25K of overhead. It was, however, promising enough to repeat the exercise about a year later and to eventually add the process to the normal senior management process reviews on a semi-annual basis.

By the second iteration of the analysis, we had developed an approach for quantifying the savings associated with the reduction in rework and the resulting return on investment. In doing so, we made some simplifying assumptions. First, we considered the savings and the investment over a one-year period (1990). Although the savings in any one year are due to process improvement investments in prior years as well as the current year, we ignored that factor and used only the 1990 savings and the 1990 investment.

As a baseline for the pre-improvement rework costs, we used the average value of the projects (41%) at the time the initiative started (August 1988). Then, we calculated the rework savings by project by month as the difference between the actual and the baseline (41%). Summing this over the sample projects for the one-year period yielded a total savings of $4.48M.

During 1990, the sample projects had employed 58% of the total available SEL labor force. Assuming that all projects benefited from the process improvement investments, we prorated the total investment ($1M) to the sample projects, yielding an investment of $0.58M. The return on investment (ROI) was thus 7.7 to 1 ($4.48M/$0.58M).

As the analysis was updated (annually in 1991 and 1992, and semi-annually thereafter) new projects were added to the database and new insights gained. Projected savings that had been predicted early in the development were, in fact, occurring. Two of the original six sample projects completed software development during 1991 with substantial reserves intact. Both projects were large, with software-only costs in the $17M range, and both completed slightly ahead of schedule. One was four % under budget and the second was six % under budget. When the latter project was delivered to the customer, Raytheon received a schedule-incentive award of $9.6M, which is not included in any of the above ROI calculations.

By 1994, 18 projects were in the database and the data-gathering exercise had become more routine. Although the full analysis was being made semi-annually, some department managers were requiring their project leaders to provide the cost-of-quality data along with their normal monthly tracking data. It was gradually becoming a part of some managers’ approach for monitoring project status.

One of the valuable lessons learned during this period was that our cost-of-quality approach would not be sufficient as the single means of measuring the results of process improvements. One drawback to our approach was that the results were normalized, showing rework costs, for example, as a percent of total project cost. Was it possible, software lab management questioned, that costs were simply being ”shuffled” from one category to another and the bottom line cost of doing a job was not going down at all?

7.2 Software Productivity

In addition to cost-of-quality data, we collected available data from individual projects on their productivity in terms of equivalent delivered source instructions (EDSI) per person-month of development effort. Although we realized that this was not a scientifically accurate measure, two factors were in our favor. First, this was one of the measures management routinely used in reviewing projects containing any combination of new, modified, and reused code; and second, most of the projects were similar in type (real-time embedded) and in size (70,000 to 500,000 DSI).

We combined the data from all projects using the same approach as the cost-of-quality data; namely, by calculating the monthly weighted average using staff size as the weighting function. The plot of the results showed that average productivity was, in fact, increasing as a function of time meaning that jobs were costing less. These data are also now gathered routinely and regularly.

DSI estimates are tracked throughout a project beginning at proposal time. Our initial estimates are formulated in conjunction with developing “thin specs” for each computer software configuration item. Our methodology accounts for both new and reused code. Both modified and added lines are counted as new. Reused DSIs are weighted according to the relative effort (as compared to new code) of modifying or reusing it (including its related documentation). Proposal DSI estimates are used to project the development effort based on historical productivity data. The counts are updated during each phase. One of the most important updates is done after the requirements specifications have been completed. This estimate is part of the process that establishes the baseline plan used to manage the software development project. This baseline includes the DSI counts, schedules, staffing, and costs. The actual DSI counts are determined during the coding phase and are maintained through the testing and delivery of the product.

After delivery, the product may be reused, either partially or completely, and incorporated in a product to be delivered to another customer. In addition, our customers often request changes to the original baseline that are delivered as a new version. This reuse baseline code is tracked as reused DSIs, and new code (enhancements or additions) is tracked as new DSIs. The cost estimating process includes a factored count of the reused code.

We continue to update weighted average productivity as projects are added to our database; we now have data on 24 projects, not all of them complete. The latest results are reflected in Figure 9, which shows an average productivity increase of about 170% over the period of the initiative. Figure 9 does *not* include multiple counts of software captured in multiple releases to other customers — all our programs are a single release of the system.

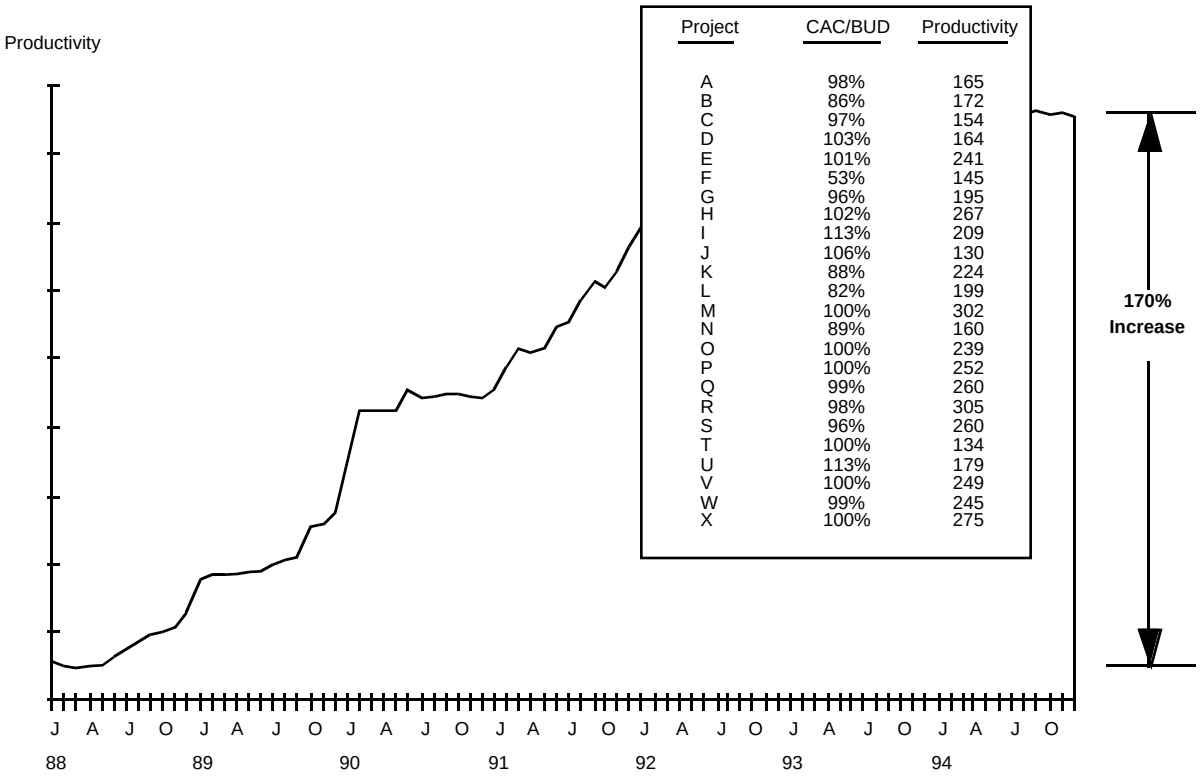


Figure 9: Software Productivity Increase Versus Time

Comparing these 24 projects over time is a valid way to evaluate productivity improvement because, although every project is unique, they are all of the same type (real time, embedded) and with a reasonable size range (70,000 to 500,000 DSIs). Thus, if neither the nature of the application nor the measurement method changes in this time, it is reasonable to credit the improvement to what *did* change, namely, the process. Our productivity calculations include engineering (software design, code, unit test, and software integration), document preparation and generation (SDP, SDD, IDD, and software development folders, and SPS), pathfinding (risk mitigation) and prototyping, software configuration management, software management (task and line management), administration, and resolution of software problem reports (document changes and updates to the software through integration). Our productivity calculations do not include software requirements analysis or formal qualification test. Productivity is calculated using the DSIs measured at the completion of a project and the development effort in

staff months. The measured DSIs is the sum of the new code and of a factored amount of reused code. The development effort included in the productivity is measured using our standard cost accounting system. The calculation for each project is the quotient of the two measurements.

The 2.7 times improvement factor was calculated using the average productivity during August 1988 and the average productivity of recently completed projects. To protect the proprietary nature of our actual productivity achievements, Figure 9 represents productivity relative to the initial 1988 value. Each point on the graph is given by *100 x (productivity - base productivity)/base productivity*. The productivity for each point is a weighted average based on the staffing level of each project and the measured productivity for each.

7.3 Cost Performance Index

Another concern we had was whether we were really performing on projects. This issue was addressed by collecting data on the project's budgeted (predicted) cost and its actual cost at completion (CAC). This cost performance index ratio (CAC/Budget) for each project was then used to compute the monthly weighted average (again using the same approach as the cost of quality) to yield a plot of this time-variant measure. The results were encouraging, showing that the cost performance index was improved dramatically from about the 40% overrun range prior to the start of the Initiative to the $\pm$3% range by late 1991 (when we achieved SEI Level 3) and continuing through the present time (see Figure 10). Recognizing that we would ideally like this to be a 0% fluctuation, we are convinced that software process maturity carries with it a fair amount of schedule and cost predictability, which is a fundamental goal of our process!

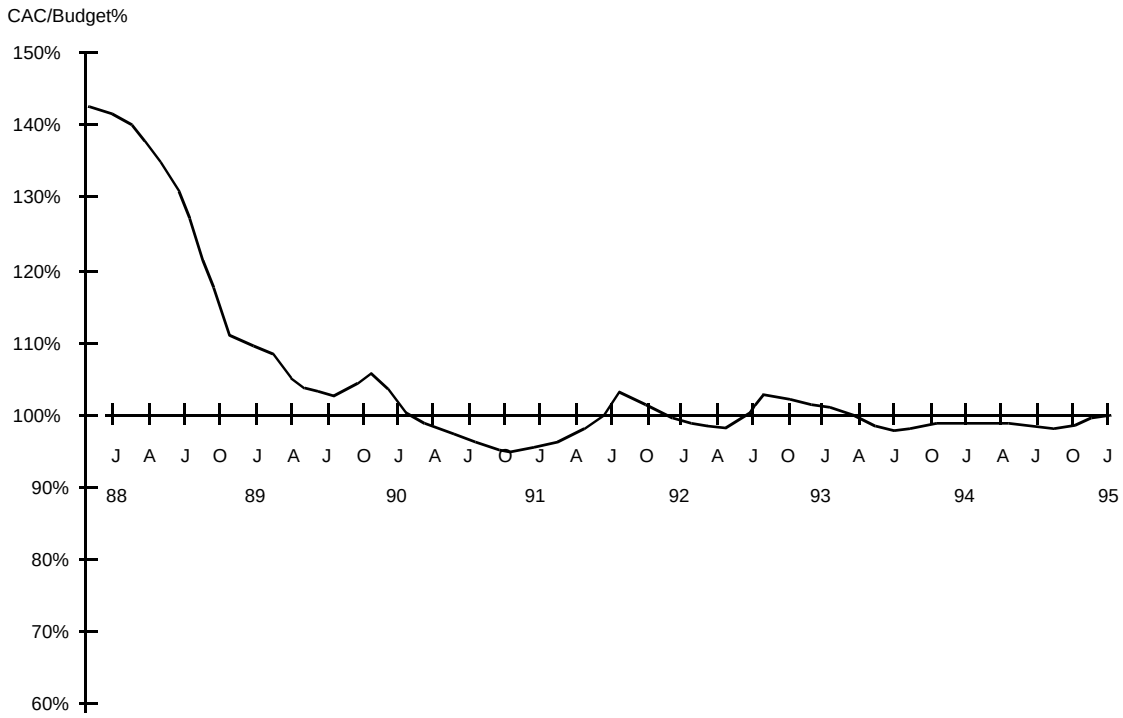


Figure 10: Achieving Project Predictability

7.4 Overall Product Quality

The primary measure used to assess overall product quality is the defect density in the final software products. We measure this factor in “number of software trouble reports (STRs) per thousand lines of delivered source code (STRs/KDSI)” on an individual project basis. The project defect densities are then combined to compute the monthly weighted average (using the same approach as the cost of quality described above) thus yielding a time-variant plot of our overall product quality measure. As shown in Figure 11, data collected over the period of the initiative shows an improvement from an average of 17.2 STRs/KDSI to the current level of 4.0 STRs/KDSI.

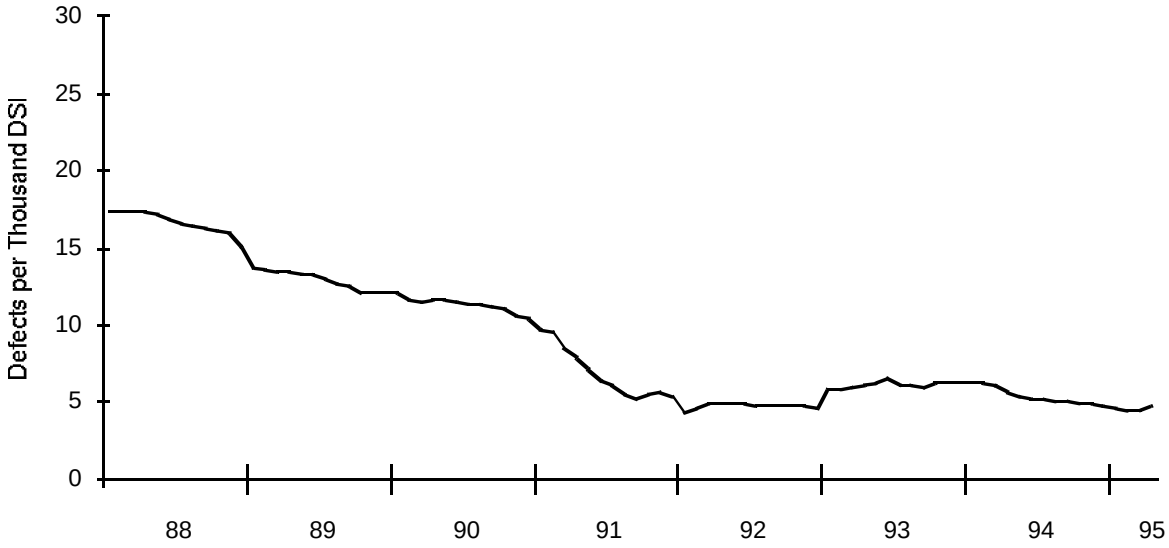


Figure 11: Defect Density Versus Time

Of course, the ultimate demonstration of overall product quality is the contributions software has made to RES's success with software-intensive systems. Our first major success was on the FAA's Terminal Doppler Weather Radar program. For this new system, we developed over 200,000 lines of software tightly coupled to a new-technology radar designed to detect incipient microburst conditions in the vicinity of air traffic control (ATC) terminals. Using our newly-minted Level 3 process, we managed to break with Raytheon tradition by moving software off the program's critical path and helped lead the way to initial delivery 6 months ahead of schedule, thereby earning Raytheon a maximum early-delivery incentive of nearly $10 million.

This was immediately followed by our first major Ada program, COBRA DANE System modernization. COBRA DANE is a technical intelligence gathering phased-array radar on the tip of the Aleutian chain for which Raytheon developed nearly 350,000 lines of new code and once again managed to remain well off the critical path. This $50 million system upgrade was turned over to the Air Force 4 months ahead of schedule. We've more recently built software-intensive, state-of-the-art air-traffic control systems that feature UNIX-based open systems architectures, significant COTS content, and a productization focus that is achieving significant software reuse. These are attracting world-wide attention and are instrumental in Raytheon moving into a dominant position on the international ATC front. A derivative of our ATC automation products is now emerging in the domain of vessel traffic management, with the commissioning of a Coast Guard system in Valdez, Alaska.

7.5 Other Organizations

Our software process is institutionalized at SEL and has been extended to our facilities in Portsmouth, Rhode Island and Cossor Electronics of the United Kingdom. The process at SEL now encompasses a 1,200-person Software Engineering Laboratory (SEL). More recently, E-Systems was acquired by Raytheon. We are working with them to acquire the best of each of our practices.

RES's SEPG has supported the greater Raytheon software community in a variety of ways. Training materials have been shared with multiple Raytheon entities (Computer Sciences Raytheon and Raytheon Canada) and with Raytheon customers, including the Air Force Electronic Systems Center's PRISM program, the FAA, and the aviation authorities of Norway and Germany. We use our process to evaluate and benchmark the processes of potential subcontractors. We flow down our process to our subcontractors, and in some instances, we have been placed on contractor teams *because* of our Initiative and ability to migrate our process to the prime and other team members.

Our SEPG experience has been shared with the software community at large and SEI affiliates in a number of forums, including advisory boards, workshops, briefings, and correspondence groups. We were instrumental in forming and continue to be active in the Boston Software Process Improvement Network (SPIN), a mechanism for stimulating the sharing of process information among companies in the greater Boston area. We continue to be one of the few companies who publish cost-benefit data, lessons learned, and specific, numerical results of our process improvement efforts for the greater benefit of the software community. A comprehensive list of the meetings and publications where this information was shared is contained in the list of additional readings (Appendix B).

7.6 Personnel

In addition to the project characteristics that we carefully track to evaluate process improvement success (cost of rework, productivity, cost performance index, and overall product quality), we see less tangible but equally important results occurring in areas that affect our personnel.

The metrics discussed in the previous sections quantify the outstanding performance by our software engineers which we hope gives them the job satisfaction and career enhancement that comes with successful performance on programs within RES. The real challenge is in management providing adequate support. Also, the initiative funds the quarterly SEL Newsletter containing numerous job-related articles written by SEL personnel, including an up-to-date list of ongoing projects and proposal efforts.

8 Summary

Raytheon began its software improvement activities in 1988, driven by compelling business reasons to improve upon the predictability of the cost and schedule of the software components of its major business areas. We chose the path of process improvement, guided by the CMM, because (1) it made sense and (2) our customers supported this approach. This choice has proven wise because we have made our software development activities predictable, more productive, and capable of producing higher quality products. Along with this, we have become more competitive and won additional business.

We organized our initiative into an executive committee responsible for steering and oversight, and into four SEPG Working Groups — each responsible for a major area in the initiative. The Policy and Procedures Group initially captured and documented our best practices so that they could be applied across all projects. The Training Group elevated the importance of training from *ad hoc* "on the job" learning to a full commitment of the software organization to ensure that each project had its engineers fully trained before beginning work. The Tools and Methods Group developed the technologies (CASE tools, workstations) and the methods (Ada, object-oriented). The Process Database Group developed the process and quality metrics and statistical process control to assess the performance of both projects and the process. These working groups tailored the process to be effective within the Raytheon culture.

There were five fundamental reasons for our successful Software Engineering Initiative:

1. The vision and commitment for the initiative came from the manager of the software organization. The vision and commitment included more than just funding — The manager of the software organization was the focal point and actively drove the effort.
2. We had support from general management — They became active sponsors. Commitments of funding and general management's requirement that all business areas adhere to the process were part of this sponsorship.
3. Our process improvements clearly and continually demonstrated business benefits to projects.
4. We carefully considered the corporate culture of Raytheon — We understood how our company managed engineering work, allocated discretionary resources, and made commitments.
5. Most importantly, we ran the initiative from within the ranks of the software organization — Task managers and line engineers did the vast majority of the work and hence felt ownership of the process and the products. It was something that they helped create, rather than something imposed upon them from outside their control. Thus the projects and the process worked together to achieve the increases in predictability, productivity, and quality.

Today we find that our business demands — both defense-oriented and commercial — are changing along with the accelerating computer hardware and software technologies. Demand for major software products developed “from scratch” is shrinking, and is being replaced by the demand for complex new software products, initially prototyped, utilizing industry standard open interfaces and protocol COTS/NDS products that are produced at lower cost and with shorter schedules.

Because of these circumstances, our software process is changing so we continue to deliver effective solutions within the context of the Software Engineering Initiative. We are taking the technology and processes developed by our RAPID (Raytheon Advanced Prototyping, Integration, and Demonstration) laboratory and institutionalizing them within the framework of the Initiative’s organization. The processes for effective COTS evaluation, prototyping, software systems and COTS integration, use of object-oriented techniques, and domain-specific reuse are becoming as standard as code inspections within Raytheon.

Our Software Engineering Initiative is an exciting and worthwhile endeavor. It continues to grow and change as both Raytheon and the software industry change. We view the IEEE Process Achievement Award as one endorsement of the vision and results of our initiative.

References

[Camp 89] Camp, Robert C. *Benchmarking*. Milwaukee, Wisconsin: ASQC Quality Press, 1989.

[Crosby 84] Crosby, P. *Quality Without Tears*. New York: McGraw-Hill, 1984.

[Humphrey 87] Humphrey, W. and Sweet, W. *A Method for Assessing Software Engineering Capabilities of Contractors* (CMU/SEI-87-TR-23, ADA 187230). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, September 1987.

[Humphrey 89] Humphrey, W. *Managing the Software Process*. MA: Addison Wesley, 1989.

[Paulk 93] Paulk, M; Curtis, B.; Chrissis, M; Weber, C.*Capability Maturity Model for Software, Version 1.1* (CMU/SEI-93-TR-24, ADA 263403). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, February 1993.

Appendix A Applying SPC to the Software Development Process

Since 1988, Raytheon has utilized detailed design and coding peer inspections as part of our Software Development Process. According to industry-wide gathered data, only 50 to 60 percent of all defects are detected during these phase inspections. Subsequently detected defects typically end up costing between 10 and 20 times as much as those detected during their own phase. Therefore, increasing our ability to detect defects during these inspections would not only improve the performance of our products but would reduce our overall costs as well. For this purpose, statistical analysis of detailed design and code inspection data was undertaken. Data collected across several programs produced an exponentially distributed histogram with a mean of 32 action items (AIs) per thousand lines of code (KDSI) and a standard deviation of 32 (Figure A-1). Our objective was to identify the inspection process conditions for newly developed code that will increase the detected number of action items per KDSI while reducing the variation between inspections.

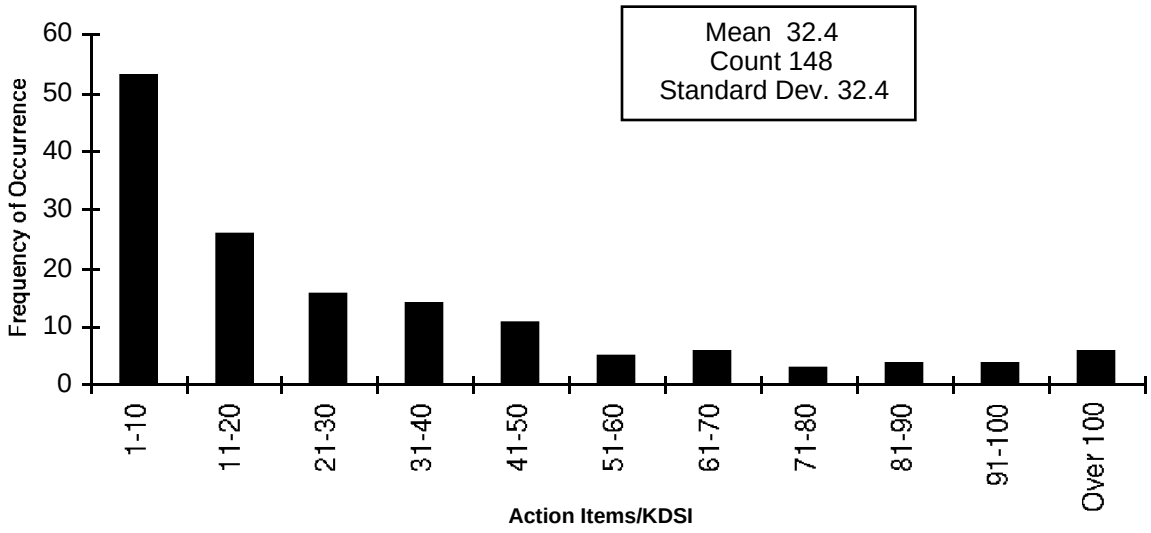
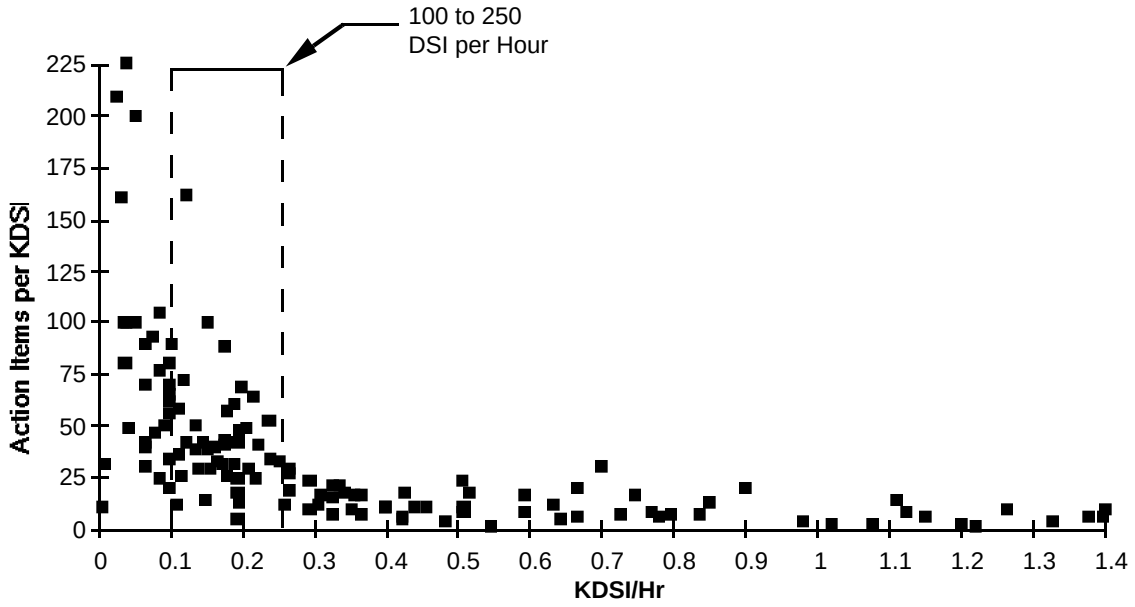


Figure A-1: Distribution of Action Item Frequency for Inspections

Various inspection process parameters which were thought to potentially impact our inspection process were modelled using regression techniques. Our analysis indicates that three key process conditions appear to be correlated with our new code defect detection rate (AIs/KDSI):

1. review speed (KDSI per hour)
2. review package size (KDSI)
3. preparation time (total preparation hours)

Figures A-2 and A-3 present our scattergram plots individually correlating inspection review speed and review package size to our observed new code defect detection rate. This is not to say that other factors do not influence our defect detection rate, only to indicate that these three key factors play a significant role in the efficiency of our inspection process.



Note: 20 Action Items per KDSI equates to 1 Action Item per page of review material (code or design)

Figure A-2: Action Item Density Versus Inspection Review Speed

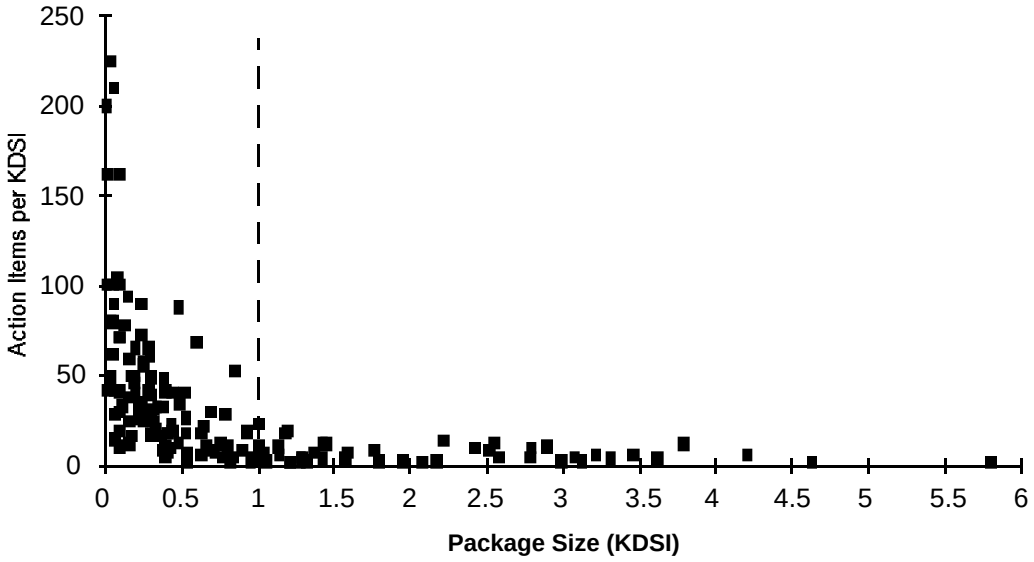


Figure A-3: Action Item Density Versus Inspection Package Size

When inspections are run at our statistically preferred levels of 100 to 250 DSI per hour, with a total preparation time of greater than 5 hours per KDSI and package sizes of under 1000 DSI, the average number of action items detected per KDSI increased by 34 percent (from 32 to 43 AIs per KDSI), while its variability decreased by 49 percent (from a standard deviation of 32.4 to 16.6). Figure A-4 presents a defect density histogram of our improved inspection process. Based on these data, initial SPC control limits of 20 to 60 AIs per KDSI for our inspection process under preferred conditions were established. An approximately 80 percent confidence interval was chosen for our control limits instead of the more traditional three sigma limits for two primary reasons: our low false alarm rate cost of investigation and our expectation that inspection teams would have an initial tendency of reverting back to their old process.

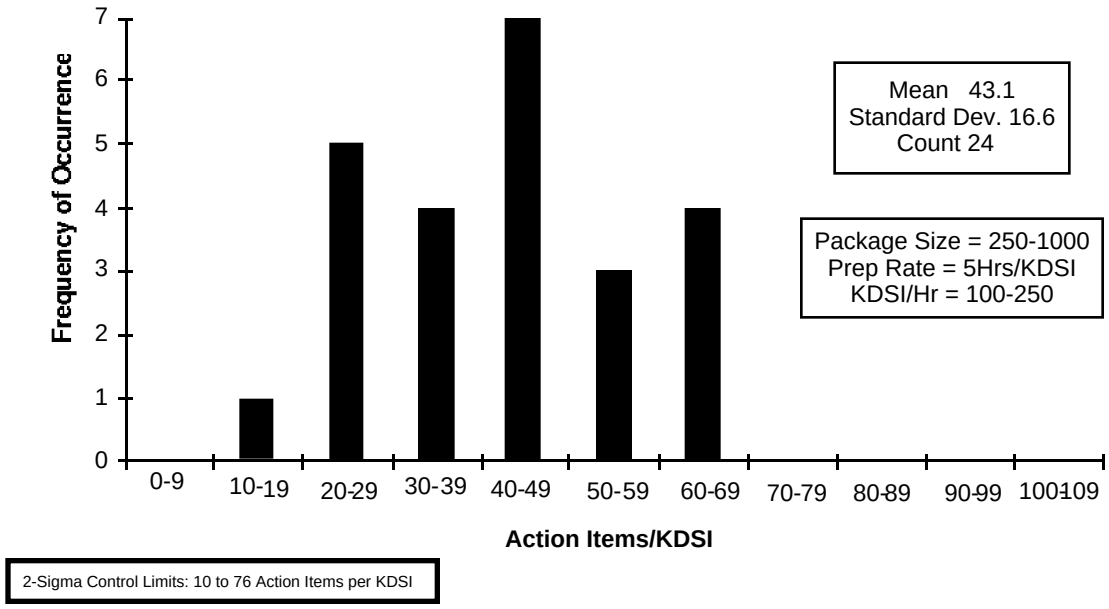


Figure A-4: Basis for Control Limits

An Air Traffic Control (ATC) software development effort was chosen as the pilot program for SPC implementation. A three-hour course, Statistical Methods for Software Improvement, was held as a method of introducing the project leads to statistical thinking in general and, in particular, to our inspection analysis results and their practical application.

As expected, analysis results on the ATC program to date have tracked closely with the results previously obtained. When our preferred conditions are applied to new or mostly new code (limited reuse), our ability to detect action items per KDSI increases substantially. Inspections that were run under preferred conditions but did not product an AI per KDSI ratio within our control limits were further investigated. It should be noted that inspections that fall outside of our expected range are not necessarily poor inspections or code, but merely warrant further investigation due to their unexpected performance. One example of this involved an inspection where despite running at preferred conditions, a lower defect density than expected was noted during the review. Upon investigation,we found that the software engineer developing the code is as close to a C guru as there is on the ATC program. For this reason, our Software Laboratory plans to gain leverage from this employee's skill set and development process whenever possible. Investigation into why other inspections were not previously run at preferred conditions also yielded interesting information. One CSCI lead noted that when he is the author, he has a difficult time getting the critical review he would like and, therefore, his reviews tend to move at a faster rate than preferred. This result has reinforced our commitment to the consistent use of peer review during the inspection process.

To date, our SPC implementation efforts on the ATC program have validated our previous statistical analysis efforts. Reviewing new code (or mostly new code) and operating under our preferred conditions typically results in 20 to 60 action items per KDSI. Data points outside these thresholds merely warrant further investigation. Further investigation may result in continuous improvement opportunities (process learning), corrective measures, or no action taken. It appears from our experience that the best way to operate inspection reviews under preferred conditions is to whenever possible

- Break review packages down into 1000 DSI or less.
- Allocate and schedule the review meeting length to be consistent with approximately 100 to 250 DSI per hour.
- Allow for adequate preparation time (a minimum of 5 total hours per KDSI), and ensure that sufficient peer reviewers are present.

As presented, the practical use of SPC methods is an effective way of improving and controlling your software development inspection process.

Appendix B: Additional Information

Additional related information is presented below.

1. “Creating and Implementing Working Groups,” SEI Third Annual SEPG Workshop, November 7, 1990
2. “Quantifying the Benefit of Software Process Improvement,” Software Process Improvement Workshop, November 8, 1990
3. “Actually Measuring the Cost of Software Process Improvement,” NSIA Seventh Annual National Joint Conference on Software Quality and Productivity, April 23, 1991
4. “Industry Experiences with SCEs” panelist, SEI Fourth Annual SEPG Workshop, April 9, 1992
5. “Elements of a Process Improvement Program,” IEEE Software, July 1992
6. “Measuring the ROI of Software Process Improvement,” DACS Fourth Annual Software Quality Workshop, August 3, 1992
7. “Cost of Quality as a Measure of Process Improvement,” SEI Software Engineering Symposium, September 17, 1992
8. “Raytheon’s Software Process Improvement Program: History, Impact, and Lessons Learned,” presented to the Air Force Comm - Computer Technology Integration Center, November 6, 1992
9. “Process Improvement and the Corporate Balance Sheet,” IEEE Software, July 1993
10. “Project Performance Improvement Through Software Process Improvement,” Tri-Ada '93 Conference, September 22, 1993
11. “Applying the SEI Capability Maturity Model to Real Programs,” TQM '93 Conference, November 4, 1993
12. “Raytheon’s Experience in Process Improvement: A Two-fold Increase in Productivity,” Sixth International Conference on Software Engineering and Its Applications, November 17, 1993