

SPI Guidelines for Improving Software

SEMATECH and the **SEMATECH logo** are registered service marks of SEMATECH, Inc.

SPI Guidelines for Improving Software

Technology Transfer # 94092541A-ENG

SEMATECH

October 31, 1994

Abstract: These guidelines for improving software by increasing the maturity of the processes by which it is developed and maintained include the Software Process Improvement (SPI) model of maturity and the method for using the model. The model and method are specifically tailored to the needs of supplier companies in the semiconductor industry. The guidelines apply to the work of software engineers, their managers, and the quality engineers in supplier companies who champion change and improvement. This material is intended to provide a context for developments that support and expedite software improvement. In addition, this document discusses the challenge software is posing for the semiconductor industry and the global effort to provide resources and enact standards for software improvement. This document supercedes Technology Transfers #93091806A-ENG, 93091807A-ENG, 93091808A-ENG, and 93102048A-ENG.

Keywords: Project Management, Quality Management, Software Reliability, Software Development, Software Development Life Cycle, Suppliers

Approvals: Ted Ziehe, Author
Harvey Wohlwend, Project Manager
Gary Gettel, Director
Laurie Modrey, Technical Information Transfer Team Leader

Table of Contents

1	EXECUTIVE SUMMARY.....	1
1.1	Resources for Improvement.....	1
1.2	Product Improvement.....	2
2	SOFTWARE AND SEMICONDUCTOR MANUFACTURING	2
2.1	Fab Operations Involve More and More Software	4
2.2	Manufacturers Buy a Lot of Software.....	5
2.3	Software is a Challenge for Suppliers.....	7
2.4	SEMATECH Helps Suppliers Improve Software.....	9
3	SOFTWARE IMPROVEMENT FOR SUPPLIERS: SOME BASICS.....	10
3.1	Software: What Is It?	11
3.2	Software Has a Place in Your Products and Business	11
3.3	Know How You Do Software.....	12
3.4	Doing Software Involves More than Coding	13
3.5	A Software Effort Produces More than Code.....	13
3.6	Shortcuts Often Make the Path Longer.....	14
3.7	A Checklist for Your Software Operation	15
3.8	Solving Some Common Problems	17
3.8.1	Configuration Mangement	18
3.8.2	Peer Reviews (Inspections).....	19
3.8.3	Project Planning and Tracking	19
3.8.4	Software Testing	20
3.8.5	Defect Tracking.....	21
3.8.6	Software Requirements	22
3.9	Improvements Beyond the Basics is Good Business.....	23
4	A VISION FOR SOFTWARE	23
4.1	Flawless Software in Fab Operations	25
4.2	Business Success Through Software	26
4.3	Continuous Improvement of a Software Process.....	27
4.4	Improving Products Through Process Improvement	28
4.5	Implementing the Vision.....	29
5	THE SOFTWARE PROCESS, MATURITY, AND PRODUCT QUALITY	30
5.1	A Model of Maturity Exists for Software.....	31
5.2	The Model's Key Process Areas Indicate Maturity.....	32
5.2.1	The Structure of the Indicators.....	34
5.2.2	The Content of the Indicators.....	38
5.2.3	Some Starting Points for Process Improvement.....	42
5.3	Metrics Tie Process Change to Product Quality	42
5.4	Organizational Factors Impact Improvement Initiatives.....	43
5.5	An Organization Must be Ready for Software Improvement.....	43
6	INCREASING PRODUCT QUALITY AND PROCESS MATURITY.....	44
6.1	Success Is in Your Hands	45
6.2	A Method for Using the Maturity Model Exists.....	46

6.3	A Cycle of Five Phases Provides the Framework.....	47
6.3.1	Accomplishments and Milestones Around the Cycle	48
6.3.2	A Supporting Infrastructure	50
6.3.3	Completing a Plan for Improvement.....	53
7	REACHING THE GOAL SET FOR SOFTWARE IMPROVEMENT.....	55
8	REFERENCES.....	57

List of Figures

Figure 1	World Semiconductor Equipment Sales—Worldwide Market Share.....	3
Figure 2	World Semiconductor Market Share.....	4
Figure 3	Growth Path for Software Embedded in Equipment.....	5
Figure 4	Growth Path for Software in a Semiconductor Fab	6
Figure 5	The Three Leverage Points for Improving Software Products.....	24
Figure 6	The Requirements for Organizational Change.....	30
Figure 7	The Maturity Levels of the Capability Maturity Model.....	31
Figure 8	The Key Process Areas by Maturity Level.....	32
Figure 9	The Capability Maturity Model and Its Accessories.....	33
Figure 10	The Steps to Institutionalizing the Practices of a Key Process Area.....	35
Figure 11	The Key Process Areas Assigned to Process Categories.....	37
Figure 12	Another Assignment of Key Process Areas to Categories	38
Figure 13	The IDEAL Approach to Improving a Software Process.....	48
Figure 14	The Major Accomplishments and Milestones in Improving a Software Process	49
Figure 15	Organizational Functions That Support Continuous Improvement	51
Figure 16	Technology Change Plan Template	56

List of Tables

Table 1	Descriptions of Maturity Levels in the Maturity Model	34
Table 2	Initial Issues for Software Improvement.....	42
Table 3	Organizational Factors for Software Improvement.....	43
Table 4	Readiness Criteria for Software Improvement.....	44

Acknowledgements

About This Document

This handbook of guidelines for improving software is a product of the Software Process Improvement (SPI) Project at SEMATECH. The SPI Project is committed to increasing the operational reliability of the software that SEMATECH's member companies acquire from their suppliers.

This document is part of a set of materials the project is preparing and collecting for SEMI/SEMATECH members to use. These 170+ companies are U.S. suppliers of semiconductor manufacturers. The planned set of materials includes the following:

The SPI Project: A SEMATECH Initiative to Improve Software (a general overview of the project)

SPI Guidelines for Improving Software, Release 3.0 (this document)

Process Assets that support the SPI Guidelines, with Scenarios for Using Them (templates, drafts, check lists, and other aids for improving software and the software process, many in electronic form)

Collections of Resource Materials for Software Improvement (a library of general reference and resource materials)

- Maturity Models
- Curriculum of training courses
- Case studies
- Catalog of products and services
- Bibliography of other materials

These materials are specifically for those members of SEMI/SEMATECH who deliver software (embedded or otherwise) to their customers.

This presentation of the guidelines reflects lessons learned through working partnerships with six suppliers companies during the past two years. Therefore, the material is also from

- Applied Materials
- Eaton Corporation, Semiconductor Equipment Division
- Plasma and Materials Technology, Inc.
- PRI Automation Inc.
- Teradyne, Inc., Semiconductor Test Division
- Varian, Thin Film Systems

The SPI Project functions as custodian of the guidelines, keeping them aligned with the experiences of supplier companies and with the various resources and standards on which the guidelines are based.

We have addressed this materials very explicitly to the executive manager in a supplier company who sponsors the software improvement initiative, and to the other champions and change agents who provide the needed leadership and expertise. Sometimes in the document we refer to you collectively as the SEP (Software Engineering Process) Leaders or simply “you.”

Other people who provide leadership for software improvement also will find this material useful:

- Various software managers and engineers in supplier companies
- Manufacturers' staffs who deal with suppliers
- Commercial providers of software improvement services

Information about this document or any of the other materials listed above is available from the SPI Project at SEMATECH:

Ted Ziehe
Internet e-mail: Ted_Ziehe@sematech.org
Phone: (512) 356-3816
Fax: (512) 356-3575

Copies of this document (and others as they are completed) can be ordered from SEMATECH:

Customer Service
SEMATECH Planning & Technology Transfer
2706 Montopolis Drive
Austin, TX 78741

Internet e-mail: info@sematech.org
Phone: (512) 356-SEMA
Fax: (512) 356-3081

Several reports from the Software Engineering Institute (SEI) are cited in this document. Copies of those reports are available from the following distributor:

Research Access, Inc.
3400 Forbes Avenue
Suite 302
Pittsburgh, PA 15213

Phone: (800) 685-6510
Fax: (412) 682-6530

Improving software by maturing a company's software capability—the focus of these guidelines—is also motivating the other elements of the SPI Project's strategy. These other elements are to

- Rigorously define the industry's software issues and problems
- Actively support ongoing software improvement initiatives in supplier companies
- Execute a strategy and program for proliferating software improvement among supplier companies
- Ensure that manufacturers specify software quality in their procurements

x

- Make a business case for software improvement to supplier company managers
- Develop the requisite skills of champions and change agents for software improvement in supplier companies
- Attract experienced software people into roles within the industry

1 EXECUTIVE SUMMARY

This handbook is the third annual release of guidelines for improving the software an organization produces. The guidelines are based on an approach that improves software by increasing an organization's overall software capability, in particular the processes used to develop and maintain software.

In addition to process improvement, the guidelines deal with training that improves people skills and with software engineering that advances the use of technology. In all, the guidelines apply to the work of software engineers, their managers, and the quality engineers who champion change and improvement.

This version of the guidelines is from SEMATECH's Software Process Improvement Project and six supplier companies that are working to improve their software.

1.1 Resources for Improvement

Two resources are the foundation of ongoing improvement initiatives and are integral parts of these guidelines:

A model of maturity for software. The model we use is the Software Engineering Institute's *Capability Maturity Model for Software*, Version 1.1.

A method for using this maturity model. We present a method framework, enhanced by our experience working with supplier companies. It is the IDEAL Framework for Software Process Improvement, also from the Software Engineering Institute.

Used together, the Model and Method ensure that the improvement goals an organization sets are suited to its current capability and needs and that an organization achieves the goals that it sets.

The Model represents "completeness" for a software process. It does this by naming 18 important areas of activity in software development and support. These 18 areas, known as Key Process Areas (KPAs), are organized in a manner that defines a roadmap for evolutionary improvement. Each stage of improvement provides the foundation on which to build improvements in the next stage. The completeness that the KPAs represent is an aspect of maturity; additional criteria in the KPAs relate to training and other prerequisites of maturity.

The Model further characterizes maturity by listing for each KPA a set of detailed goals together with representative practices for reaching the goals. Success metrics defined for each KPA also connect software process maturity to product quality.

With the Maturity Model in hand, the Improvement Method is a guide to using it. The Method is a resource for establishing a "system" for improving software by continuously maturing the process used to produce it. First, an organization develops an approach to continuously improving its software. This may be done by simply making some improvements and then "systematizing" the way they are done. The next step is to institutionalize the "improvement system" so that it becomes a permanent part of a company's operation. This second step also makes the improvement system an integral and permanent part of the software process.

1.2 Product Improvement

The key issue is to ensure that continuous improvement of the software process also improves the software products. It works as follows:

- An organization uses its software process to produce software. That software process may be
 - - Roughly outlined or not defined at all
 - Immature relative to a maturity model, or very mature
 - Executed by managers and a software staff that are well trained and experienced or by staff that is deficient in the requisite experience and skills
- In any event, the organization does have a software process.
- The Maturity Model gives the characteristics of a mature software process, plus characteristics of a capable staff and of effectiveness in the use of technology. (The Model's key process areas express these characteristics.)
- Using the Maturity Model, the company comes to understand its software process in terms of the Model's characterization of maturity. It does this on a recurring basis. Guided by these understandings, the organization makes process changes that both improve the quality of its software products and increase the maturity of its software process.

This advance—from making improvements in either an ad hoc manner, or not at all, to a continuously-working, model-directed improvement system—puts an organization on the road to software quality and process maturity. The purpose of this handbook is to provide guidance in establishing and operating such an improvement system.

2 SOFTWARE AND SEMICONDUCTOR MANUFACTURING

The worldwide semiconductor industry is every changing, and competitive challenge is the theme of its culture. This has been true since its beginning, spurred on by developments in the technologies used to manufacture silicon chips. today manufacturers continue to drive change as they strive for products with smaller circuit geometries, greater component complexities, shorter production runs, and higher yields of useable product.

During the past decade, the U.S. semiconductor industry met the challenge of global competition through a revitalization of the supplier infrastructure (see Figure 1) and a reversal of trends in worldwide semiconductor sales (see Figure 2). These competitive pressures are continuing, and more and more they involve issues and functionalities related to software.

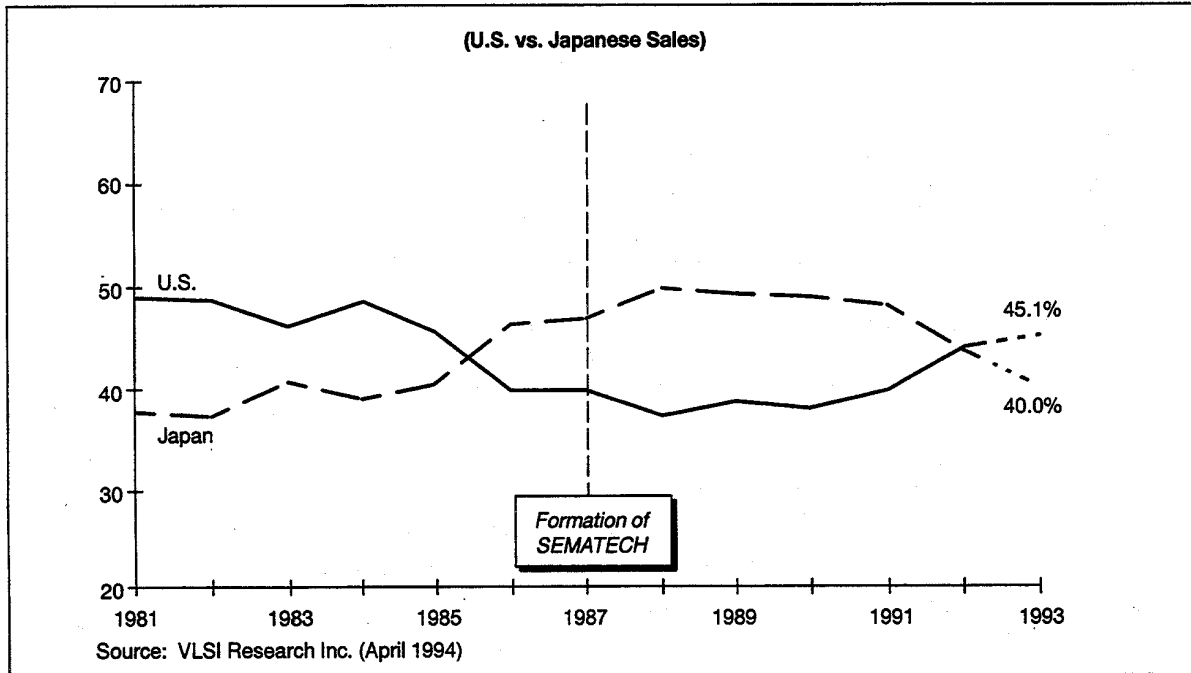


Figure 1 World Semiconductor Equipment Sales—Worldwide Market Share

It is within this setting of rapid change and competitive challenge that we offer this document.

SPI Guidelines for Improving Software

Tailored to the needs of suppliers, it is a resource for increasing software quality and an organization's overall software capability.

These guidelines from the SPI Project at SEMATECH (see Section 2.4) address the champions and agents of change in supplier companies and elsewhere throughout the industry. Your roles as “change agents” and “improvements champions” are central to software improvement within each of your companies.

This first section of the document sets the stage for the material that follows. Its overview of software in the industry—especially software from suppliers—makes clear the need for an industry-wide initiative to improve software quality.

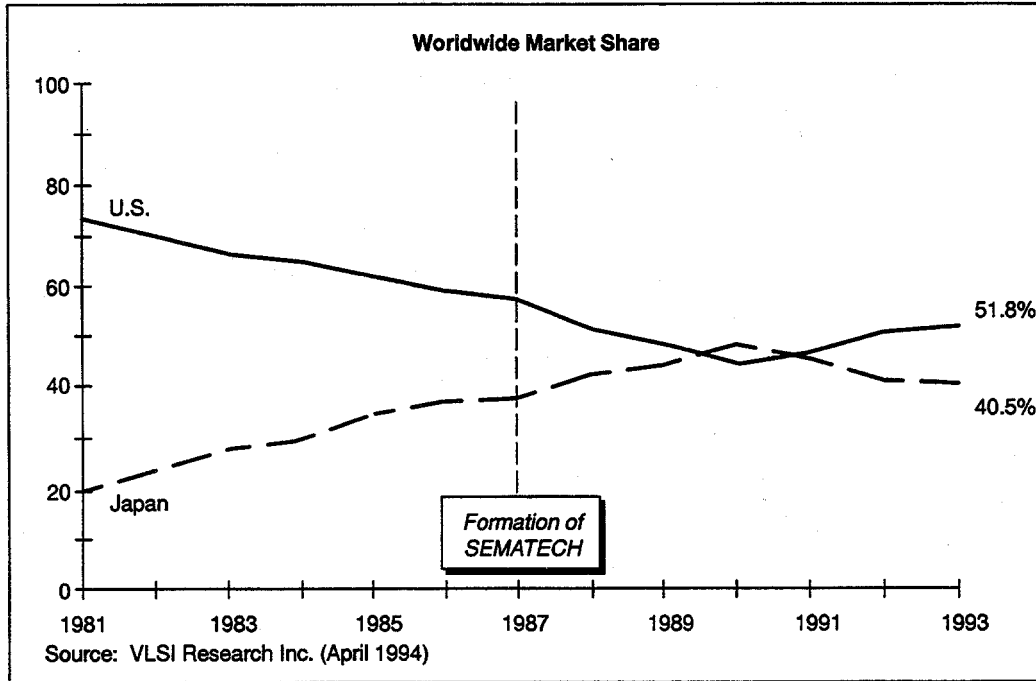


Figure 2 World Semiconductor Market Share

2.1 Fab Operations Involve More and More Software

Software is now a significant aspect of semiconductor manufacturing. And its significance, along with its volume, is increasing to a point that challenges the industry. The first National Technology Roadmap for Semiconductors [1] lists software as one of five “show stoppers” to realizing the industry’s potential during the next decade.

In some other industries the challenge came sooner; these industries have successfully incorporated software into their products and operations [2]. But the semiconductor industry is in the early stages of dealing with software. For example, two surveys of industry suppliers show little if any activity to improve software capabilities during a recent three-year period [3]. Similarly, manufacturers are just beginning to recognize that the procurement of highly reliable software requires proactive purchasing transactions that define software quality and make it a requirement [4].

Automation and automated information systems in semiconductor fabs are on the increase. A 1993 study of eight fabs in the U.S. and Japan [5] found that in computer-integrated manufacturing (CIM) and information systems, the leading fabs collect and analyze large amounts of data, enabling them to troubleshoot their overall manufacturing operation quickly and comprehensively. The leading fabs also perform extensive automated yield correlation analyses, and they perform on-line electrical and particle measurements. As problems are detected, information systems are used to rapidly alert appropriate staff and to assist them in troubleshooting efforts.

Because of these and other trends, the volume and complexity of software in a silicon factory is increasing exponentially, and a leveling-off is not in the forecast. Figure 3 shows the growth of

the software component in some representative lines of fab equipment. Figure 4 estimates the overall rate of increase for software involved in semiconductor manufacturing [6]. Superimposed on the projection in Figure 4 is a plot showing the increase for software in manned spacecraft vehicles during the past 20 years, indicating that a representative fab involves an equivalent amount of software. Also shown is a projection of the generally accepted annual productivity increase for software engineers. "That means that the number of people required to provide and support the needed software is increasing sharply and will continue to increase during the next decade.

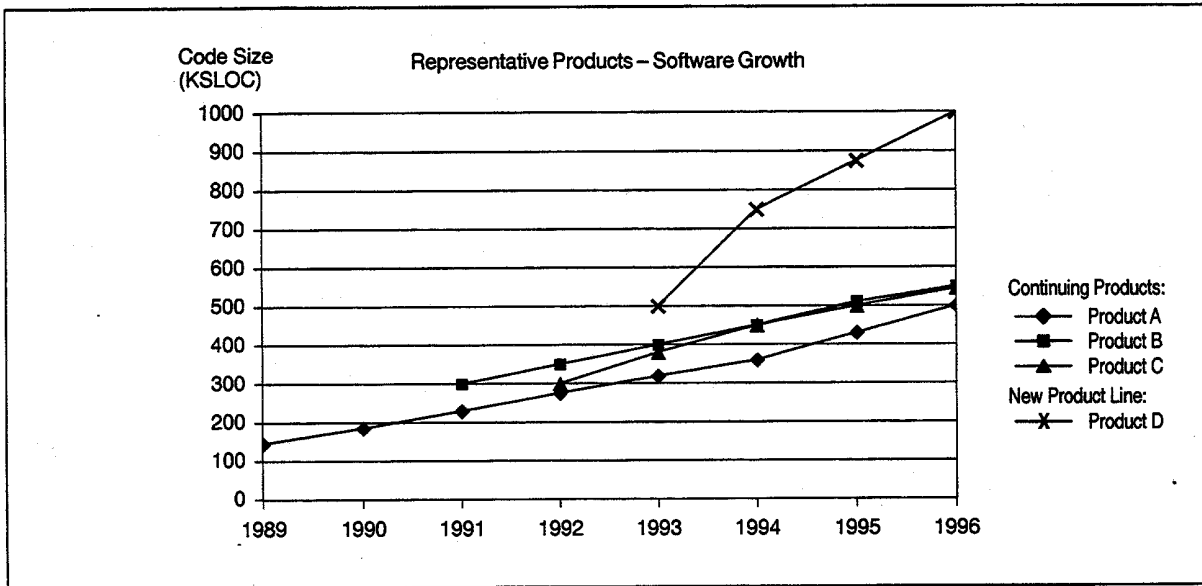


Figure 3 Growth Path for Software Embedded in Equipment

There are, however, some problems. In a recent survey [7] fab managers in SEMATECH's member companies listed unscheduled equipment down time as having the greatest impact on capacity loss. Other data indicate that software is a prime contributor to equipment down time. One researcher [8] gives this account from interviews with staff at one representative fab:

There is tremendous frustration with software, its lack of capabilities, its faults, the difficulty of diagnosis and revision, etc. Faulty software consumes a large portion of engineering resources and process time... I'm appalled at what is tolerated in terms of software quality, and at the relations between fabs and software houses. Exception handling is extraordinarily poor, fault tolerance nonexistent, documentation is bad, upgrades are exceptionally unreliable, people don't know how to talk about or measure software, and failure is so rampant that its presence is often ignored.

Clearly, there is much work to do on software improvement.

2.2 Manufacturers Buy a Lot of Software

Manufacturers buy much of the software used in semiconductor manufacturing. Some is purchased as standalone systems for shop floor control, but much of it is embedded in

equipment—equipment that processes silicon chips of that automates support activities such as material handling or product testing.

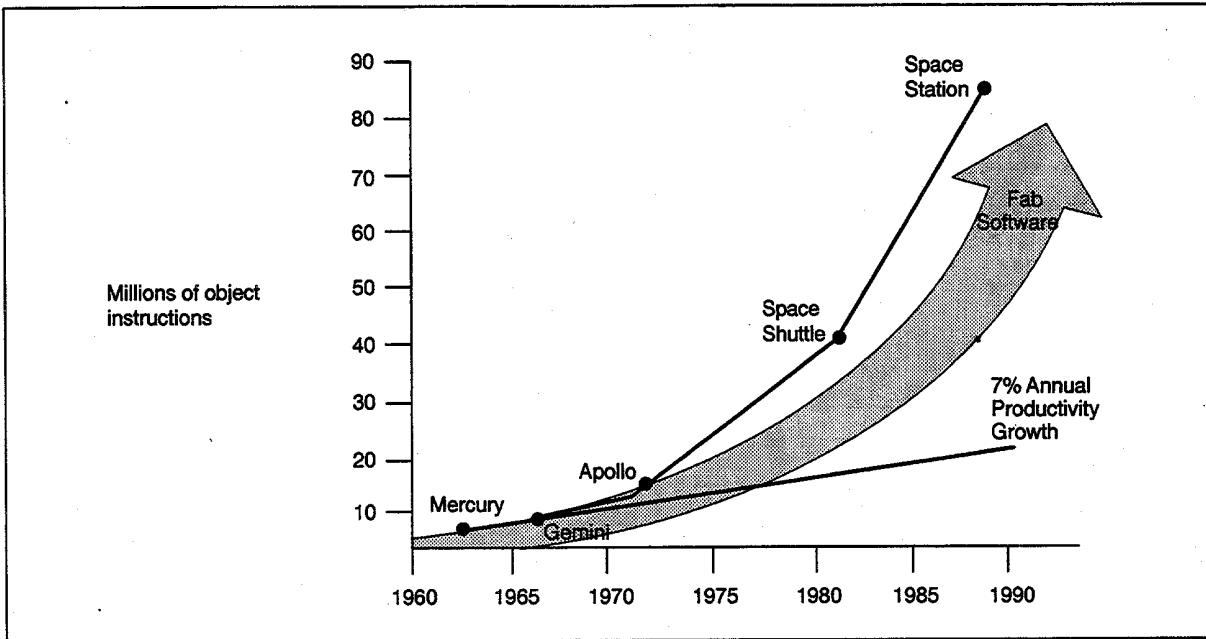


Figure 4 Growth Path for Software in a Semiconductor Fab

Embedded software from suppliers includes such functions as

- Equipment control
- Recipe management and processing
- Sensor and actuator management
- An interface with the equipment user
- Error detection, reporting, and recovery
- Interconnects with other pieces of equipment
- Interconnects with the control programs for a cluster, cell, and/or on the shop floor

In some cases, these software functions must conform to official or *de facto* standards; in others, a supplier is string for an advantage, or at least differentiation, over competitors.

As embedded software and standalone systems increase in scope and complexity, manufacturers are recognizing that requirement specifications in many acquisitions involve large and complex bodies of software. Such software procurements are increasingly important; arriving at specifications, costs, and schedules for them requires additional attention, knowledge, and expertise.

SEMATECH's member companies are individually and collectively requiring software quality along with product functionality. As software issues in semiconductor manufacturing become clearer, these requirements will become more focused and explicitly. Already, in some cases, they extend beyond the quality of delivered products to the maturity of the supplier's product development and support processes.

some manufacturers have established formal programs that set minimal standards for software quality from suppliers, including requirements that relate to testing, support, and quality assurance [9]. In these formal programs, manufacturers define minimum acceptance criteria that suppliers must satisfy to qualify for consideration. We expect this practice to become the norm in the years ahead.

2.3 Software is a Challenge for Suppliers

In the “good old days,” equipment suppliers developed and marketed their products based on expertise in mechanical engineering and the physical sciences involved in etching, diffusion, and lithography. Later, electrical engineering entered the picture, and today many of these suppliers are wrestling with a part of their businesses that produce and support software.

Similarly, suppliers of information systems that inform, organize, or control factory operations are also seeing change. Their systems must accept data from and provide data to software-based functions in equipment throughout the factory.

These changes are plain to see in the complexity of equipment used in silicon wafer processing. A decade ago, the level of machine intelligence needed could be hardwired control logic. Now the intelligence can be implemented only as software typically, as large and complex systems.

In addition to the equipment control issue, several industry trends are increasing demand for software from suppliers [10]:

- To increase throughput, different pieces of equipment are linked together, either by tracks or in clusters. These configurations require that the actions of the various pieces of equipment be coordinated, increasing the amount and complexity of the software required.
- In the past, factories were fairly static configurations, well adapted to large production runs. Today’s market is requiring flexible lines of processing equipment that can be reconfigured quickly for the small volume runs typical of application-specific integrated circuits.
- The industry is moving from run-to-run control to *in situ* metrology. Whereas the quality of a process step used to be measured after the step was complete, now measurements are being made on the fly by sensors integrated within the equipment, allowing the process to be tuned in real time.
- Increasingly, factories are installing equipment for automated material handling. These devices reduce or eliminate operator interactions in some situations, replacing them with automated interactions between pieces of equipment.
- Factory automation notwithstanding, people are still important in factory operations. More complex equipment adds complexity to all of its interactions with an operator.
- Manufacturers also want equipment in factories to help train operators, reducing the amount and cost of downtime and the cost of simulating factory operations in offline training.

In each case cited, software is the key to resolving the issue. Even as more suppliers decide to buy off-the-shelf software and apply it to their products, it is critical that each company have a highly developed software capability. AS the capital cost of equipment escalates and the cost of operations follows a similar trend, equipment utilization becomes critical. High reliability and high usage rates decrease a tool’s overall cost of ownership.

Why then is software capability such a challenge for the industry? And where is the industry in meeting the challenge?

First, software development is inherently difficult; many other industries suffer from the same problems [11]. Second, software difficulties are confronting manufacturers, suppliers of software systems for manufacturing, system integrators, and equipment suppliers alike. Some aspects of this challenge are reflected in the following:

- Software practices are evolving at a steady pace, yet are far from mature. CASE tools, reuse libraries, and object-oriented techniques are examples.
- In some cases, supplier companies have assigned engineers with no software experience to develop complex machine-control software, without even considering buy-versus-make tradeoffs.
- Managers without software experience have talked with us about software engineering in terms that equate it with coding, thinking that software is simply a matter of coding.
- One supplier has had seven unfilled position on its software staff for more than six months, unable to find people with experience in semiconductor manufacturing and software.
- Equipment suppliers have reported that they sometimes have received specifications for equipment interfaces the day before the ship date for a piece of equipment.
- Suppliers and manufacturers both acknowledge that technical and nontechnical requirements frequently change throughout product development. Verbal communications, no change control discipline, and little change impact assessment have created chaos in some projects.
- Some supplier companies, describing their product development teams, indicate that software engineers get involved after decisions are made about product functionality and design.
- One supplier, in response to questions from a customer, found more than 20 versions of its software in one fab, in spite of using a tool for configuration management and release control.
- Supplier executives have told us they use anyone who has a “systems orientation” to do system design and engineering; few hire staff that are trained and experienced system engineers.

The equipment industry, in general, has had difficulty adapting to its domain the advances developed in the rest of the software industry during the 1980s. Some of this difficulty is due to the uncompromising requirements in semiconductor manufacturing. For example, the notion of time: If a management information system (MIS) slows down during peak demands, the user may notice longer response times. But this is quite different from a robot arm crashing into a machine because a command was received too late.

Your company as a supplier to semiconductor manufacturers can reap substantial and tangible benefits through a focused and well supported software improvement initiative. Companies in different business sectors are reporting successes with such programs [12]. Typically, they report returns on their investments between 4 to 1 and 9 to 1, with approximately two years between startup and observable benefits. In some companies, this time delay taxes the attention span of managers, especially middle managers, who are delivery-oriented and in many cases crisis-driven and therefore resistant to change.

Under the best of circumstances, changing the technical practices and culture of an organization is not a simple matter. Motivation that is strong and well founded is critical to success; it must drive the entire organization. People at all levels must embrace the idea of continuously improving the way they work, thereby making their company more competitive.

High quality software has payoff aside from meeting the specs in a customer's order. It can improve your market position and image, strengthen your balance sheet, and result in less tangible but very important benefits such as decreased staff turnover, pride in workmanship, and increase morale.

SEMATECH's SPI Project exists to help the members of SEMI/SEMATECH realize these benefits.

2.4 SEMATECH Helps Suppliers Improve Software

SEMATECH is a consortium of U.S. semiconductor manufacturers. Its members are Advanced Micro Devices, AT&T, AT&T Global Information Systems, Digital Equipment, Hewlett-Packard, IBM, Intel, Motorola, National Semiconductor, Rockwell International, Texas Instruments, and the Department of Defense.

SEMATECH's mission is "to solve the technical challenges required to keep the U.S. number one in the global semiconductor industry." Software is one of the industry's technical challenges, so SEMATECH has chartered a project—the SPI Project—to improve operational reliability of software that member companies acquire from suppliers.

The SPI Project considers all members of SEMI/SEMATECH that deliver software to member companies' fabs as colleagues in the work of software improvement and as the audience for this document. The manufacturers, the project's primary beneficiaries, expect results that include fewer defects in delivered software, higher mean time between interrupts for equipment in the fabs, lower cost of ownership, on-time delivery, and products that are easier to integrate and use.

Many groups throughout the industry are working to improve software. Several SEMATECH member companies and some suppliers have improvement initiatives within their own organizations. Some manufacturers also partner with their suppliers to improve software quality. In addition, various SEMATECH projects involve software improvement, including an ongoing initiative to improve the consortium's own software process.

SEMATECH's SPI Project fosters cooperation among these various efforts. But you, the "change agents" and "improvement champions" in each supplier company, are the keys to success. Your roles are central to software improvement within your respective companies.

This handbook is to orient you and your colleagues to your task. Part of that orientation is to coordinate your work with other participants. The primary participants in the project's work, and the role of each, are as follows:

- SPI Project staff at SEMATECH: Planners of the proliferation initiative and coordinators of its implementation, providing expertise and support services to the full extent of available resources.
- Manufacturers: The initiative's primary beneficiaries, but also committed to partnering with suppliers for software improvement and being responsible users of software, including specifying and procuring it.

- Commercial service providers: Sources of additional expertise in software improvement and related matters. Consultants who have worked with the SEMATECH program are available to suppliers at competitive market rates.

The document also provides a basis for networking and communicating with “change agents” and “improvement champions” in other supplier companies about your work on software improvement. But our primary objective in this presentation of the guidelines is to show how to produce higher quality software products, as follows:

- Section 3 leads off with some basic information about software and the work of producing and supporting it. It outlines several basic capabilities that every software organization must master. Without these capabilities, an organization cannot be a reliable provider of quality software. We believe each supplier company will find this information useful in its effort to improve software.
- Section 4 moves beyond the basics to project a vision for software in this industry. The vision anticipates benefits for suppliers as well as the manufacturers. The guidelines as presented in Sections 3, 5 and 6 are our recommendations for implementing this vision for software.
- Section 5 introduces the Maturity Model we recommend. It is important for you to realize that the Model does not depict a software process; i.e., it does not model the flow of activities that constitute a software process. Rather, the Model gives characteristics of maturity that can be used to evaluate your software process. This section introduces you to those maturity characteristics.
- Section 6 presents an Improvement Method for using the Maturity Model. The Method is not a one-size-fits-all approach to software improvement. Rather, it is a set of templates that you use to develop a plan that addresses the needs and situation of your particular company. This section introduces those templates and their use.
- Section 7 discusses how to apply the model and Method to your needs in your situation. Its focus moves beyond improving the process to improving the software products you deliver. This involves the use of product quality baselines, goals, and metrics.

Throughout this material are references to documents that contain additional information.

3 SOFTWARE IMPROVEMENT FOR SUPPLIERS: SOME BASICS

Improving the quality of software is a hot topic. At an increasing rate, organizations in various segments of business and industry are committing resources to improve the software they produce and support. Lessons learned from these efforts are providing valuable insight into the basics of how to do it.

During the past two years the SPI Project at SEMATECH has helped launch, and is continuing to support, improvement initiatives in six companies. This experience within our own industry, combined with insights from outside, has ignited interest in getting improvement initiatives started in more of the supplier companies.

This section identifies six basic functions that serve as starting points for work on software improvement. Many companies have one or more of these functions operational, but very few

have implemented them all. With even a first-level capability in each area, an organization puts itself on the road to software improvement.

3.1 Software: What Is It?

Software is a sequence of instructions for a computer to execute [13]. Two synonyms for software are computer code and computer program. Each of these three terms says something that is important to hear:

- Software—we are dealing with something distinct from “hardware.”
- Computer code—ultimately, the representation is for computers to process, not humans.
- Computer program—the flow of instructions has meaning in addition to the meaning of individual commands.

Putting this together says that software is a body of structured information with logical, functional, structural, and behavioral properties.

Software must be specified and documented in ways that adequately describe these several properties. This is a challenging task in developing the work products at each stage of software development:

- Concept document for operations and use
- Requirements specification
- Interface control documents
- Design Documentation
- Test Plans
- Commented Source Code
- Training Materials
- User’s Manual and Principles of Operation
- Maintenance Guide

You can be somewhat informal about these work products for small amounts of software, but as software increases in size and complexity, being informal leads to misunderstandings, confusion, and defects in the finished product. It has been said that

*Programming is the activity of developing a computer program for personal use;
software engineering is the activity of developing software for use by others.*

Many companies today are trying to produce industrial strength software with programming. The results include low-quality products, high production and maintenance costs, and unhappy customers.

3.2 Software Has a Place in Your Products and Business

All major pieces of equipment in a semiconductor fab depend on software for correct operation. Most reconfigurations and upgrades of this equipment are accomplished by changing software. In addition, software has become essential to the functioning of a fab, just as in most other

businesses today. So whatever systems or equipment you, as a supplier, are delivering into the fabs, the role and place of software are significant for your business.

Earlier we spoke about the challenge of software for suppliers (see Section 2.3). Here we want to make the point that the development and modification of industrial strength software is a difficult engineering problem. It is difficult for the following reasons:

- Software engineering is an intellect-intensive, knowledge-based, team-oriented discipline; people are the most important resource.
- Products with embedded software must be high integrity, well-designed systems that appropriately allocate functionality to hardware, electronics, and software.
- There are significant differences among hardware, electrical, and software engineering. Products with embedded software require interdisciplinary teams that encompass all three of the disciplines—plus systems engineering.
- Software involves logical complexity and attention to detail that has a non-linear relation to size.
- Ongoing product maintenance often competes for resources with new development efforts, and multiple product versions intensify the competition.

Software engineering, to be effective, requires the participation of managers at various levels. In some instances, you may buy the software you need; in others, your organization produces it; and in all cases, someone supports it. Software is, without a doubt, an important part of your business.

3.3 Know How You Do Software

We have worked with executive managers who readily admit that they know much less about their software operations than any other part of the business. In their companies, the work of developing and supporting software is done within a black box.

This situation notwithstanding, management is responsible for the software delivered to customers. So when a manager at any level has abdicated responsibility for software, the organization has a problem to solve. But the problem can be solved.

The solution is to know how your organization does software. Know the steps involved; know the complete process. While there are many ways to develop and support software, and organization does well to agree on what “our way to do software” is.

Increasingly, customers want to know how their suppliers do software. In some instances they even set standards. So you might want to make your customers—at least certain ones—a party to your agreement on how to do software.

One approach to reaching an agreement is to write down how you do software. A brief description is sufficient. This is management’s opportunity to ensure that the software process is in fact manageable and that it meets your customers’ requirements. Many people find that the very act of documenting the process provides insight for improving it.

Once you have agreed on how you do software, the final step is to follow the agreed-upon way with consistency. That does not preclude agreed-upon variations. When a variation from the

basic approach is needed, make it—but not unilaterally, because variations, just like the basic approach, should be agreed upon by all parties.

Work it out. Know how you do software. Be sure the way you do it is manageable and be sure you manage it. Then do it that way consistently, making agreed-upon variations when needed.

It's not rocket science—just common sense—and good business.

3.4 Doing Software Involves More than Coding

When a company keeps its software activity in a black box, people outside the box tend to think that “doing software” is equal to “coding.” Although coding is important, it is but one of the several activities involved in doing software. Reviewing the software life cycle reveals the variety of activity.

The cycle begins with a needs assessment followed by requirements collection and analysis. Then there is system design and (in the case of an equipment system) allocation of functionality to software, followed by some further design work on the software subsystem.

Only now is there some coding; it should include unit testing of the code. Inspections of the requirement and design documents, as well as the coding sheets, is again distinct from coding but a very important activity. And then there is testing the modules of software as an integrated system, first in your own facility and then at the customer's site.

Any software development cycle also relies on support activities that are quite distinct from coding. These include configuration management of the software and related documents, defect documentation and tracking, project planning and tracking, product documentation, and user training.

A baseball team is a good analogy. A group that can throw, catch, and hit the ball is not enough. You need some diversity in how they throw and catch; their skills and experiences must match the very different requirements of catcher, pitcher, short stop, and center field. And that's not all; they must be able to hit the pitches thrown by different pitchers; fast balls, curve balls, knucklers, and others.

The message here is that coding is that the only dimension of software engineering. And these dimensions are discussed throughout the rest of this document.

3.5 A Software Effort Produces More than Code

People who equate software with coding also tend to see code as the only result from software engineering. But again, there is much more output than the code itself.

Several artifacts of the development cycle are widely written and talked about, but in practice they are frequently passed over—that means not done. These include documented requirements, design documents, test plans, and defect reports. There are also support documents such as user manuals, maintenance guides, and training materials. In most cases, all of these are important and deserve a ranking equal to code as outputs of a software effort.

An even more important result than the above is factual information about the effort. Such information, absolutely critical for software improvement, includes schedule data, both plan and

actual; resources data, both plan and actual; activities data, including unscheduled activities such as rework; and results data, including code volumes with defect descriptions and densities.

Preserving and providing access to historical information about the software efforts your company has completed is a prerequisite of software improvement. It is not an overstatement to say “You cannot make improvements without it.”

3.6 Shortcuts Often Make the Path Longer

The guidelines for software offered thus far can be summarized as follows:

- Recognize software as a significant part of your business and software engineering as a professional discipline.
- Know how your organization does software and describe it; be sure that management and customers are in full agreement.
- Provide for the full range of competencies that software engineering requires, not just coding.
- Give priority to and schedule time for software results other than code, especially factual information about each project.

Organizations that focus on coding ignore and even resist such guidelines. Their resistance shows in how they formulate their business strategies and plans, hire and compensate software staff, schedule and resource software projects, and communicate with their customers.

If your goal is defect-free software—or as close to that as is economically beneficial—and your are entirely serious about that goal, then it’s important to know what’s involved in reaching it.

The first point is that there will be some defects in the software you produce; you can’t avoid it. But you can do two things about it: first find and remove defects as soon as possible after they are injected; and second, work on minimizing the number of defects in your software.

Techniques exist for doing both, and the economics are straightforward. Defects are most expensive to fix after software is in the customer’s hands. The reverse of this fact is that the cost of defect removal is minimized by identifying and removing defects as soon as possible after they are created.

There are two ways to reduce the number of defects created. First, don’t take shortcuts; they increase the number of defects. Some of the most common shortcuts that increase defect counts are as follows:

- Deficient requirements for software
- Incomplete or incorrect designs
- No inspections of requirements, designs, or code
- No testing or inadequate testing of code

The second is to analyze the defects that are created to determine what caused them to occur, then to make improvements designed to avoid such defects in the future.

In many software organizations, rework is a major component of development cost, and the cost of software maintenance can exceed the cost of development. How you deal with software defects is an important matter.

Maybe this observation makes the point: If it's worth doing, it will cost less to do it right with competence.

3.7 A Checklist for Your Software Operation

Managers who are without software experience sometimes try to avoid the subject, thinking software can get along without much attention from management. But that approach leads to many problems.

This section provides a checklist that will enable you, as a manager or other software leader, to quickly get some insight into the completeness and the health of the software activities in your company. It will help you spot capabilities that should be in place but are not and to recognize where improving should be considered.

Using this checklist does not take the place of a formal appraisal of your software capabilities. Nor is it designed to guide your company's software improvement effort. It simply gives you some idea of where you are in the software component of your business.

How You Handle Requirements

- Do you have written requirements for your software projects?
- Do you base project work estimates on an analysis of the requirements?
- Do you analyze the software requirements for testability?
- Do you control changes to requirements?
- Do all affected groups agree to the requirements received and to changes?
- Do you validate your understanding of requirements and changes with the customer?

How You Plan Software Projects and Track Them

- Do software engineers and managers participate in and agree to commitments on product functionality and delivery schedules?
- Do you base plans for software projects on requirements and the people available to work on the project?
- Do you base plans for software projects on the product and schedule commitments made to customers?
- Can you document the estimates and schedules made for software projects and use them in project tracking?
- Does management track the actual results and performances of a software project against the project's plan?
- When a software project deviates significantly from its plan, do you take corrective actions and manage them to closure?
- Do all affected groups agree to any changes made to software commitments?
- Are software engineers involved in identifying, tracking, and resolving interdisciplinary issues?
- Does your organization qualify in a formal way the software suppliers you select?

- Do you track the actual results and performance of a supplier against the commitments made?
- Do you and your software suppliers have regular communications that allow you to quickly resolve misunderstandings?

How You Manage Software As You Develop It

- Are the intermediate results of your work on software—requirements, project plans, designs, code, maintenance and user documentation, test plans, etc.—kept organized and made available throughout the project?
- Do you control all changes that affect the intermediate results of your software work?
- Do you establish baselines and identify versions of the software that you have under development?
- Do you have a formal release-to-manufacturing step in your work on software?
- Do you keep those with a need to know informed about the status and content of software baselines?
- Do you have a formal release-to-the-customers step in your work on software?

How You Do Software Engineering

- To produce software, do your software engineers consistently use a sequence of well defined and integrated tasks?
- Do your software engineers use tools to automate their work and to do tasks that otherwise could not be done?
- Do your software engineers formally test the software that they produce?
- Do you verify that your software products and the way they are produced adhere to any applicable standards, procedures, and requirements?
- When you have a nonconformance issue that cannot be resolved by the software engineers, does upper management get involved?

How You Handle Peer Reviews and Defect Tracking

- Do you use peer reviews to inspect software?
- Do you inspect any of the other work products of software development?
- Do you document software defects identified in the work products inspected and then track them until they have been corrected?
- Do you document and track the defects identified in ways other than inspections, especially those reported by customers?
- Is it your objective to remove defects from any software work product within the work phase that produced that defect?

How You Test Software

- Do you test the software you produce?
- Do you test software by following a well defined testing procedure?

- Do you test software at more than one point before releasing it to manufacturing or the customer?
- Does someone test software other than the person who developed it?
- Do you include testing in the scheduling you do for projects and product requirements?
- Do you base your plans for testing on the project and product requirements?
- Do you do regression testing on each new release of software to customers?

How You View Software Activities

- Have you defined and documented how you do software, both at the level of software engineering and at the level of planning and managing projects?
- Are you aware of strengths and weaknesses in how you currently do software?
- Do you periodically capture the “lessons learned” about how you do software and use them to improve?
- Do you have staff that is chartered to improve how you do software and the quality of the software you produce?

How You Do Measurements Related to Software

- Do you measure the following for your software projects: software cost, effort required, time required, and the quality of the finished product?
- Do you count and evaluate software failures, defects, and unresolved problems?
- Do you measure the operational reliability of your software?
- Does management use measurements and other factual data for visibility into software activities and to make decisions related to software?
- Have you compiled historical data about your software activities and products and used it to establish baselines for your company?

How Extensive Is Your Education, Mentoring and Training

- Do you provide educational opportunities for developing the skills and knowledge needed to perform software management, software engineering, and quality engineering?
- Do you provide training in software to other disciplines in your company that depend on software technology?
- Is your organization building a more capable and efficient “software-smart” workforce, able to communicate effectively about software topics?

3.8 Solving Some Common Problems

The guideline of first importance is for you to make a commitment to software improvement, put some resources behind your commitment, and get to work on one or two problems that are basic and that are visible to your customers.

If you are not sure where to start, we recommend that you review the six common problems discussed below. The all deal with basic capabilities in producing software; solving them will

make a difference to your customers. Most supplier companies we know are without some of these capabilities, and more than a few have none of them. Any one of the six is a good place to begin work on software improvement.

The scope of this document limits us to a brief discussion of each problem and its solution. The SPI? Project is prepared to provide additional assistance by defining materials and process assets. Technical support for developing robust practices in any of these areas is available from various commercial suppliers of SPI services at competitive market rates.

3.8.1 Configuration Management

Software configuration management (SCM) is a set of tracking and control activities [14]. SCM begins when a software project starts and terminates when the software is taken out of operation. Most often SCM is applied to the code itself, but it should also include requirements, designs, user manuals, and test plans.

The purpose of SCM is to identify change, control change, ensure that change is being properly implemented, and report change to others who have an interest.

For suppliers in this industry, change is a fact of life for software development. Customers want to modify requirements. Developers want to modify the technical approach. Management wants to modify the project approach.

Why so much change and so often? The answer is simple: as time passes, all constituencies know more about what they need, which approach is best, how to get it done and still make money. This additional information drives most of the changes.

SCM is an important element of software quality assurance. Any discussion of it involves a set of complex questions:

- How to identify and manage the many versions of software and its documentation?
- How to control change before and after software is released to the customers?
- Who has responsibility for approving and prioritizing changes?
- How to ensure that changes have been properly made?
- What mechanism to use to apprise others of the changes made?

These questions lead to the definition of five basic SCM tasks:

- Identification—This task controls and manages the items put under SCM. A variety of automated tools are available that make the current form of an item immediately available and other versions easily available.
- Version Control—This task allows one to specify alternative configurations of the software system through the selection of appropriate versions.
- Change Control—This task combines human procedures and automated tools to provide a mechanism for controlling change.
- Configuration Auditing—This task complements the formal technical review by assessing characteristics that are generally not considered during the review.

- Reporting—This task answers questions such as What happened? Who did it? When did it happen? What else will be affected?

Several ANSI/IEEE standards apply to SCM, such as 828-1983, 1042-1987 and 1028-1988 [15]; these are recommended for both large and small software engineering organizations.

3.8.2 Peer Reviews (Inspections)

Software has defects. The best programmers on their best days write defects into the code they produce. And defects in code result from other causes, too, such as defective or incomplete requirements, incorrect designs, or unanticipated side effects of changes to the code. The goal is to have software that is defect-free, but the reality is that defects are there.

Software inspections have proven to be the most effective way to find defects [16]. Before inspections were used, defects in software generally were detected during testing. Inspections allow defects to be removed earlier in the development cycle when it is less expensive.

Inspections are formal peer reviews of the material being inspected. They are formal in that they follow a set agenda. A limited amount of material is covered in a set amount of time. Each person participating has a clearly defined role. All issues raised at the inspection are rigorously recorded and subsequently resolved.

Inspections identify errors that the author of the material cannot find. If the errors detected are analyzed to determine their causes, such defects can be prevented in the future. Finally, inspections help train those who participate to recognize defects and to produce work with fewer defects.

3.8.3 Project Planning and Tracking

Some suppliers establish software projects that support their product development programs, while others make software an integral part of product development. Still others do not have formally defined projects; they simply “get the products out.” In any of these settings, project planning and tracking brings an organized and disciplined approach that increases productivity and effectiveness.

Planning and tracking a project is a management activity. Many suppliers acknowledge that they inadequately manage in this area. This is not the only industry in which that happens, but the good news is that it’s a situation that can be corrected.

A third important issue has to do with the scale of projects. Much software in supplier companies modifies existing software, i.e., develops new features, enhances old features, and fixes known weaknesses and problems. Only occasionally is there a project that develops a major new software system from scratch. This means much of the work on software is done in an interdisciplinary setting, by only one or a few software engineers, and over a short project cycle.

Each of these issues influences the implementation of software project planning and tracking [17]. And it is up to you to decide how to implement it.

Software project planning involves developing estimates for the work to be performed, establishing the needed resource commitments, and defining the approach and schedule for performing the work. These estimates, commitments, and plans provide the basis for performing

and managing project activities. They are also useful in communications with customers and in monitoring the risks and other contingencies related to the project.

Tracking and oversight gives management visibility into the actual progress of a project so that corrective actions can be taken when performance deviates significantly from the plan. Corrective actions could include changing the resources allocated, the scope of the project, or the project schedule. Tracking and oversight also prepares management to communicate accurately with customers, other project teams, and other departments in the company about project status. It's not rocket science, but it makes a big difference when done practically and effectively.

3.8.4 Software Testing

At its best, software testing exercises software under carefully specified and controlled conditions. It observes or records the results and evaluates the software against expectations.

Actually, this definition describes only the second half of the testing process; the first half deals with developing the testware itself. This part of the process includes

- Determining requirements and designs for the testware code, data and procedures.
- Creating, acquiring, or stimulating representative situations the software must handle.

In addition, during test execution the testware must be evaluated along with the software being tested [18].

Some of the work products of software testing are as follows:

Test Objectives—Statements of observable behaviors the software must exhibit during execution.

Testware Design—Description of what will be used and done to exercise the software under test.

Test Cases—Data values and event sequences needed to demonstrate behaviors of the software under test.

Test Procedures—Specification of the required sequences of test cases; timing of event sequences, durations, and overlaps; and the manipulations required to set up, initiate, and observe the test.

Test Logs—A time-stamped record of test execution; unexpected results are documented in the log.

Test Coverage—A measure of the “thoroughness” of the test, including coverage of the test objectives and the software mechanism being tested.

These work products have meaning for the testing that is done in each stage of the software life cycle:

- Unit testing during development
- Integration and system testing prior to product release
- Installation testing at the customer's site
- Regression testing after making modification to the product

To establish testing in an organization, first focus on producing the specific work products used in testing, proceeding as follows:

- Integrate the testing process with the development process.
- Document the software failures and fixes.
- Automate test case execution.
- Establish testing objectives.
- Measure actual test coverage.
- Establish test coverage objectives.

Once the work products have been developed, you can improve the methods used to produce them.

3.8.5 Defect Tracking

Defect tracking supports the work of eliminating defects, a high priority matter in software improvement [19]. You realize its importance when you deliver software to your customers—only to have the customer discover and point out defects. And if anything bothers a customer more than discovering a software defect, it is discovering a defect for a second time—after you declared it had been fixed.

A defect tracking system allows tracking defects in accordance with a defined process that typically has automated support. Such a system supplies the information needed to do root cause analysis. This gives insight into which aspects of the software process require improvement, and predicts software problem areas to what appropriate, corrective actions can be taken.

The following capabilities are currently available in commercially available defect-tracking tools:

- Record and track problems through resolution
- Report on the quality of a product
- Determine readiness for the next work phase, including release
- Indicate high error-density components needing rework
- Identify error-prone process areas
- Define a basis for quality improvement

We do not recommend developing your own software for defect tracking.

The capabilities listed above allow a defect tracking system to answer the following questions:

- When are defects discovered?
- Who is discovering the defects?
- Which software components have the most defects?
- How fast are the defects being handled?
- Which and how many defects have been handled in a stated time?
- What is the severity of the defects currently unresolved?

It is important to have an effective working relationship between defect tracking and customer support services, whether these services are based at the home office or out in the field.

Defect tracking is a practical way to begin work on software improvement. It addresses the central issue: defects. It has visibility with your customers and can contribute quickly and directly to improved customer satisfaction (unless you have never had a customer uncover a defect in the software you delivered). And finally, defect tracking is a particularly effective base for building a metrics and measurements initiative within a software organization.

3.8.6 Software Requirements

Development: a process that transforms someone's desires into a product that satisfies those desires. The requirements process is that part of development at which people attempt to discover what is desired [20].

Five words in that definition deserve special notice: desire, product, people, attempt, and discover. They signal the challenges of working with requirements. And if you watch how people really develop systems, you will see that the requirements process is actually a process of developing a team of people who

- Understand the requirements
- Stay together to work on the project
- Know how to work effectively as a team

the requirements for a software project include both functional and performance requirements and requirements for hardware, software, firmware, and user interfaces. They may also specify development or quality assurance standards for both the product and the project that develops the product. Requirements can be stated either quantitatively or qualitatively, but the preference is to quantify.

The first issue with requirements is their source: Whose requirements are they? And Who takes responsibility for them being complete and correct? The second issue is communicating with that source: Will contact be directly or through third parties like sales and marketing people, managers, and/or system engineers? Will requirements be documented or verbal? And will there be playback of how the requirements are understood? The third issue is dealing with change: Who is authorized to change requirements? Is the cost of a change a mater to be considered?

Some challenges for those working on requirements are as follows:

- User needs analysis is often a conflict-based process in which negotiation takes place over the life cycle of a system.
- Requirements are as complex to manage as the system they represent.
- Insufficient clear requirements at project initiation forces assumptions to be made that may be wrong.
- Turning requirements into systems that meet the requirements is an extremely dynamic task with much fluctuation.
- Several parties must understand and use requirements for different purposes, leading to multiple requirement specs.
- Responsiveness to users conflicts with requirements stability.
- Technology shifts pressure a system architecture to change form the bottom up.

It is well worth resolving these matters with care because it costs between five and ten times more to repair errors during coding than during the requirements phase and between 100 and 200 times more during maintenance [21].

3.9 Improvements Beyond the Basics is Good Business

A company with even an initial capability in each of the six areas cited above has accomplished something significant. Customers will see the results and these functions, once institutionalized, provide the foundation for an ongoing effort to improve software.

But suppliers in this industry, as elsewhere, sometimes try to improve software by buying new technology in the form of tools or methods. The guidelines in this document certainly do not preclude that, but they have a different focus: on buying technology to make the defined software process more effective.

This focus means making the process complete, practical, and efficient. It also means improving the skills of managers and engineers who execute the process. Staff improves as a result of training and by repeated and consistent use of an established process. As maturity increases, new tools can more easily be incorporated to automate the process.

Documented case studies show that software improvement through overall capability improvement is good business. But these studies clearly show that each organization must assess its own situation, establish its own baselines and goals, and set about earning for itself a return on what it invests. As is so often true, in SPI nothing succeeds (or motivates) like success.

4 A VISION FOR SOFTWARE

The cost of capital equipment in the semiconductor industry is rising dramatically in an increasingly competitive market. This places a heavy burden on factories to operate more cost effectively. That means they must achieve higher equipment utilization, more aggressive cycle-time goals, and increased equipment throughput. To meet these goals, the causes of capacity loss (the difference between theoretical and actual factory capacity) become matters of increasing concern. Studies are now being conducted to identify more clearly what causes capacity loss in fabs and how to reduce or eliminate those causes [22].

Based on currently available information, unscheduled equipment downtime is a leading cause of capacity loss, and software is a major cause of unscheduled downtime [23]. From this perspective, our vision for software in semiconductor fabs is

Software becomes an insignificant factor in any aspect of capacity loss for fabs.

More needs to be known before projecting a time for achieving this vision, and there are no quick fixes for software quality problems. But many software problems are obvious, and solutions to these problems do exist. So the balance of these guidelines connect the near-term, quick-result efforts discussed in Section 3 to achieving the vision for software articulated above.

These guidelines are based on the view that the vision can be realized only by an assault on all aspects of the software problem. Achieving results now is very important and is possible, but we must proceed in a manner designed to achieve a long string of results.

The view taken in Section 3 is pretty much “It’s broken and needs to be fixed, so let’s fix it right now.” To support the work of software improvement over the long term, we add to this focused, results-oriented approach two important formalisms or standards:

A Capability Maturity Model for Software

A Method for Using the Maturity Model to Improve Software

These resources are presented and discussed in Sections 5 and 6, respectively.

What does it mean to assault all aspects of the software problem? To improve software involves people, technology, and process. Figure 5 shows these three as related to each other, and that they, in combination, determine an organization’s capability in software.

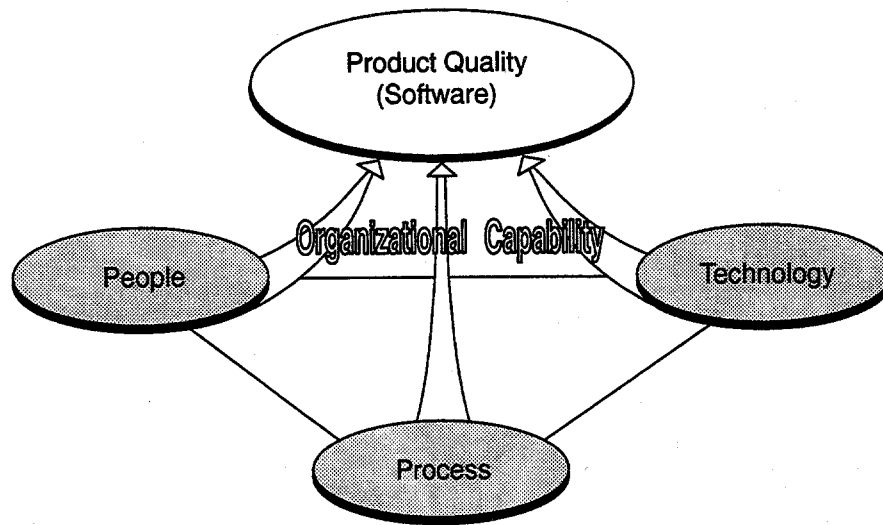


Figure 5 The Three Leverage Points for Improving Software Products

An organization expresses its capability in software through its software process—the interplay and net effect of its people (managers and engineers) and its technology. The process defines the way people and technology work together to produce and support software. People, technology, and process are leverage points for increasing the maturity of an organization’s software capability, and also for increasing software product quality.

Think of a three-legged stool, with these three components as the legs. Raising any one leg or any two of them may temporarily give the impression of raising the whole construct; but ultimately this will tip it over instead of lifting it. Only by raising all three legs simultaneously will the increase be a stable change.

A typical American approach to the software process is technology-centered. The relative abundance of cheap hardware and a certain ingrained pride in American inventiveness cause many to believe that no problem can resist an attack with the right tools. By contrast, the Japanese and Europeans typically stress engineering and management aspects of the software process. These guidelines foster a balanced approach; all three components (process, people, and technology) are required to improve software quality.

Organizations in other industries that have committed resources to improvement initiatives similar to the type anticipated here have achieved attractive returns on their investment in the range of 4 to 1 and 9 to 1. But we must acknowledge some significant challenges or even barriers to achieving the vision:

- A resistance to change among people in general, and in this situation among software engineers and managers alike.
- Economic pressures and commitments to customers that work against changing the process.
- A rate of technology change and intense competitive pressures that make requirements chronically incomplete and variable.
- Realization that software is always written with defects, so that defect detection and removal must be part of the software process.

These guidelines recognize and deal with each of these realities.

Before discussing the Maturity Model and the Improvement Method, we will briefly address some topics that are important for an ongoing, industry-wide effort to improve software:

- The role of manufacturers in realizing the vision
- Motivations for the suppliers to help realize the vision
- Definition of continuous improvement of a software process
- The way software process improvement also improves products.

4.1 Flawless Software in Fab Operations

If flawless software in the fabs (or something close to it) is to become a reality, what can manufacturers do to help achieve it?

Manufacturers purchases some software used for fabs; the balance the manufacturer provides, combining or interconnecting it in various ways with software from suppliers. That means these guidelines apply to software improvement within the manufacturers' software organizations as well as in supplier companies.

Beyond that contribution to software quality, the manufacturers also are a direct influence on the quality of software that they purchase. For example,

- **Statements of Requirements.** A manufacturer is the source of requirements for software, and the quality and completeness of requirements has a significant influence on software quality.
- **Definition of Quality.** Software quality has many aspects, such as operational reliability, functionality, usability, maintainability, and efficiency. It is up to the manufacturer to assign priorities, thereby defining software quality.
- **Procurements that specify Quality.** Having defined software quality, a manufacturer's procurement specifications must require software quality if it is to be delivered.

There are additional ways for manufacturers to positively influence the quality of the software they purchase:

- Partnering with a supplier for software improvement
- Participating in a supplier's development team

In these ways, a manufacturer communicates exactly what software quality means, why it is very important, and why the software that is purchased must have it.

4.2 Business Success Through Software

Software is here to stay—in semiconductor manufacturing and in the businesses of its suppliers. But what is not at all certain is how well a particular company will do software. Why should you, as a supplier, make a commitment to improve software?

Observers report that the capability gap between the best and the average software development organizations is widening. The best will become five times more productive than the average and will produce products that are 100 times higher in quality than the average. The higher performance will not come from any particular improvement but from a series of incremental improvements [24].

The guidelines in this handbook are to help your company make incremental and significant increases in your software capability, beginning in the short term and continuing over the long term. Some guidelines that head you in the right direction are as follows:

- Make sure that your software process covers all essential functions and that it is manageable.
- Make sure that you follow your software process and that managers are adding value to the process.
- Ensure that software engineering participates fully with the other disciplines in your product development process.
- Schedule the activities that software engineering involves and provide the skills and experience the activities require.

Years ago, Phil Crosby, a founder of the quality movement, made a convincing case for the assertion that “quality is free.” At a recent symposium on software quality, a response to that claim went like this:

Quality may be free, but nobody is giving it away. You only get it the old fashioned way—by earning it.

These guidelines are to help you, as a supplier, earn it. And these are some of the opportunities available to you:

- Decrease the cost of software development:
 - define your software process; follow it consistently
 - inspect code, designs, requirements; fix the root causes
 - do project post mortems to reduce rework; fix rework causes
 - increase productivity through training, process, tools
- Decrease the cost of software product support:
 - reduce the number of delivered defects

- increase productivity of staff doing product maintenance
- Deliver quality to the customer:
 - focus on and put effort into documented requirements
 - set a goal to increase MWBI or another reliability measure
 - do testing that reduces defects in delivered software
- Reduce the cost of ownership for your equipment:
 - all of the above
- Recognize that some less tangible benefits are also important:
 - lower staff turnover
 - pride of workmanship
 - higher staff morale

It literally is up to you. The tools exist. Using them in your company is up to you.

4.3 Continuous Improvement of a Software Process

Following these guidelines, and using the resource for improvement discussed in Sections 5 and 6, the continuous improvement of a software process works like this:

- An organization's software process is the set of steps it takes to produce software. Your software process may be
 - Roughly outlined or not defined at all.
 - Immature relative to a maturity model, or very mature
 - Executed by managers and a software staff that are well trained and experienced or by staff that is deficient in the requisite experience and skills

In any event, every organization that delivers software to its customers has a software process.

- The Maturity Model gives the characteristics of a mature software process, plus characteristics of a capable staff and of effectiveness in the use of technology. (the Model's key process areas express these characteristics.)
- Using the Maturity Model, the company comes to understand its software process in terms of the Model's characterization of maturity. It does this on a recurring basis. Guided by these understandings, the organizations makes process changes that both improve the quality of its software products and increase the maturity of its software process.

This advance—from making improvements in either an ad hoc manner or not at all, to a continuously working, model-directed improvement system—puts an organization on the road to software quality and process maturity. As the improvement system becomes institutionalized, ideas for what to improve come from many sources:

- Formal assessments using the Maturity Model
- Inspections and root cause analysis of defects identified
- Project post mortems focused on how to avoid rework

- Software engineers gaining experience with a defined process
- The vast literature on software process
- Networking with others interested in software improvement
- Out of the blue

The first purpose of this handbook is to provide guidance in establishing and operating an improvement system that makes effective use of ideas for improvement, whatever their source. Its second purpose is discussed in the next section.

4.4 Improving Products Through Process Improvement

Using the Maturity Model to improve your software process and your organization's overall software capability does not automatically result in higher quality products. You must define and set priorities on the types of outcome you want to see. We recommend that product improvement be one type, but there may well be others to support your business case for the improvement initiative, such as less time spent on rework. You must also establish ways of measuring each type. That way you will know if you are getting there and at what pace.

Some of the activities that connect software process improvement to higher quality products are the following:

- Define quality. Because software quality has so many different aspects, determine which has highest priority and use customer inputs in making that determination.
- Define defect and failure. Because there are many kinds of software defects and failures, establish category and severity codes and be clear about which kinds you want most to eliminate.
- Determine product quality. Figure out how your current products stack up against your definitions of quality and defects and on that basis set a realistic improvement goal.
- Use inspections to achieve the improvement goal. Do this by taking the following steps:
 - locate defects with inspections and fix the defects
 - do root cause analysis of the defects identified
 - make process improvements that reduce defect injections
- Use the Maturity Model to achieve the improvement goal. Do this by taking the following steps:
 - use assessments to identify candidate improvements
 - evaluate candidates for their product improvement potential
 - give priority to those candidates that improve products

This gives you the idea; Section 5.3 adds some additional detail. The Maturity Model spawns ideas for many types of improvement. It is up to you to provide the additional focus needed to reach outcomes that match your intentions.

4.5 Implementing the Vision

Making changes that permanently improve the quality of software can be a daunting task. The Model and Method discussed in Sections 5 and 6 are resources that help. Major elements of the context in which these resources are to be used are as follows:

- Make a business case for software improvement that applies to your company; know why you are committing to do it.
- Secure commitments from staff throughout the company, including both managers and technical staff.
- Get familiar with your current software process and document it in a way that points up its strengths and its weaknesses.
- Build an infrastructure to support change—people who are chartered, experienced, and equipped for the work to be done.
- Get thoroughly familiar with a model of maturity for software that will serve as an improvement roadmap.
- Define what you mean by software defects and software quality, and set goals and metrics for your improvement initiative.
- Incorporate into your software process a method for using the model, one that is tailored to your company.
- Repeatedly apply the model and method throughout your software process, measuring the impact of change and progress to goals.
- Ensure that the software improvement initiative has a positive impact on your business goals and strategies.
- Never, ever give up.

This is not an ordered list of things to do, and you certainly cannot do them all at once. But knowing what elements make up the big picture helps maximize your return from each task.

Although it is not possible to attend to all of the elements listed at once, Figure 6 clearly shows the negative effect of missing ingredients. These guidelines for software improvement help you avoid there “missing ingredient” experiences.

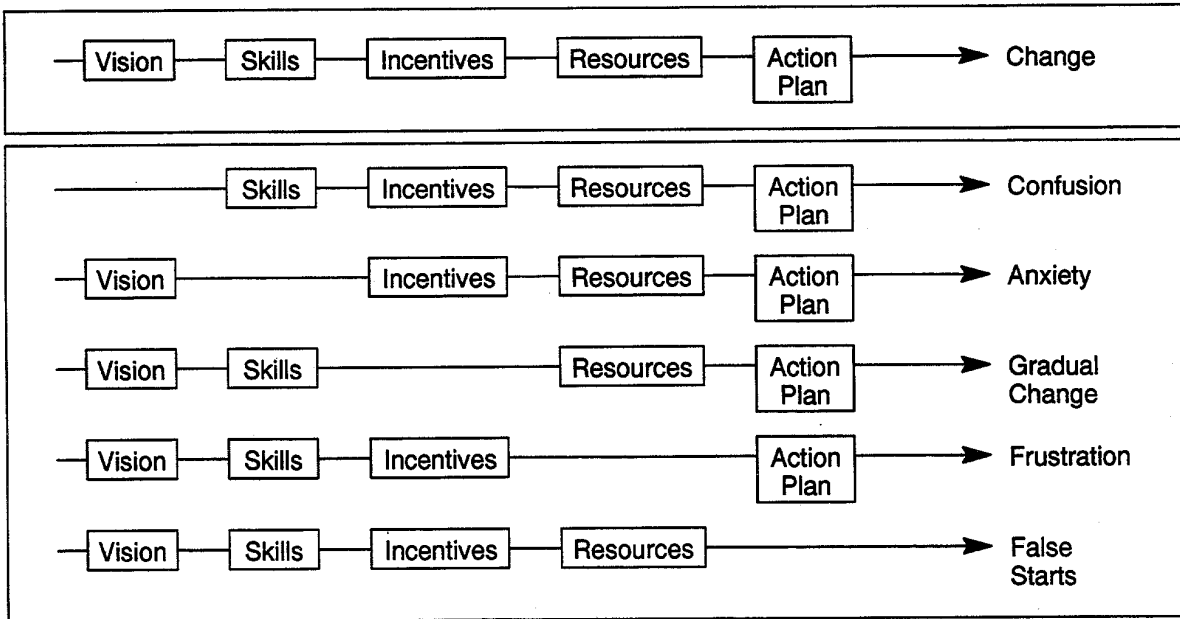


Figure 6 The Requirements for Organizational Change

5 THE SOFTWARE PROCESS, MATURITY, AND PRODUCT QUALITY

There are many, many ways to develop software—perhaps as many as there are people doing it. Certainly, there is not one right way. But when an organization develops software, multiple people are involved—managers and customers as well as the software engineers. And an agreed-upon way of getting the job done is important.

An “agreed-upon way of doing software” is what we mean by a software process. Having an agreed-upon way means your managers can fulfill their responsibilities and make commitments based on an understanding of what the status of work really is, and who is doing what at any time. An agreement that involves customers also means that they understand your process, a prerequisite to having confidence in it and in the products you produce using the process.

Having a software process raises a second set of issues: How good is your process? How skillful and experienced are your people in executing the process? How effective is the technology you are using? In other words, how mature is your organization’s overall software capability?

Answers to these questions are just as important as having an agreed-upon way to do software because if your software capability is low, the reliability and overall quality of your software products are more than likely to show it. Watts Humphrey of the SEI is credited with this assertion:

The quality of a software system is governed by the quality of the process used to develop and maintain it.

The material in this section assumes that your organization has an agreed-upon way of doing software. If you do not, establish one right away. Or if your software process does not have the

buy-in of all your managers and customers, we recommend that you get that done as quickly as possible.

Then, with your software process in mind, think of this section as a gauge to figure out what changes to make to improve the way you do software. You also can think of the gauge as a roadmap for improving your overall capability as an organization and, most important, the software products you produce. Section 6 continues with some information about how to use the gauge and roadmap, and about tailoring use of it to your organization.

5.1 A Model of Maturity Exists for Software

During the past ten years the SEI at Carnegie Mellon University has developed a resource for improving your organization's software capabilities. This section introduces the Capability Maturity Model for Software, Version 1.1, developed by the SEI [25]. The information has been tailored to your needs and interests as a supplier to semiconductor manufacturers.

The SEI's Maturity Model provides guidance on how to gain control of processes for developing and maintaining software and on how to evolve toward a culture of software engineering and management excellence. The Model helps organizations select improvement strategies and identify the few issues most critical to software quality and process improvement. By this focusing and then working aggressively to implement specific improvements, an organization steadily improves its organization-wide software process and makes continuous and lasting gains in its software capability.

The Model characterizes software maturity with 18 key process areas (KPAs). A KPA identifies a critical part of the software process—an area that is a prime candidate for improvement. The Model organizes the KPAs by maturity levels to create a roadmap that leads from lower to higher maturity. Figure 7 shows the maturity level structure of the Model; Figure 8 names the 18 KPAs within that structure. Section 5.2 covers these subjects in more detail.

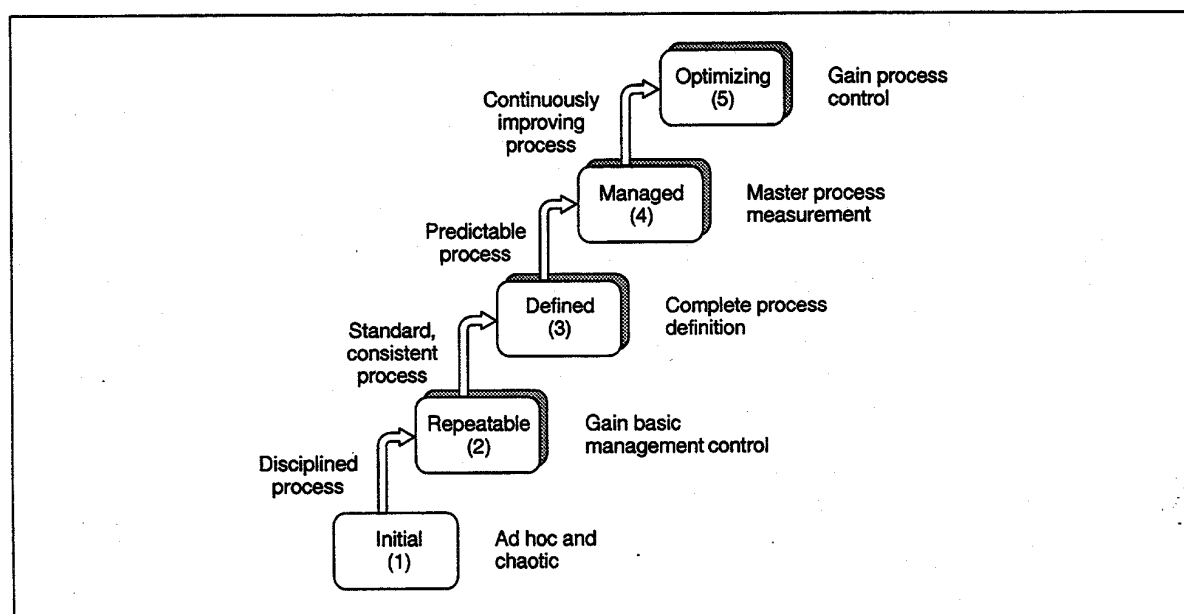


Figure 7 The Maturity Levels of the Capability Maturity Model

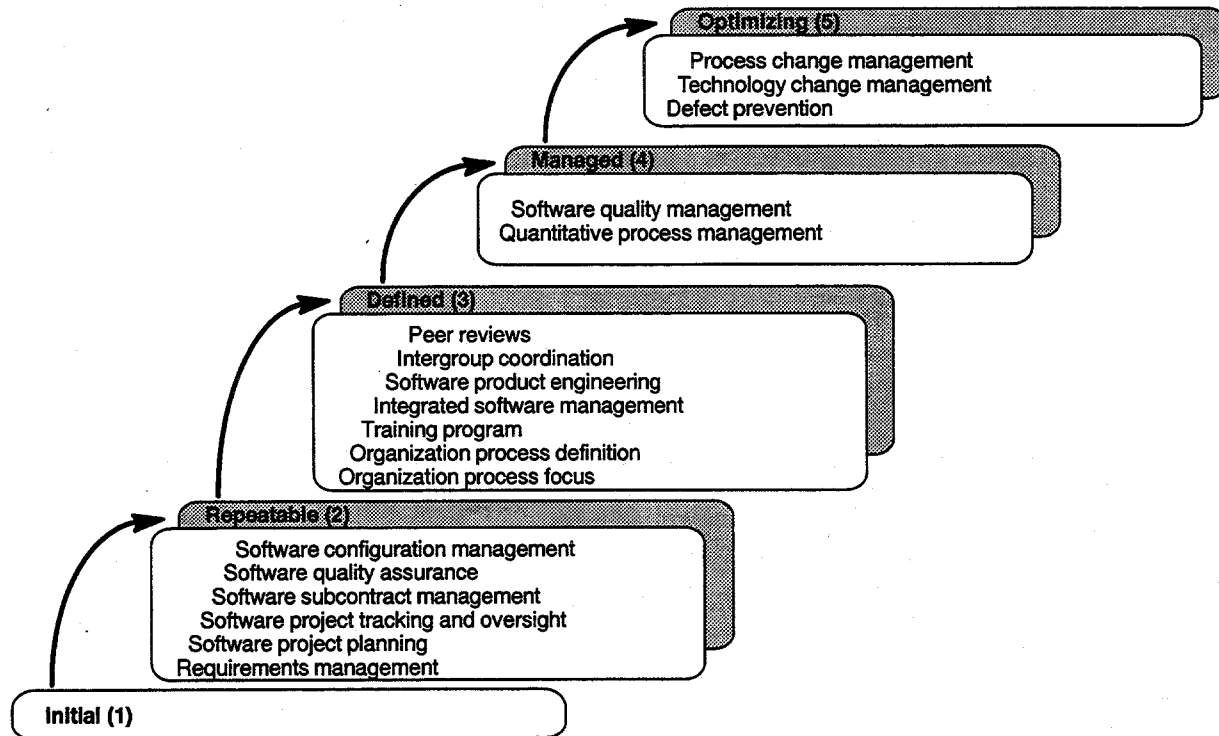


Figure 8 The Key Process Areas by Maturity Level

As accessories to the Maturity Model, the SPI Project offers three additional resources as an approach ramp:

- **Initial Issues**—A list of starting points for work on software improvement. Each item on the list relates to practices in one or more of the Model's key process areas. (See Section 5.2.3 for more information.)
- **Organizational Factors**—A list of dynamics within an organization that affect a software process and its agents and thus are important to consider from the outset of an improvement initiative. (See Section 5.4 for more information.)
- **Readiness Criteria**—Criteria for determining an organization's overall readiness to reap the benefits of software process improvement. (See Section 5.5 for more information.)

Figure 9 depicts the relationship of the Model to these three accessories.

5.2 The Model's Key Process Areas Indicate Maturity

The maturity of an organization's software capability is a function of three variables:

How complete its software process is and how institutionalized

How comprehensive the skills and experience of its staff are

How effective the technologies it uses are

The 18 KPAs of the Maturity Model characterize “completeness” of the software process. Choice of the 18 KPAs was based on experiences in many organizations, not on a proof of correctness that they are THE ultimate indicators of completeness.

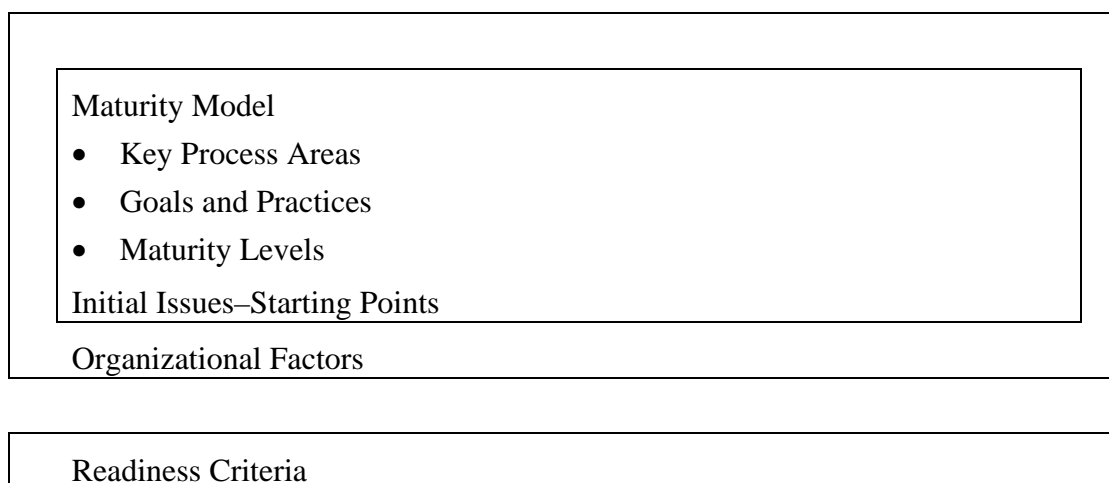


Figure 9 The Capability Maturity Model and Its Accessories

For each KPA, a set of detailed goals and characteristic practices for reaching the goals models the other attributes of maturity. These goals and practices express characteristics that relate to

- Scope and effectiveness of software practices, staff skills, and technologies used
- Development and institutionalization of a process focus, at both project and organization levels
- Quantitative management, applied to the process and products produced using the process
- Change management of the process as well as the technologies used within the process

These characteristics are the essence of the Maturity Model.

Before continuing, it is important to note that the Maturity Model is not a model of a software process. That is to say, the Model does not connect KPAs or their practices into a process. This is deliberate. Each organization must define its own software process, one that best serves its purpose. So the Maturity Model doesn’t need to provide a process, not even a generic one. Your process, however mature, may correspond to all or only a few of the KPAs. But it is up to you to define what your software process is.

A few of the Model’s KPAs may not apply in some organizations (e.g., Software Subcontract Management is not applicable in organizations that do not subcontract any of their software work). The KPAs that are applicable are prescriptive of a mature process. Individual practices within the KPAs are suggestions; i.e., you must determine how the “spirit” of the practices applies to your situation—what best practices will achieve each KPA’s goals.

5.2.1 The Structure of the Indicators

We have already discussed the maturity level structure of the model. A second formally defined structure organizes practices within the KPAs, and the SEI uses a third type of structure that is informal but useful.

The Model's maturity levels organize the KPAs into a roadmap for software process improvement. Figure 7 depicted this five-level structure; Table 1 briefly characterizes software activities at each of the levels [26] (as experienced by one user of the Maturity Model). Figure 8 distributes the KPAs over maturity levels 2 through 5.

Table 1 Descriptions of Maturity Levels in the Maturity Model

Maturity Level	Characterization of Operations
1 <i>Initial or Anarchy</i>	Programmers do what they individually think best, hoping their work comes together at the end of the project. Cost, schedule, quality are unpredictable and out of control. There are no formal plans or programming practices. Change control is informal, and tools are not integrated into the development process. Senior management doesn't understand software problems and issues.
2 <i>Repeatable or Folklore</i>	Programmers have enough experience developing certain kinds of systems that they believe they have devised an effective software development process. they have learned to make plans and meet estimates; corporate mythology informally embodies their accumulated wisdom. Their strength is in doing similar projects, but they falter when faced with different projects, new tools or methods, or organizational change. Knowledge is contained in the minds of individuals.
3 <i>Defined or Standards</i>	Corporate mythology is written down in a set of standards and procedures that define the process. Even if the process is followed consistently, there is little or no evidence to say how well it works. So programmers debate a lot about the value of software process, and about what process and product measurements to use, if any.
4 <i>Managed or Measured</i>	The standard process is measured, and hard data that assesses its effectiveness are collected. these data eliminate arguments that characterize life at Level 3. Hard data are available to judge the merits of competing processes objectively.
5 <i>Optimizing</i>	In lower levels of process maturity, an organization focuses on repeatability and measurement primarily to improve product quality. It might measure the number of defects per 1000 lines of code to know how good its code is. Now the focus is on repeatability and measurement so it can improve its development process. Defects/lines of code indicate how well its process works. It can vary its process, measure the results of each variation, and determine which is a process improvement. It has tools in place that automatically collect the data needed to analyze the process and improve it.

The maturity levels establish an approximate order for dealing with improvement issues as an organization moves from a low to a high level of maturity. It means an organization should plan to master the KPAs at lower maturity levels before mastering those on higher levels. This does not mean, however, that you should delay all work with KPAs on Level 3 until the KPAs on Level 2 are mastered.

In practice, organizations do not literally progress through the KPAs level by level, so it is imprecise to apply maturity level labels to organizations. The maturity level labels apply only to the model itself, even though people sometimes refer to the maturity level of an organization. Such references are frequently misleading and detract from the emphasis in these

guidelines—that the measure of organizational improvement must be quality in products produced.

The second structure of the Maturity Model is within each key process area. The practices in each area are categorized under the five headings shown in Figure 10. This structuring of the maturity indicators classifies how far an organization has progressed with institutionalizing a particular KPA.

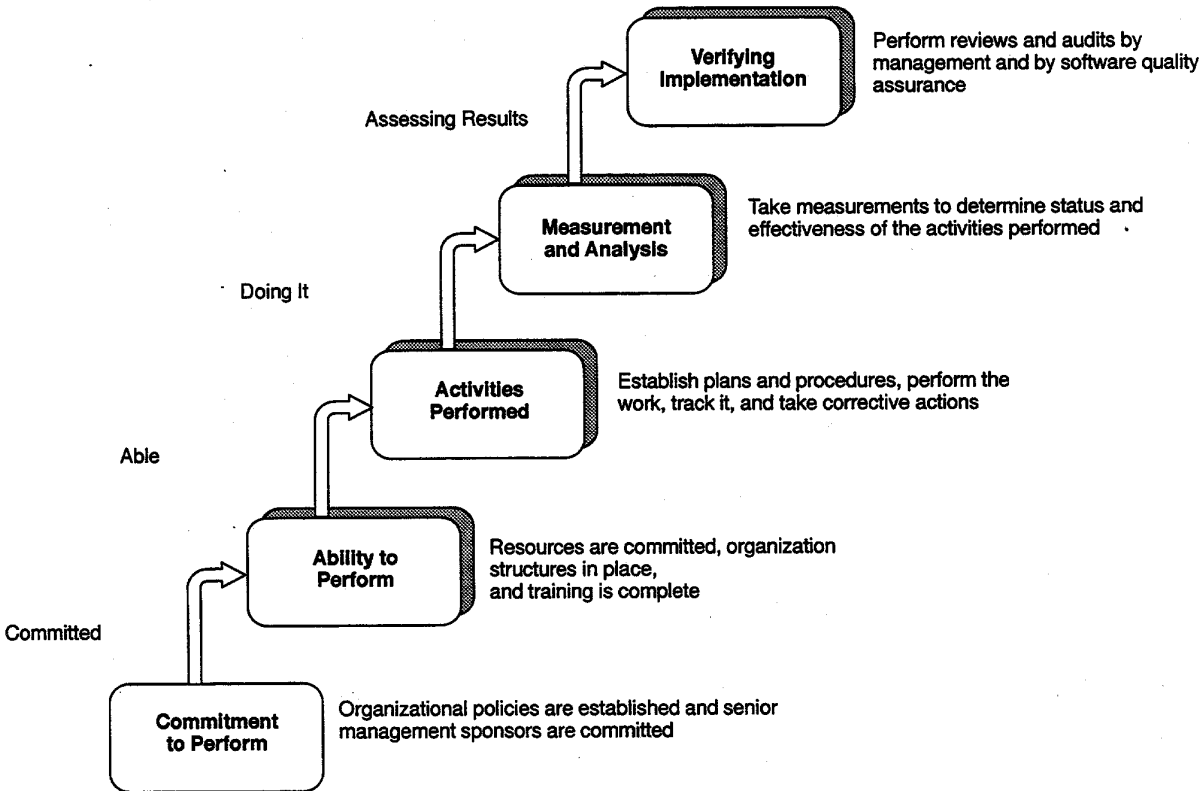


Figure 10 The Steps to Institutionalizing the Practices of a Key Process Area

It is important to notice what this structure implies about institutionalizing improvements:

Commitment to perform—implies communicating the commitment to all participants. An organization at all its levels must stay committed as improvement is made step by step. And the prerequisite to being committed is to be involved.

- Ability to perform—implies that people skills and experience are a prerequisite of success. If people in the organization do not have the necessary skills and experience, they must either be trained or people that do have them must be found.
- Activities performed—implies that the essence of improvement is in making changes. In some cases, changes will be pilot tested in a specific project and then proliferated throughout the organization. In others, the changes will be directly implemented throughout the organization.

- Measurement and analysis—implies the need to understand factually the impact of changes as they are made. Measurements determine whether a change is in fact an improvement. Management uses these factual data to justify the change and to motivate the participants in the process.
- Verification of implementation—implies that what people actually do is what really matters. Policies, plans, and procedures are of little value unless people execute properly. Verification involves steps that ensure activities are performed in compliance with the established process.

the third type of structure is informal and used to show how certain clusters of KPAs relate to each other within categories that have meaning for an organization. Figure 11 is an instance of this structure. The key process areas are grouped into three categories that correspond to processes labeled Management, Organizational, and Engineering:

The Management category contains project management activities as they evolve from planning and tracking at Level 2, to managing according to a defined software process at Level 3, to quantitative management at Level 4, to innovative management in a constantly changing environment at Level 5.

The Organization category contains the cross-project responsibilities as the organization matures, beginning with a focus on process issues at Level 3, continuing to a quantitative understanding of the process at Level 4, and culminating with the management of change in an environment of continuous process improvement at Level 5.

The Engineering process category contains the technical activities, such as requirements analysis, design, code, and test, which are performed at all levels, but that evolve toward an engineering discipline at Level 3, statistical process control at Level 4, and continuous measured improvements at Level 5.

Note that a Levels 4 and 5 there are key process areas that span two categories.

Process Categories			
Levels	Management <i>Software project planning, management, etc.</i>	Organizational <i>Senior management review, etc.</i>	Engineering <i>Requirements analysis, design, code, test, etc.</i>
5 Optimizing		Technology Change Management	
		Process Change Management	Defect Prevention
4 Managed	Quantitative Process Management		Software Quality Management
3 Defined	Integrated Software Management Intergroup Coordination	Organization Process Focus Organization Process Definition Training Program	Software Product Engineering Peer Reviews
2 Repeatable	Requirements Management Software Project Planning Software Project Tracking & Oversight Software Subcontract Management Software Quality Assurance Software Configuration Management		
1 Initial	Ad Hoc Process		

Figure 11 The Key Process Areas Assigned to Process Categories

A second instance of this structure is in Figure 12. The three categories and the key process areas assigned are as follows:

Producing the Software Product—the four KPAs assigned cover the life cycle of product development, from gathering requirements through product delivery and maintenance.

Working for Product Quality and Process Maturity—the six KPAs here involve the leadership needed to develop a process focus for software, to evaluate the organization's software capability, and to increase the quality of its software products through a planned improvement initiative.

Managing the Software Effort—the other eight KPAs cover all product management activities that relate to software, from projects to the executive level.

This way of viewing the Maturity Model is more meaningful when the KPAs are well understood. Then it becomes a way of mapping the Model into your particular organization. We therefore encourage you to define categories that cluster the key process areas into a roadmap for improvement which is meaningful for your organization.

Levels	Process Categories	Producing the Software Product	Working for Product Quality and Process Maturity	Managing the Software Effort
5- Optimizing:			Defect Prevention	Technology Change Management Process Change Management
4-Managed:				Quantitative Process Management SW Quality Management
3-Defined:		Peer Reviews SW Product Engineering	Integrated SW Mgmt. Training Program Org. Process Def. Org. Process Focus	Intergroup Coordination
2-Repeatable:		Configuration Management Requirements Management	SW Quality Assurance	SW Subcontract Mgmt. SW Project Track./Oversight SW Project Planning
1-Initial:	Initial Issues-Starting Points			

Figure 12 Another Assignment of Key Process Areas to Categories

5.2.2 The Content of the Indicators

The Model's key process areas are a gauge for an organization's maturity in software. This section states the purpose of each area. The statements are ordered by maturity level within the three staff categories used in Figure 11: software engineering, quality engineering, and managing the software effort. Complete documentation for the key process areas fills a large notebook and is available on request [27].

Software Engineering. Software Engineering is the core of the software process, but certainly not the full extent of it. The Maturity Model devotes one key process area to software engineering exclusively, and three additional areas to closely related activities. The four areas and their maturity levels are

Key Process Area	Level
Software Configuration Management	2
Requirements Management	2
Peer Reviews	3
Software Product Engineering	3

The four areas involve customers in the software process (from requirements to product releases) and address defect detection and removal in a manner that leads to defect prevention (peer reviews).

Therefore, starting with your software process, the Model says to improve it until the four key process areas listed above are institutionalized. The participation and support of management and quality engineers are needed to make this transition a success.

The purposes of the key process areas in this cluster are as follows:

Software Configuration Management—to establish and maintain the integrity of a project's products throughout the product's life cycle. This function is intimately related to an organization's established process for software product engineering.

Requirements Management—to establish (between a customer and a project team) a common understanding of the customer's requirements. These requirements, which the project is to address, and the agreement about them is the basis for planning and managing a project. The ongoing relationship between project and customer is supported by an effective change control process.

Peer Reviews—to remove defects from work products early and efficiently. An important corollary effect is to develop a better understanding of the work products and preventable defects. The peer review is an important and effective method that can be implemented through inspections, structured walkthroughs, or other collaborative technical reviews.

Software Product Engineering—to consistently perform a well defined process that integrates all of the technical activities—requirements analysis, design, code, test, and others—that are required to produce correct, consistent software products effectively and efficiently.

Quality Engineering. The Maturity Model recognizes a broad spectrum of activities related to building quality into software as it is developed, and then retaining quality as the software is supported throughout its life cycle. The cluster of six key process areas devoted to these activities, with their maturity levels, are

Key Process Area	Level
Software Quality Assurance	2
Organization Process Focus	3
Organization Process Definition	3
Training Program	3
Integrated Software Management	3

The core of the cluster is the area titled Organization Process Focus. This function, once established, prepares an organization to implement improvements corresponding to the other parts of the Maturity Model applicable to the organization. Placing high priority on this area is the key to fast-track improvement of software.

Larger companies do this by establishing and staffing a function called the Software Engineering Process Group (SEPG). Smaller organizations may be less formal and commit staff part time instead of full time to this leadership role. In all cases, though, the leadership is required for an improvement initiative to succeed.

Your company must determine for itself which aspects of these six areas to implement and to what extent. One important lesson learned is that a company cannot establish a software improvement initiative and institutionalize improvements without dedicating staff resources to that objective.

Starting with your software process, the Model says to improve it until all six key process areas listed above are institutionalized. The participation and support of managers and involvement by software engineers are needed for this transition to succeed.

The purposes of the key process areas in this cluster are as follows:

- Software Quality Assurance—to provide management with appropriate visibility into the process a project is using and the products it is building. (Some organizations, especially smaller ones, also include integration and regression testing as responsibilities of Software Quality Assurance. SEMATECH supports that approach even though the SEI's CMM does not include testing practices in its definition of this KPA.) This area of activity is intimately related to software product engineering and to project planning and management.
- Organization Process Focus—to establish responsibility for activities that improve an organization's overall software process capability. An important result of this activity is a set of process assets that is used by software projects.
- Organization Process Definition—to develop and maintain a usable set of software process assets that improve process performance across projects and provide a basis for defining meaningful data for quantitative process management. These assets are a foundation that can be institutionalized through mechanisms such as training.
- Training Program—to develop the skills and knowledge of individuals so they can be effective and efficient. Training is an organizational responsibility, but projects should identify their needed skills and provide training when their needs are unique.
- Integrated Software Management—to integrate software engineering and management activities into a coherent, defined process that is tailored form an organization's standard software process and related process assets. Tailoring is based on the business environment and technical needs of a project.

- Defect Prevention—to prevent defects from recurring by analyzing them, identifying their causes, and changing the defined process. Any process changes of general value are transferred to other software projects.

Managing the Software Effort. Management of the software process begins with project planning and tracking (including subcontracting) and extends to two issues critical to the success of any improvement initiative: management by fact and change management. The Maturity Model includes a cluster of eight key process areas that relate primarily to management activities. Those eight areas and the maturity level of each are

Key Process Area	Level
Software Project Planning	2
Software Project Tracking/Oversight	2
Software Subcontract Management	2
Intergroup coordination	3
Software Quality Management	4
Quantitative Process Management	4
Process Change Management	5
Technology Change Management	5

This cluster does not have one core area as in the other two clusters. In software-only companies, the first two are important; for equipment suppliers, Intergroup Coordination is critical; and in all cases, the two areas on Level 4—management by fact or data—are the only sound basis for improvements in products and process, respectively.

Starting with your software process, the Model says to improve the way you manage it until all eight KPAs listed above are institutionalized. The cooperation and support of your quality and software engineers are needed for this transition to succeed.

The purposes of the key process areas in this cluster are as follows:

- Software Project Planning—to establish reasonable plans for software engineering and other project activities. These plans are the foundation for project management. In fact, without realistic plans, effective project management is not possible.
- Software Project Tracking and Oversight—to maintain adequate visibility of actual progress so that management can take effective actions when a project's performance deviates significantly from the plans.
- Software Subcontract Management—to select qualified software subcontractors and to manage them effectively.
- Intergroup Coordination—to establish a way for a software engineering group to participate actively with other engineering groups so that a project team can better satisfy the customer's needs.
- Software Quality Management—to develop a quantitative understanding of the quality of the project's products and to achieve specific quality goals.

- **Quantitative Process Management**—to control a project’s performance quantitatively. A project’s performance comprises the actual results achieved from following a software process. The focus is on identifying special causes of variation within a measurably stable process and correcting, as appropriate, the circumstances that created them.
- **Process Change Management**—to continually improve an organizations’ processes with the intent of improving quality, increasing productivity, and decreasing development time. This activity takes incremental and innovative changes that are validated improvements and makes them available to the entire organization.
- **Technology Change Management**—to identify beneficial new technologies (e.g., tools, methods, and processes) and transfer them into an organization in an orderly manner. The focus here is on efficient innovation in an ever-changing world.

5.2.3 Some Starting Points for Process Improvement

The Maturity Model assumes the existence of several basic software practices. Organizations that have implemented these basics—and many have not—find the first set of KPAs, those on maturity level two, particularly challenging.

The Initial Issues identify these basic software practices. An organization can begin to work on software improvement by picking one or more of these to implement. With all of these practices in place, an organization is well positioned to deal with the KPAs on Level 2 and beyond.

Table 2 lists the Initial Issues. The Maturity Model has KPAs assigned to maturity levels two through five, so we regard these as Level 1 issues. Figure 11 shows this.

Table 2 Initial Issues for Software Improvement

Software version identification and release control
Software change control
Software testing plans and results
Defect tracking
Metrics for and measures of each software effort
Description of the software process
Requirements allocated to software

5.3 Metrics Tie Process Change to Product Quality

Section 4.4 points out that the Maturity Model does not make an automatic concoction between process change and product quality. This connection must be defined and implemented by the leaders and managers of the software improvement initiative. Definitions and priorities for software quality and defects, together with metrics for measuring both, are the basis for the connection. From this base, priority must be given to changes that result in product improvements.

As illustrated in Figure 10, each key process area includes basic measurements and some practices related to determining the status of implementing that particular key process area. Reviewing these metrics, along with the other practices in the KPA, reveals the relationship among each of the 18 areas and product improvement.

5.4 Organizational Factors Impact Improvement Initiatives

The participants in an organization's software process are software engineers, managers, and quality engineers. These people are subject to influences that originate in other parts of the company outside the software process. Such influences can determine the outcome of an improvement initiative. You can think of these dynamics as aspects of your company culture.

Table 3 lists some important organizational factors. These should be assessed early in an improvement initiative. If your company can show that the dynamics related to each affirm and support the software component of your business, you can be sure the work to improve software will benefit.

Table 3 Organizational Factors for Software Improvement

Policy statements that relate software, quality, improvement systems, and goals to company business plans, strategies, and goals.
Communication channels between your company and its software customers, including the buyers, users, and management.
Relation between Total Quality Management for the organization as a whole and software improvement.
Pay, incentives, and career paths for software and quality engineers, compared to other engineering disciplines.
Teamwork and communication among the engineering disciplines in your company and between engineering and others.

On the other hand, factors that have a negative impact make the task of software improvement more difficult, or even put its success at risk. You, therefore, are advised to review these situations with the appropriate managers, working to replace negative pressures with positive reinforcement, along the lines suggested below:

- Policy statements that relate software to company business objectives
- Communication channels between your company and its software customers
- Clear relation between Total Quality Management and software improvement
- Pay, incentives and career paths for software and quality engineers
- Teamwork and communication among various work teams in your company

By aligning the dynamics of each issue so that it gives constructive support, you are maximizing your chance of successfully improving software.

5.5 An Organization Must be Ready for Software Improvement

During recent years, many organizations have tried to improve software quality and staff productivity. The attempts that failed reveal two fundamental problems [29]:

Many organizations have incomplete and immature software processes.

Management of the software process is often dysfunctional.

Overcoming such problems requires an entire organization to be involved; an effort by the software staff alone cannot succeed.

We offer the Readiness Criteria listed in Table 4 to help distinguish organizations that are prepared to address the full range of issues involved in software improvement—or are already addressing them—from organizations that are not.

Table 4 Readiness Criteria for Software Improvement

Software is important to our customers and to our own business success.
We want to improve our organization's software capability as well as the quality of the software we produce, and we have some ideas about how to go about it.
Communicating with our customers about software is important and we want to improve our dialogue with them.
Stated requirements for each software project are an important aspect of software improvement.
To be effective, management must be involved in software activities, including planning, progress reporting, and problem solving. Management must know how resources are used and how decisions and commitments throughout the company impact work on software.
To improve software requires specific expertise and an appropriate level of effort that is dedicated to the task.
Improving software, like anything else, requires knowing current levels of achievement, having goals for improvement and plans for achieving the goals, and measuring progress toward the goals.

If your company's response to each criterion is positive and substantive, you are ready to seriously pursue software improvement. Even if you cannot respond positively in some cases, but your company—management and technical staff—is open to the type of change indicated, that is equivalent to a positive response.

If these Readiness Criteria cannot be met, SEMATECH believes it is not cost-effective to commit its resources or those of the supplier organization to software improvement. You are advised to work on changing attitudes in the company before beginning an improvement initiative.

Passing this readiness test indicates that your company is well positioned to launch and succeed with a software improvement initiative.

6 INCREASING PRODUCT QUALITY AND PROCESS MATURITY

From reading Section 5, can you visualize how to use the Maturity Model? Where would you start? Who would do what? How direct would you be about using the Model? This direct?

To increase our maturity, we'll simply select practices from the Model and adjust them as necessary to fit our situation.

In a sense you do that, but it's not all you must do. Improving a software process involves changing a company's operations—even its culture. And no company changes to that degree until its people have a reason or incentive to change. In addition to an incentive, you also need the following:

- Committed leaders throughout the organization
- Some experience and a well made plan
- The required resources and people skills
- Dogged persistence over time

This section describes an improvement framework for you to use. It uses the Maturity Model to improve the quality of software products by

- Establishing an agreed-upon process for its software activity or affirming the process already in place
- Identifying the strengths and weaknesses of its current process and products and making improvements to both
- Extending its software process to include continuous process improvement and other aspects of maturity
- Creating an organization infrastructure that supports the work of continuously improving its software process
- Developing and, if necessary, acquiring the various staff skills and experiences that software improvement requires
- Ensuring that the changes made to its software process and the technologies used also improve its software products

Through such extensions, development, and creations, you and your organization can achieve its goal—increases in overall software capability and improvements to software products.

It is important to note that once a continuous improvement process is in place, it can be used to improve the software process to the full extent of the Maturity Model, if that is appropriate.

Already several SEMI/SEMATECH member companies have applied the Model, learning in the process that it takes more than desire and intuition to get results. Experience, insight, and commitment are essential. Some external prodding helps. The Maturity Model is a foundation, but it is up to you to build on it.

6.1 Success Is in Your Hands

Some organizations launching improvement initiatives succumb to the risks and abandon the effort without positive results [28]. Our aim is to help members of SEMI/SEMATECH manage the risks attendant to software improvement.

We believe the outline of success involves the following:

- Attend to the obvious and basic problems first
- Define what you mean by software quality and defects
- Set realistic goals for quality improvement
- Define metrics that measure your progress toward the goals
- Lead as well as manage your way to the goals

That means, be practical and direct, visionary and creative, logical and well organized—mix and service in generous portions.

The Improvement Method can help, but your challenge in this kind of initiative is to combine strong leadership with strong management. Let's briefly consider how leading and managing are different and complementary [29].

Management is about coping with complexity. There is little argument that software improvement involves complexity. Without good management, initiatives like this become chaotic to a degree that leads to failure. Only with good management can you achieve results that matter to your customers and to your own business.

By contrast, leadership is about coping with change. Software improvement is a conscious and deliberate effort to change your company. Following are examples of change you might encounter:

- Viewing software activity as a process
- Expanding the skills of software practitioners beyond programming
- Developing and using an improvement system
- Setting goals, defining metrics, and measuring progress
- Elevating the role of software within the organization
- Making management decisions based on facts
- Being a learning organization that institutionalizes change
- Putting a high value on diversity

Change requires leadership from people in positions throughout the organization: executive and middle managers, project managers, and software and quality engineers. That means people and their development are critical—careful selection of staff and delegation of responsibility at the outset, ample training and coaching and consistent nurturing and encouragement along the way.

Sometimes undertakings of this kind are undermanaged—but they are almost always underled. The Improvement Method cannot take the place of proactive leadership. There can be no substitute; you must know when and how to manage and when and how to lead the charge that changes.

6.2 A Method for Using the Maturity Model Exists

The SEI at Carnegie Mellon University has developed a framework for software improvement that is compatible with the Capability Maturity Model [30]. The SEMATECH SPI Project has related to that framework experiences gained and lessons learned from two years of work with suppliers in the semiconductor industry. The combination is a generic Improvement Method for suppliers that can structure and inform the plan you develop for your company.

This section introduces the Improvement Method. You can think of the Method's principle thrust as establishing capabilities related to one specific key process area of the Maturity Model—the one on Level 3 called Organization Process Focus. This KPA's purpose and scope are as follows:

- Purpose—To establish responsibility for activities that improve the organization's software process and its overall software capability. (Notice that improving the product is not mentioned; it's your responsibility to see to that.)

- **Scope**—To develop and maintain and understanding of the software process, both the organization-wide process and the process at the project level; also to assess, develop, maintain, and improve these software processes.

What this says is that the Improvement method helps you develop, institutionalize, and use capabilities related to the Organization Process Focus area. You then use these capabilities to develop, institutionalize, and use capabilities that relate to each of the other key process areas that is applicable to your organization.

To do this, each company must develop its own improvement plan because

- There is not a one-size-fits-all plan for improving software
- No outsider has a ready-made plan that will work for you
- Silver bullets cannot do away with your software problems

But using the Method as a base will ensure that your organization's plan

Has balance—that you improve your agreed-upon process, the skills of your people, and the uses you make of technology.

Uses the experience of others—your initiative benefits from the experiences and lessons learned by others.

Has staying power—your initiative uses a roadmap that charts beneficial results all the way to the objective.

6.3 A Cycle of Five Phases Provides the Framework

The IDEAL framework charts a simple, common-sense approach to improvement. It's a cycle of five phases, as shown in Figure 13. The first letter in the names of the five phases form the acronym IDEAL.

An improvement initiative makes progress by repeating the IDEAL cycle many times. Each repetition includes the Initiating Phase because the stimulus for improvement and everyone's commitment to it must be revisited periodically—just as each of the other four phases must be repeated.

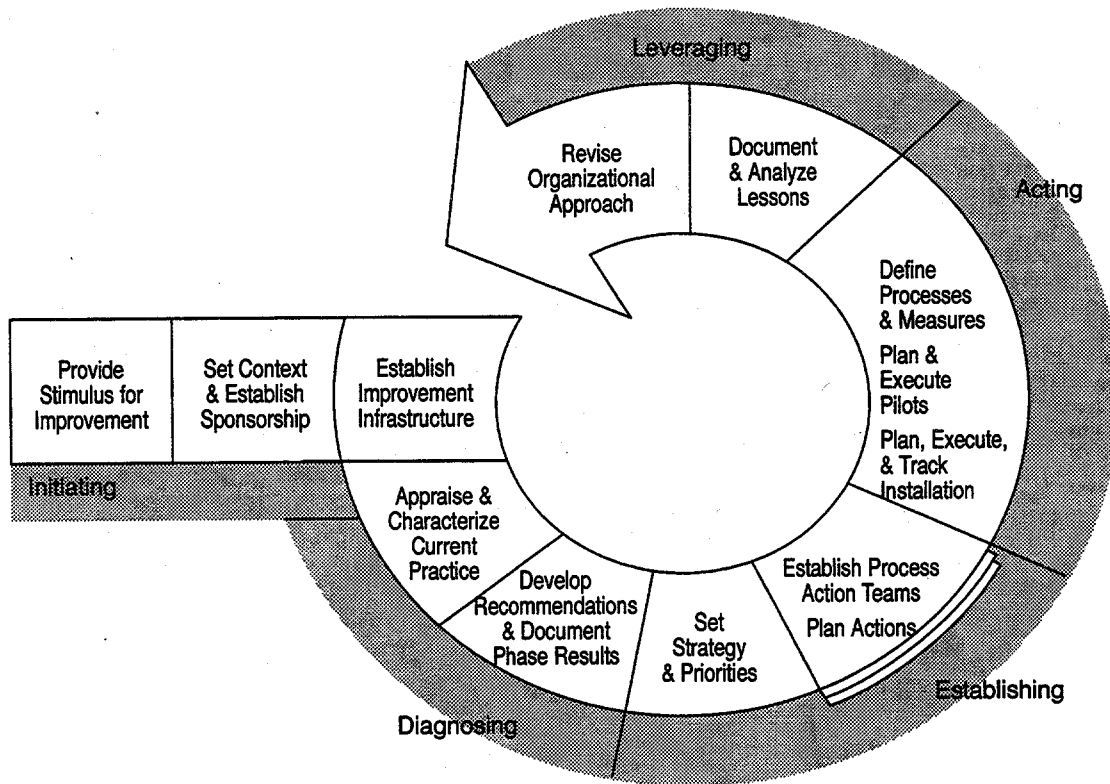


Figure 13 The IDEAL Approach to Improving a Software Process

From our work with suppliers in the semiconductor industry we add the following to the IDEAL Framework:

Representative accomplishments and milestones, from start-up to product improvement (see Section 6.3.1)

A practical infrastructure that supports continuous improvement (see Section 6.3.2)

The SEI has filled in the IDEAL framework with a generic set of improvement activities (see Section 6.3.3). Selecting from these for your customized plan will ensure long-term balance and staying power.

6.3.1 Accomplishments and Milestones Around the Cycle

The phases in the IDEAL cycle are appropriately labeled with verbs—Initiating, etc.—that suggest the activities in the phases. In Figure 13, the boxes around the cycle contribute two or three additional phrases for each phase that further define the activity and identify some results.

This section gives examples of accomplishment resulting from each phase, some of which represent important milestones. These examples are drawn from work with suppliers in the semiconductor industry. Figure 14 illustrates the connection of these accomplishment categories to the phases of the IDEAL framework.

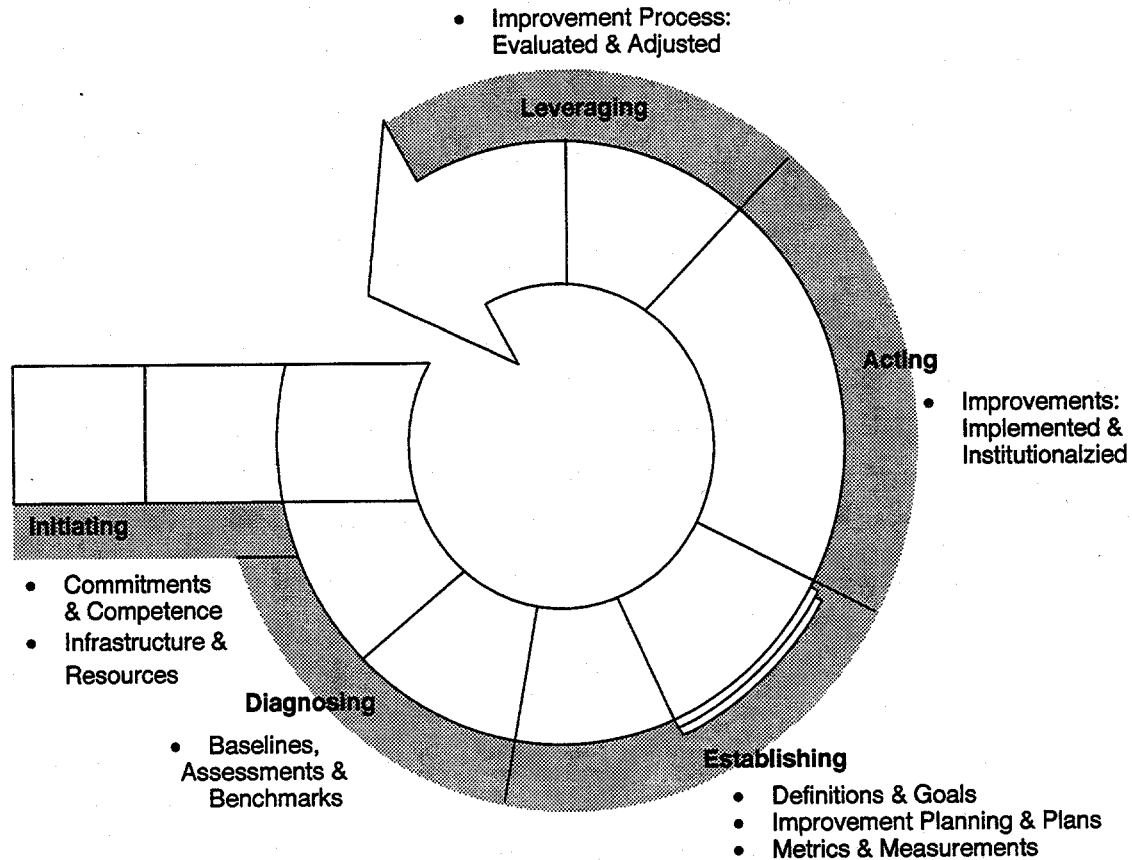


Figure 14 The Major Accomplishments and Milestones in Improving a Software Process

Initiating Phase. Accomplishments of the following types relate to this phase:

- Commitments by managers and staff throughout the organization to improving software.
- Competence of managers and staff in making the changes that result in improving software.
- Infrastructure that drives and supports change, from charters for change agents to measures and assets for change-making.
- Resources allocated to the work of software improvement, in particular staff time.

Diagnosing Phase. Accomplishments of the following types relate to this phase:

- Baselines of the organization's current levels of performance, maturity, and quality
- Assessments of the organization's current software process, people skills, and uses of technology
- Benchmarks of what other comparable organizations have achieved in levels of performance, maturity, and quality

Establishing Phase. Accomplishments of the following types relate to this phase:

- Definitions of what the organization means by quality, defects, failures, etc.
- Goals for the organization related to increasing quality, decreasing defects, etc.
- An Improvement Planning Process that is integrated into the organization's overall software process and institutionalized
- Plans for improvement that identify tasks and paybacks and that provide support for longer term change
- Metrics defined to measure progress in KPAs, and link process change to product improvement and other business objectives
- Measurements of products and processes preserved and used to justify/motivate work on improvements and to improve the organization's ability to plan and estimate

Acting Phase. Accomplishments of the following types relate to this phase:

- Changes to the software process, either at the project or organization level, that are validated to be improvements
- Validated process improvements institutionalized by the organization and made part of the software process
- Improvements in skill and performance levels of managers and software engineers and in the uses made of technology

Leveraging Phase. Accomplishments of the following types relate to this phase:

- Evaluations of the organization's process for improving software that show what has been accomplished and remains to be done and reveals strengths and weaknesses of the approach
- Adjustments to the organization's process for improving software that strengthen it and increase commitments to it

6.3.2 A Supporting Infrastructure

An organization that has institutionalized its approach to software improvement has an infrastructure to support change-making that includes

- Organizational functions that are chartered and staffed
- Defined metrics and an available history of measurements
- Process assets and artifacts in an easy-to-access form
- Staff capabilities and a training curriculum

It takes time and a lot of work to build this infrastructure, but its existence is the difference between an improvement initiative that is still coming into its own and one that is well established and productive. This section briefly discusses each of the four aspects listed.

Organizational Infrastructure

Many companies charter an organizational infrastructure that has the three parts, as shown in Figure 15 [31]. The first part is the management steering group (MSG). Its members represent the highest level of management in the organization. This group guides the improvement

initiative by establishing its goals and objectives and by setting priorities. The MSG also allocates staff and other resources to the initiative.

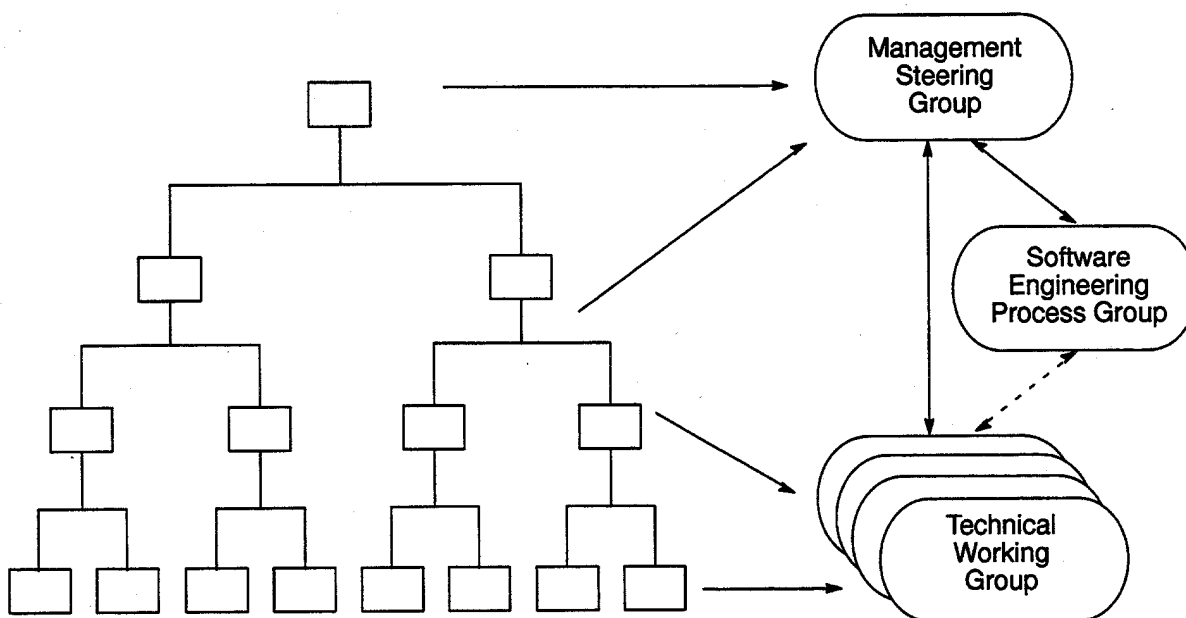


Figure 15 Organizational Functions That Support Continuous Improvement

Reporting to the MSG is the Software Engineering Process Group (SEPG). The leader of the SEPG also participates as a nonvoting member of the MSG, sometimes serving as its facilitator. Membership of the SEPG is drawn from the software practitioners throughout the organization. Depending on the organization's size, SEPG membership can be full or part time. IN all cases, a leader should be designated who is committed to the effort full time (if warranted by the organization's size). If not, the commitment must be at an appropriate level, with half time as a practical minimum.

The SEPG is the focal point for the improvement initiative. It is responsible for and facilitates the activities that relate to all phases of the improvement process (e.g., action planning, process change, technology change, etc.) The SEPG communicates about the initiative to all parts of the organization, serves as a facilitator for all activities, and serves as the continuing point for networking with similar initiatives in other companies. As the program's catalyst, one of the biggest challenges is to maintain the motivation and enthusiasm for improvement across and between all levels of the organization.

Also reporting to the MSG with a dotted-line relationship to the SEPG, are the technical working groups (TWGs). TWGs are the solution developers for the improvement initiative. Each is formed to address a specific area of the software process and is given a charter, resources, and authority to complete its activity. membership in the TWGs is drawn from those areas that would be affected by the improvements made.

The purpose of a TWG is to improve a specific area of the software process. To succeed, the TWG must be given proper guidance by the MSG. Another key to success is that the membership of a TWG be drawn from staff who

- Are knowledgeable about the process being improved
- Work within that part of the software process
- Would be affected by the changes made to improve the process

The leader of a TWG should be the owner of the area being improved.

The organizational infrastructure created must be well defined and integrated, with each component having a specific role to play. Care should be taken that the size and shape of the infrastructure is adequate for the task.

Metrics and Measurements

Throughout these guidelines we have frequently referred to the role and importance of metrics. The success of an initiative to improve software is, in our experience, directly proportional to use quality of its metrics and the effectiveness with which the leaders and managers put them to use.

Few organizations define metrics or have historical data related to their software products and process. The infrastructure required, other than the metrics themselves, is a repository for historical data to which project managers and others throughout the organization have easy access.

The Maturity Model provides some basic guidance on metrics for each key process area. Three KPAs of particular importance are Project Planning, Project Tracking and Oversight, and Inspections. Metrics and measurements related to defect tracking are another important area, as are measurements related to software reliability and other matters of concern to your customers.

The Process Assets

A library of process assets contains generic documents, templates, checklists, spreadsheets, presentations, and other resources that help accomplish specific software improvement tasks. These resources are used by an organization's change agents in tasks throughout a software improvement initiative—to introduce, plan, implement, and support it. Ideally, you would assemble a set of process assets for work related to each key process area that is applicable to your organization.

In some cases, an organization will develop such resources of itself and wisely preserve them for reuse at other times as the work progresses. In other cases, such resources are available from other organization working on software improvement. The important thing is to recognize the importance of such assets and to look for opportunities to collect useful items.

The SPI Project is working on a process asset library for supplier companies to use. As this set of materials becomes available, you will receive information about how to access it.

Staff Capabilities and a Training Curriculum

Software improvement, as any aspect of the software process, has been largely ignored by companies, whether they sell standalone software products or embedded software. The consequence is that few organizations have staff well versed in matters such as the Capability Maturity Model, the definition of a software process, software process assessments, and asset library, and continuous improvement of a software process.

This lack of staff experience and know-how is a challenge for suppliers in the semiconductor industry. From the Initiating Phase on, you must assess the skills and experience levels of your staff and provide the training that is prerequisite to being successful.

That brings us to the fourth aspect of an infrastructure to support software improvement—a training curriculum. The options are similar to those discussed for the process asset library; do it yourself, get it from an external source, or take advantage of courses available from SEMATECH’s SPI Project.

A company that is fully committed to software improvement will itself take on much of the work involved in training its staff. And again, many of the courses needed are available from a variety of commercial providers. SEMATECH is working with its collaborators to provide the following courses:

- SEP Leader Training, including the MSG and TWG
- Training on the Capability Maturity Model
- Planning Software Improvement Projects
- Software Development Life Cycle
- Project Management
- Configuration Management
- Software Quality Assurance
- Software Requirements
- Software Testing and Inspections
- Risk and Change Management for Software

As courses of these types become available, you will receive information about how to take advantage of them.

6.3.3 Completing a Plan for Improvement

This section is a sketch of generic tasks in each of the five phases of the IDEAL cycle [30].

Initiating Phase. In this phase, identify one or more specific reasons your company wants to change. Typically economic, the reasons must relate the improvement initiative to the goals, strategies, and plans for the business. This sets a context and establishes sponsorship. With sponsorship in place, you begin to establish the infrastructure that software improvement requires. This includes adjusting relevant organizational systems to support the improvement initiative. Most important, you build support for the effort throughout the organization.

This phase accomplishes the following:

- Sponsors and direct participants are properly chartered and understand their respective roles and responsibilities.
- A statement (in draft form at least) of the company’s vision and strategy clearly incorporates software improvement as a strategic initiative.
- A planning process is established and a plan specifies the resources, schedules, and activities for software improvement.
- Detailed plans for the next phase are made and approved.

- Organizational systems are realigned to support software improvement, especially the rewards and communication systems.

The benefits of these accomplishments are the following:

- The software improvement effort is aligned with the company's strategy and vision for the future.
- The software improvement effort is based directly on current business needs of the organization.
- Management has taken on the role of actively leading the software improvement effort.

Diagnosing Phase. In this phase, carefully examine your current software process, using a standard gauge such as the Maturity Model (see Section 5). Of first importance is to understand your software activities as a process and to determine a baseline that characterizes its current level of accomplishment. In the course of that, identify its strengths and weaknesses, including how effectively the software process is managed. Also, identify and assess any risks related to improving the process. Based on such insights, set priorities and recommend specific actions.

This phase accomplishes the following:

- A report of assessment findings is filed, including current strengths/weaknesses and your potentials for improvement. These findings are communicated throughout the organization by briefings and in documents, complete with detailed data, rationales, and recommendations.
- A measurements report is filed that establishes a baseline for your software process and products, expressed in terms of metrics adopted for the improvement initiative.
- The risks are profiled for the improvement initiative.
- The lessons learned are reported for the initiative to this point.

The benefits of these accomplishments are as follows:

Your approach is based on proven principles of process change and process management.

- A community-owned model of software process maturity is the yardstick by which you have evaluated your process.
- The yardstick is also an orderly way to prioritize improvement actions.
- You have provided opportunity for buy-in by those who are to make improvements and will use the changed processes.
- You have a baseline from which to measure progress.

Establishing Phase. In this phase, an infrastructure to sustain software improvement is built. The infrastructure includes functions and staff assignments that implement a process for periodically preparing strategic and tactical plans and a continuous cycle for identifying and specifying improvements. Management must be involved during this phase and committed to funding and supporting the work called for in the tactical plans. The functions established must include teams that will implement improvement plans as they are made and funded.

This phase accomplishes the following:

- A strategic plan for software improvement is finalized and integrated with the strategic plan for the company
- Resources are allocated for the acting phase of the cycle or at least for the next year
- Plans for action are completed with priorities assigned
- Staff are assigned to Technical Working Groups
- The lessons learned are reported

The benefits of these accomplishments are as follows:

- Making changes successfully empowers the entire organization
- The organization achieves its strategic and tactical goals
- The staff demonstrates a capacity to meet challenges
- The company enjoys increased competitive and strategic advantage

Leveraging Phase. This phase completes a cycle of improvement. Completing one cycle literally prepares for the next. Documenting and analyzing the lessons learned allow you to update your approach. Before beginning the next cycle, it is important to revisit the needs of the business and customer priorities to see if they are being met or if they have changed. Other preparations include renewing sponsorship and setting goals for the next cycle.

This phase accomplishes the following:

- The plan for software improvement is revised and updated.
- The lessons learned and historical data are in a repository.
- The initiative has credibility because it is “walking the talk” of continuous improvement.
- There is sustained interest in and commitment to the software improvement initiative.

The benefits of these accomplishments are as follows:

- Continued sponsorship is assured.
- Goals are established for the next improvement cycle.
- Organization policies and practices are based on lessons learned.
- Business values are quantified and documented.
- Improved processes are available for the next cycle.
- Insight into the value of software improvement is clearer.

These accomplishments prepare you for the next cycle of this ongoing process.

7 REACHING THE GOAL SET FOR SOFTWARE IMPROVEMENT

The semiconductor manufacturing industry needs software that is highly reliable within a very demanding environment—the semiconductor fab. Even though the software for a fab is produced by many different organizations, it must function as an integrated system, interface to human

operators in a consistent manner, and interact effectively with information systems that are used to plan and manage fab operations.

For you, a supplier, to successfully improve the software used in this arena requires commitment that begins with your senior management. The formula for success continues with experienced leadership for software improvement; if such experience does not exist within your organization, you can buy it (in the form of a consultant) to get started. Over time you will develop the experience you need, but that takes time if you have none at the beginning.

The third success factor is the staff time invested in the improvement initiative. An often quoted rule of thumb is to commit about 3% of your software engineering staff to leading the initiative. Larger companies charter one or more full-time staff, while smaller companies devote one or more people part time.

The fourth and final factor is technology change plans that are tailored to your company. To make changes of the type presumed by these guidelines and to make them effectively requires well defined plans for the changes. For example, the adoption of a new tool or technical method should be supported by a technology change plan [32]. A template for such a plan is shown in Figure 16. The plan is completed by expressing the goal for the change in terms of objectives over an appropriate time period. Objectives should be stated for each category of the plan template.

A plan like this describes the intended end result and creates a logical path to it. The path takes into account all relevant aspects of the particular change and sequences them appropriately. To be useful, a technology change plan must be brief and must be kept current.

	Next Year	Year + 2	Year + 3
Methodology			
Process			
Tools			
Metrics			
Enterprise Costs			

Figure 16 Technology Change Plan Template

There is much more to say about applying the Maturity Model and Improvement Method to the needs of your company, but that is the purpose of Process Assets that support these guidelines. You will develop such assets, and other who are working to improve software are often willing to share what they develop. The SPI Project's work on Process Assets was referenced in the preface of this document.

The attentive reader will have noticed throughout this material the intimate relationship between improving the knowledge and proficiency of people, defining in detail the software process, and providing technology to support the people and the process. As we pointed out earlier, people, process, and technology are the three leverage points for increasing the quality of software products. All three are involved in the changes that you make to improve software quality.

And remember, we in this industry have much to learn from others as we move forward, as we establish the role of software in semiconductor manufacturing.

8 REFERENCES

- [1] The roadmap was developed at a meeting sponsored by the Semiconductor Industry Association (SIA) in November 1992. The resulting roadmap was published during the first quarter of 1993 as a pair of documents:
 - *SIA Semiconductor Technology, An Agenda for American Cooperation*
 - *SIA Semiconductor Technology, Working Group Reports*
- [2] The Software Engineering Institute, in its course for executives entitled “Software: Profit Through Process Improvement,” cites data from several industries that shows the rapid growth in software, and the timing of the transition.
- [3] The subjects in both surveys were the members of SEMI/SEMATECH. SEMATECH sponsored both; the first was conducted by a group at Sandia National Laboratory in Albuquerque and the second by the SPI Project at SEMATECH. Although the second survey had a somewhat different purpose from the first, there is sufficient overlap of information to show that little change had occurred in the area of software quality. The two reports are:
 - M. Blackledge, M. Smith, and E. Tomlin, *Software within SEMI/SEMATECH: 1989 Software Engineering Survey*, Sandia National Laboratory, Albuquerque, NM, SETEC-90-003, February 1990
 - H. Krasner and R. Reed, *Software Process Improvement Issues Survey Findings Report*, SEMATECH, Austin, TX, Technology Transfer #93011489A-TR
- [4] Two manufacturers with established programs for formally evaluating the quality of products from suppliers are TI and Motorola. They use the following documents in these evaluations:
 - Texas Instruments, *Machine Control Software, Current and Future Requirements*, June 1993, Doc. #TCNTLFUTSW
 - Motorola, *Quality System Review, Subsystem 10, Software Quality Assurance Guidelines*, June 1992
- [5] A report from the Competitive Semiconductor Manufacturing Program, a joint effort of the Engineering Systems Research Center, The Center for Research in Management, and The Berkeley Roundtable on the International Economy, all three of which are at the University of California at Berkeley, CA:
 - *The Competitive Semiconductor Manufacturing Survey: First Report on Results of the Main Phase, April 1993*, Report CSM-02
- [6] This projection is based on preliminary estimates made by the SPI Project at SEMATECH. Work is currently being done to increase the rigor and time span of this projection.

- [7] E. Neacy and R. McKiddie, *Measurement and Improvement of Manufacturing Capability (MIMAC) Survey and Interview Results*, SEMATECH, Austin, TX, Technology Transfer #94052374A-XFR
- [8] The interviews that are the basis for this informal report were sponsored by SEMATECH. The author is on the faculty at Carnegie Mellon University in Pittsburgh, PA. Copies of the report are available from the SPI Project at SEMATECH:
– R. Maxion, A Preliminary Report on Software Failures, May 1994.
- [9] SEMATECH has established the Standardized Supplier Quality Assessment (SSQA) initiative in which the 11 member companies are cooperating in assessments of selected supplier companies. The assessments are based on the international standard ISO-9000. Data from these assessments are available to the member companies. Each supplier assessed is encouraged to use the results as stimulus for making improvements throughout the company.
- [10] C. Baudoin and J. Kantor, “Software Engineering for Semiconductor Manufacturing Equipment Suppliers,” *IEEE Transactions on Components, Packaging, and Manufacturing Technology, Part A*, Vol. 7, No. 2, June 1994, pp. 230-243.
- [11] See both [2] and [10].
- [12] J. Herbsleb, A. Carlton, J. Rozum, J. Siegel, and D. Zubrow, *Benefits of CMM-Based Software Process Improvement: Initial Results*; Software Engineering Institute, Pittsburgh, PA CMU/SEI-94-TR-13, August 1994.
- [13] From presentations made at a Senior Executive Conference on Software prepared and presented by Motorola University in 1992.
- [14] S. Dart, *The Past, Present, and Future of Configuration Management*, Software Engineering Institute, CMU/SEI-92-TR-8, DTIC #AD254175, 1992.
- [15] Any publication of the ANSI/IEEE standards.
- [16] D. Freedman and G. Weinberg, *Handbook of Walkthroughs, Inspections, and Technical Reviews*, Third Edition, Dorset House, New York, NY, 1990.
- [17] B. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood, Cliffs, NJ, 1981. T Abdel-Hamid and S. Madnick, *Software Project Dynamics*, Prentice-Hall, Englewood Cliffs, NJ, 1991.
- [18] B. Hetzel, *The Complete Guide to Software Testing*, Q.E.D. Information Sciences, 1984.
- [19] W. Florac, *Software Quality Measurement: A Framework for Counting Problems and Defects*; Software Engineering Institute, Pittsburgh, PA, CMJ/SEI-92-TR-22, September 1992.
- [20] D. Gause and G. Weinberg, *Exploring Requirements—Quality Before Design*, Dorset House, 1989. A Davis, *Software Requirements*, Prentice-Hall, Englewood Cliffs, NJ, 1993.
- [21] Supporting data for this has been reported by many people in many places. Here we cite data presented by Roger Fujii of Logicon, Inc. at the NSIA Fall Joint Conference in 1987.

- [22] See [7]. This is an ongoing project that will be publishing additional reports in 1995.
- [23] Available data varies widely on how much equipment downtime results from software failures, but in all reports we have seen it is one of the factors reported. In many it is the leading factor. Some reports show it is involved in over 50% of the failures.
- [24] B. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, 1981.
- [25] The two publications that document the Capability Maturity Model for Software are:
- M. Paulk, B. Curtis, M. Chrissis, and C. Weber, *Capability Maturity Model for Software, Version 1.1*, Software Engineering Institute, Pittsburgh, PA, CMU/SEI-93-TR-024.
 - M. Paulk., C. Weber, S. Garcia, M. Chrissis, and M. Bush, *Key Practices of the Capability Maturity Model, Version 1.1*, Software Engineering Institute, Pittsburgh, PA, CMU/SEI-93-TR-025.
- [26] The description of how things are at each level of maturity (Table 1) reflect the author's experiences and may differ somewhat from the SEI's definitions for each level. We offer them here in the hope that they describe modes of operation to which you can easily relate. The material in Table 1 is only slightly modified from the following publication: S. McConnell, "From Anarchy to Optimizing," *Software Development*, July 1993, pp. 51-56.
- [27] The complete documentation referred to here are the two documents identified in [1], which together number more than 500 pages.
- [28] The Software Engineering Institute has at various times reported on the number of organizations not proceeding with improvements programs after completing an assessment of their software process. These reports have consistently been between 60% and 70%.
- [29] J. Kotter, "What Leaders Really Do," *Harvard Business Review*, May-June 1990, pp. 103-111.
- [30] The most comprehensive documentation to date for the IDEAL framework is the material distributed to participants at a half-day tutorial on August 22, 1994, offered as part of The Software Engineering Symposium, an annual event sponsored by the Software Engineering Institute. The document consists of copies of the foils presented: B. Peterson and R. Radices, *An Integrated Approach to Software Process Improvement (SPI)*, not otherwise identified.
- [31] P. Fowler and S. Rifkin, *Software Engineering Process Group Guide*, Software Engineering Institute, CMU/SEI-90-TR-24, September 1990.
- [32] W. Utz, *Software Technology Transitions*, Prentice-Hall, Englewood Cliffs, NJ, 1992.

**Customer Service
SEMATECH Technology Transfer
2706 Montopolis Drive
Austin, TX 78741**

**internet: info@SEMATECH.org
fax: (512) 356-3081
phone: (512) 356-SEMA**