

**Software Process Improvement (SPI)
Guidelines for Improving Software:
Release 5.0**

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

Software Process Improvement (SPI) Guidelines for Improving

Software: Release 5.0

Technology Transfer # 96103188A-ENG

SEMATECH

October 31, 1996

Abstract: This document provides semiconductor manufacturers and equipment suppliers with a proactive, cooperative, and preemptive method for improving the reliability and overall quality of software in fabs and process tools. This approach is being developed and implemented by SEMATECH member companies and their suppliers, coordinated by SEMATECH's Software Process Improvement (SPI) Project. The method advocates four basic elements: a management *policy* to support and measure software quality; *functions* to produce high-quality software on time and within budget; *stages* for building quality-enhancing functions into a software process; and *partnering* to facilitate and expedite organizational change to improve software quality. This SEMATECH-coordinated program is based on the Capability Maturity Model (CMM) from the Software Engineering Institute (SEI). This document supplements, but does not replace, *Software Process Improvement (SPI) Guidelines for Improving Software: Release 4.0*, Technology Transfer #95082943A-ENG.

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

Approvals: Ted Ziehe, Author
Harvey Wohlwend, Project Manager
Gary Gettel, Director, Factory Integration
Dan McGowan, Technical Information Transfer Team Leader

Table of Contents

1	EXECUTIVE SUMMARY	1
2	SEMICONDUCTOR MANUFACTURING NEEDS HIGH QUALITY SOFTWARE	2
2.1	The Software Challenge	2
2.2	Software Quality Improvement Policy	3
2.3	Recap and Preview	6
3	PROCESS, MATURITY, AND SOFTWARE QUALITY	7
3.1	Maturity and Software Quality	8
3.2	Outcomes of a Mature Process	9
3.3	Rating the Maturity of a Process	10
3.4	Functions of a Mature Process.....	14
3.5	Recap and Preview	19
4	USE “PROCESS” TO IMPROVE SOFTWARE QUALITY	20
4.1	Stage 1: Complying with the Policy	21
4.2	Stage 2: Software Process 101.....	22
4.3	Stage 3: Addressing the “Big 6” Issues	22
4.4	Stage 4: Gaining Basic Management Control	23
4.5	Stage 5: Establishing Your Process	25
4.6	Stage 6: Mastering Process Measurement	26
4.7	Stage 7: Gaining Full Process Control	27
4.8	Recap and Preview	28
5	USE “PARTNERING” TO REDUCE RISK AND SPEED PROGRESS	29
5.1	Role of a SEMATECH Member Company	31
5.2	Role of a Supplier	31
5.3	Role of SEMI/SEMATECH.....	32
5.4	Role of SEMATECH’s SPI Project	33
5.5	A Final Word	35
6	REFERENCES	38
	APPENDIX A Complying with the Policy on Software Quality Improvement.....	41
A.1	Examples of Quality Indicators: All Eight Subjects.....	47
	APPENDIX B ORGANIZING TO SUPPORT SOFTWARE QUALITY IMPROVEMENT ...	49
B.1	Core Functions.....	49
B.2	Core Extensions.....	49
B.3	Support Functions.....	50
B.4	Improvement Functions	50
B.5	Organizational Context.....	51
	APPENDIX C REPORTING IMPROVEMENTS IN SOFTWARE QUALITY	53
	APPENDIX D BENEFITS OF IMPROVING SOFTWARE QUALITY	59
D.1	Reported Benefits	59
D.2	Cost of Unreliable Software	59

List of Figures

Figure 1	SEMATECH Members' Software Quality Improvement Policy.....	4
Figure 2	Quality Metrics from the Policy on Software Quality Improvement.....	6
Figure 3	Maturity Levels in the Capability Maturity Model from the SEI	11
Figure 4	Improvement Strategy and Key Process Areas of the CMM.....	13
Figure 5	Categories of Functions for Maturing a Software Process	15
Figure 6	Materials That Constitute Software	16
Figure 7	Key Relationships of the SPI Project.....	34
Figure 8	Keeping the Productivity Engine on Track	36
Figure 9	Five Requirements for Organizational Change.....	37
Figure 10	Example of a Software Engineering Process Group.....	51
Figure 11	SSQA Module 3: Process Maturity	54
Figure 12	Management Commitment Composite Rating	55
Figure 13	SPI Composite Index Baseline	57

List of Tables

Table 1	Software Practices in Five Member Company Fabs.....	2
Table 2	Characteristics of Immature Software Organizations	7
Table 3	Definitions of Function, Process, and Maturity.....	8
Table 4	Characteristic Outcomes of a Mature Software Process.....	10
Table 5	Profiles of the Five Maturity Levels in the Capability Maturity Model	12
Table 6	Composition of the Capability Maturity Model at Levels 2, 3, 4, and 5	14
Table 7	Key Process Areas and Goals for Maturity Level 2 (Stage 4)	24
Table 8	Key Process Areas and Goals for Maturity Level 3 (Stage 5)	26
Table 9	Key Process Areas and Goals for Maturity Level 4 (Stage 6)	27
Table 10	Key Process Areas and Goals for Maturity Level 5 (Stage 7)	28
Table 11	Mission and Objectives of the SPI Project	34
Table 12	Management versus Leadership.....	50
Table 13	SPI Composite Index	56
Table 14	Reported Benefits of SPI Among 13 Organizations	59
Table 15	Assumptions Used in Assessing Software-Related Unplanned Downtime.....	60
Table 16	Estimated Costs of Software-Related Outages	60
Table 17	One Supplier’s Ratio of Field Engineers to Process Tools	61

Acknowledgements

This material reflects the insights and understandings of the entire SEMATECH SPI Project staff, gained from work with more than 40 supplier companies (members of SEMI/SEMATECH). It also incorporates a perspective provided by members of the SPI Project's Technical Advisory Board—representatives of the SEMATECH member companies, SEMI/SEMATECH, Sandia National Laboratories, and the Software Engineering Institute of Carnegie Mellon University—who serve as the project's steering committee.

1 EXECUTIVE SUMMARY

Today's semiconductor fabs require a lot of software: a shop floor control system, process and engineering databases with various application programs, and software embedded in the equipment that makes, moves, measures, assembles, and tests the products the fabs manufacture. The operational reliability of this assemblage of software—estimated to be about 50 million lines of code—is very significant when calculating the productivity and profitability of a fab.

Even so, software gets little attention until it fails. Then, under severe schedule and economic pressures and with consternation, technicians must scramble to resolve problems as they arise.

This document presents a proactive, cooperative, and preemptive way to improve the reliability of fab software and its overall quality. This approach is being developed and implemented by SEMATECH member companies and their suppliers, coordinated by the SEMATECH Software Process Improvement (SPI) Project. Based on the concept of a “mature” process for producing and supporting software, the approach has four elements:

1. A *policy* that clearly states the importance of software quality, identifies several keys to improving it, and defines a set of metrics that measure basic policy compliance.
2. *Functions* that, properly implemented within a software process, regularly produce software on time and within budget, provide objective indicators of the software's quality level, and continuously improve the level of quality achieved.
3. *Stages* for building these quality-enhancing functions into a software process, thereby maturing it and increasing the level of software quality delivered to customers.
4. *Partnering* that facilitates and expedites the change in an organization's culture that is required to successfully improve software quality.

This SEMATECH-coordinated program is based on the Capability Maturity Model (CMM)—a comprehensive framework for guiding the work of maturing a software process and providing the staff capabilities required. The CMM, from the Software Engineering Institute (SEI), is compatible with two other widely used maturity guides:

- *Module 3 of the Standardized Supplier Quality Assessment (SSQA)*, the component for software operations in an assessment tool developed and used by SEMATECH constituents.
- *The ISO-9001 standard (with ISO 9000-3 for Software)*, developed and supported by the International Organization for Standardization and widely used internationally.

The SEMATECH/SPI approach helps an organization use any (or all three) of these guides to establish, mature, and effectively use its software process, a process that an organization must tailor to its own particular needs and changing situation.

This document sketches the SEMATECH/SPI approach from the perspective of a “chief change agent.” Such a person is a software practitioner who has a vision for software, and uses that vision to drive the organization's initiative to improve software quality. The chief change agent must translate the vision into a practical improvement program for his/her organization. This requires communicating about the vision and the change process with the senior manager who owns the issue of software quality, and who also sponsors and is accountable for the initiative to improve it. In addition, aspects of the vision must be shared with software practitioners, many of whom help change the software process and then take ownership of it in its improved form. The

SPI Project's intent is for this document to help chief change agents clarify and communicate their software vision.

Readers are invited to send questions, requests for more information, and inquiries about participating in the improvement program for SEMATECH constituents to:

Harvey Wohlwend	Telephone:	512/356-7536
SEMATECH	FAX:	512/356-3575
2706 Montopolis Drive	InterNet:	harvey.wohlwend@sematech.org
Austin, Texas 78741		

2 SEMICONDUCTOR MANUFACTURING NEEDS HIGH QUALITY SOFTWARE

In the semiconductor fabs of the 1990s, several hundred pieces of equipment make, move, measure, assemble, and test products—taking a product through as many as 360 process steps that use equipment from several dozen suppliers. Each piece of equipment has embedded software—from 100,000 lines of code to more than 1 million. The embedded software is usually networked to a shop floor control system, which in turn interacts with various database managers and application programs. This complete assemblage of software is estimated to exceed 50 million lines of executable code.

2.1 The Software Challenge

The usability and reliability of a fab's software—all 50 million lines of it—are quite significant in calculating fab productivity and profitability [1]. But a recent study of software issues in member company fabs [2] reveals that the practices cited in Table 1 are commonplace.

Table 1 Software Practices in Five Member Company Fabs

- | |
|---|
| <ul style="list-style-type: none"> • Data is collected on equipment downtime, but there is ... <ul style="list-style-type: none"> ⇒ no distinction made among hardware, software, and system failures ⇒ no percent of failures due to software ⇒ no Pareto for types of software failures ⇒ no breakdown of software failure symptoms • Downtime data is logged by both operators and technicians, but there is ... <ul style="list-style-type: none"> ⇒ only informal and on-the-job training for operators ⇒ little effort made to distinguish failure causes ⇒ no estimate made of how much software down-time is costing |
|---|

Clearly, using 50 million units of software effectively, coming as it does from many different suppliers, is a challenge under the best of circumstances. The practices cited above add to the challenge, and the following industry trends make it even more difficult:

- Increasing use of equipment in clusters and cells with complex control requirements
- Emerging applications of sensor technology for regulating process operations
- More frequent reconfigurations of equipment in a fab for short product runs
- Widespread use of automated material handling and mini-environments
- Automation of the total fab

Meeting these and the special requirements of individual manufacturers is causing the volume of embedded software to increase more than 25% per year. This trend is a factor in the rapidly rising cost of equipment, which in turn is driving up the cost of fabs. Now that a fab costs a billion dollars or more—70% of which is cost of the equipment—overall equipment effectiveness (OEE) and the equipment’s cost of ownership (COO) are critical. In summary, as requirements get more complex and software volumes increase, the industry’s need for increasingly reliable software—that raises OEE and lowers COO—is more critical than ever.

2.2 Software Quality Improvement Policy

In spite of this software dilemma, the business case for improving software quality is compelling [3], and SEMATECH member companies are responding with the following actions:

- Improving the quality of software their own organizations produce
- Evaluating suppliers’ product quality and software capability during procurement
- Chartering positions for software specialists who work with equipment suppliers
- Establishing strategic partnerships with key suppliers to improve product reliability
- Giving visibility to suppliers who successfully improve software quality
- Chartering and funding the SPI Project at SEMATECH

However, the SEMATECH member company **drive to improve fab software reliability can succeed only if each supplier takes ownership of and is accountable for the quality of its software**—with accountability defined as follows:

- Senior management’s ongoing commitment to improve
- The improvement goals set and the level of effort invested in achieving the goals
- The staff’s competence with software metrics and continuous improvement
- The network of relationships established to improve the quality of delivered software

To give the drive for improvement the focus it needs, SEMATECH members are establishing a **Policy on Improving Software Quality**. It cites the high priority that members put on software quality, provides guidance for achieving such quality, and gives criteria for evaluating policy compliance. Figure 1 contains the policy’s full text.



2706 Montopolis Drive · Austin, Texas 78741-6499 · 512-356-3500

SEMATECH MEMBER COMPANIES SOFTWARE QUALITY IMPROVEMENT POLICY

October 1996

DRAFT

Software's role is critical to productivity and profitability in Semiconductor Wafer Fabrication, Assembly, and Test operations. For many suppliers, software is now also a significant factor in their competitive position and overall business success. Recognizing this, the SEMATECH Member Companies expect their equipment and system Suppliers to continuously improve reliability and the overall quality of software by the following:

1. Establish an organizational commitment to improving software quality, driven by objective Process Assessment Findings from SSQA ratings and/or SEI maturity levels.
2. Institutionalize continuous improvement by adopting Software Quality Goals and Improvement Action Plans that achieve the goals, as evidenced by recognized metrics.
3. Establish and maintain a Problem and Corrective Action Reporting System that drives continuous improvement in equipment and system reliability.
4. Establish and maintain a Software Revision Control System to assure reproducibility of all delivered software and proper application of corrections and upgrades.
5. Charter staff and allocate budget to the work of continuously improving software quality, objectively validating the quality of software produced, and verifying the software process used.
6. Provide the following data as part of each major product release:
 - Software Test Plan and Test Report.
 - Release Notes that include documentation for all changes and a list of all known problems in the delivered software, with a recommended procedure for recovery when the problem is encountered.
 - Compliance status with the GEM/SEM and SECS Interface Standards.
7. Show compliance with this policy by using the following metrics:
 - Software quality (defect trend)
 - Software reliability (MWBI/MTBF)
 - On-time delivery (schedule adherence)
 - Software process maturity (SSQA Module 3 or SEI Levels)

Detailed specifications for these metrics are available from the SPI Project at SEMATECH, or from any Member Company representative on the SPI Project's PTAB (Project Technical Advisory Board).

A Supplier's Assessment Findings (Item 1), Quality Goals and Action Plan (Item 2), and data for all four metrics (Item 7) are to be provided upon request by any SEMATECH Member Company or by SEMATECH as part of a SEMATECH program.

A Team of America's Best AMD • Defense Advanced Research Projects Agency (DARPA) • Digital Equipment Corporation
Hewlett-Packard Company • Intel Corporation • IBM Corporation • Lucent Technologies • Motorola
National Semiconductor Corporation • Rockwell International Corporation • Texas Instruments Incorporated

Figure 1 SEMATECH Members' Software Quality Improvement Policy

The policy is currently under review by the appropriate member company boards and councils. However, several SEMATECH members are already implementing it within their equipment and system procurement activities. Suppliers are responding by producing and putting to use the four metrics cited in Item 7, and graphically illustrated in Figure 2. Of the four metrics, the first two relate to the *software* itself:

- *Software Quality*—measured in terms of the defects detected in software after its release to customers. The key metrics are defects discovered and defects fixed during a specific time period, and defects remaining open at the end of the time period. The key indicator includes these three data items, measured monthly, and plotted as a trend chart.
- *Software Reliability*—A measure of software’s ability to perform to specifications within its operational setting without failing for a certain number of wafers (MWBI for wafer processing equipment), or time period (MTBF for other equipment). Reliability is calculated from either test results or production operations by logging failures and then using standard methods to compute MWBI or MTBF. The key indicator is reliability calculated monthly and plotted as a trend chart.

The other two metrics relate to a supplier’s *software process*:

- *On-Time Delivery*—A measure of an organization’s ability to adhere to the schedules it sets. Key metrics are the total number of tasks scheduled for completion during a time period and the number of tasks completed. The key indicator is the ratio of completions to total tasks, computed monthly, and plotted as a trend chart.
- *Software Process Maturity*—The measure of an organization’s ability to develop quality software, on time and on budget, as defined by the SEMATECH Standardized Supplier Quality Assessment method (SSQA, Module 3–Software Quality). This method covers twelve fundamental requirements for good software development practices, rating each (using a standardized scoring matrix) on four factors: management commitment, systems approach, deployment, and results. The key indicator is a one-page summary of the four scores for each of the 12 SSQA requirements, determined annually by a self-assessment, and plotted as a bar chart. An alternative is an assessment based upon the *Software Engineering Institute Capability Maturity Model for Software* (SEI/CMM). The key indicator in this case is the organization’s satisfaction of SEI Levels 2 and 3 Key Process Areas (KPA’s).

A supplier with current and historical data for all four of these metrics has taken a major step in complying with the policy. These data, objective indicators of quality, are evidence the SEMATECH members want, but also data a supplier must have to drive its improvement initiative. See Appendix A for more discussion of policy compliance and the importance of objective quality indicators.

Software Quality Indicators (4-Ups)

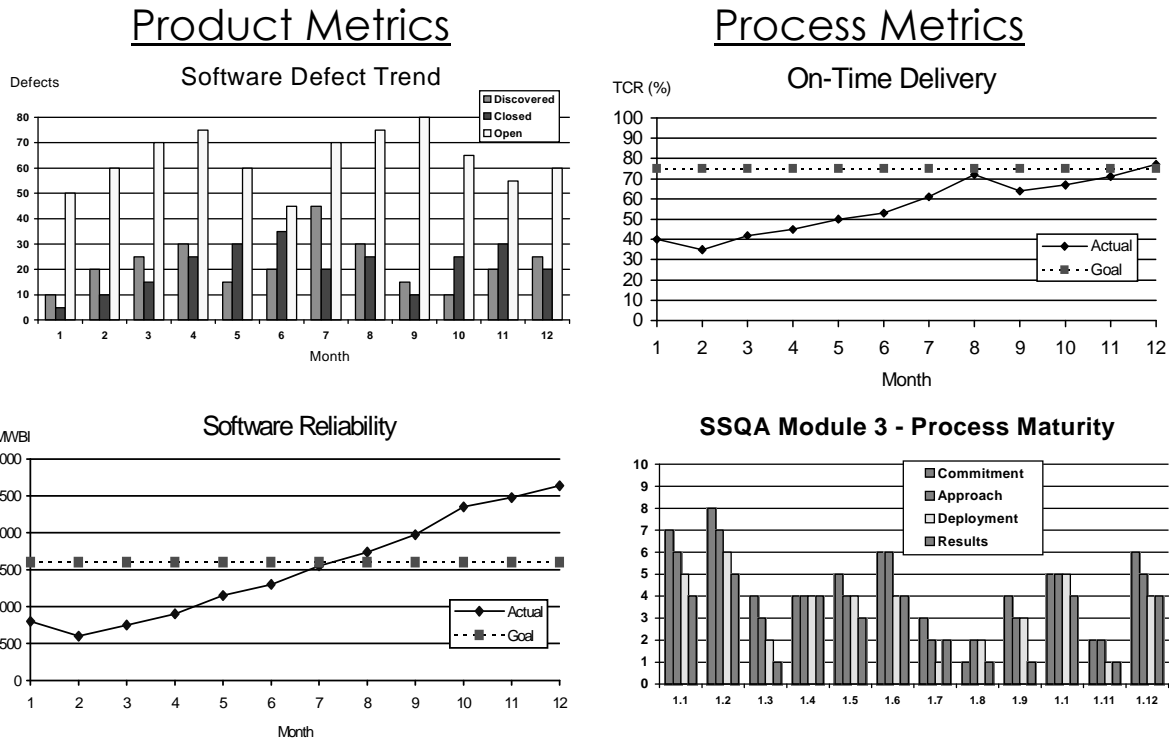


Figure 2 Quality Metrics from the Policy on Software Quality Improvement

2.3 Recap and Preview

SEMATECH member companies are establishing a policy on software quality improvement to say the following:

- Highly reliable software is critically important to the success of fabs.
- Each supplier is expected to provide software that meets quality standards set by its customers.
- A software release is to be accompanied by objective evidence of its quality, using indicators agreed upon with customers in advance.
- The objective indicators that are used must show, over time, increases in quality for all delivered software.

Complying with this policy requires much more than coding competence. It requires effectiveness in the broad range of organizational functions identified in Section 3. An organization's software process—the one that it actually uses—is its institutionalized form of such functions. For a supplier's software process to be mature, it must continuously improve the quality of the software it produces, by increasing its own level of maturity. Section 4 sketches a path for continuous improvement of the software process, a path that leads to high software quality and high process maturity. The path to quality that is sketched takes reliability as the most important attribute of quality.

As suppliers and their customers define quality and work to achieve it, there are inherent risks and the rate of progress is always an issue. Section 5 presents opportunities for partnering with customers, other suppliers, and the SPI Project at SEMATECH—partnering that reduces risks and increases the rate of progress.

3 PROCESS, MATURITY, AND SOFTWARE QUALITY

Two out of three organizations developing software today use an ad hoc, often chaotic approach [4]. Some produce good software, but always at high cost in terms of defects, late deliveries, qualification of software for production use, and work stoppages after qualification. The costs are too high in both financial and human terms (e.g., rework time, staff burnout and turnover, and lack of return on investments in tools and training). Table 2 lists some characteristics of organizations relying on this approach.

Table 2 Characteristics of Immature Software Organizations

<p>An organization's software approach is immature when ...</p> <ul style="list-style-type: none"> • Software quality is unpredictable, unknown • Projects often overrun budgets and schedules • Testing and reviews often are skipped because of schedule pressure • The number of defects in delivered software is high • Management is reactionary, focused on firefighting and crises • Success depends on individual talent and heroic efforts • Process is ad hoc and improvised, sometimes chaotic • Benefits from technology improvements are lost in confusion • Personnel practices do not cultivate software talent • Software strategy does not relate to corporate success factors

The past ten years have clarified much about “engineering” software and “maturing” the software process to increase quality—but too few organizations are applying what’s being learned. For example, this statement ...

“Software’s quality is determined by the maturity of the process that produced it.”

... is often repeated, and now is widely accepted because there is much confirming evidence [5]. But, as cited above, most organizations have not yet applied it.

In this document, that statement about quality, maturity, and process is treated as a two-step recipe for high quality software:

1. Follow a *process*—that is, discipline your ad hoc and chaotic activities.
2. Incorporate within your process *functions that mature it*.

The concept is very simple and logical, but it’s not easy to implement; so, this section and the next examine both steps. This section begins by looking at what it means to say “maturity puts quality into software,” and continues with a look at specific functions that add maturity to a

software process (step 2). Against that background, Section 4 shows why and how “process” impacts software quality (step 1), and the role of a “continuous improvement process” in producing high quality software (back to step 2 again).

Two important concepts in both steps are “process” and “function.” Table 3 provides definitions that relate them, and that distinguish a “mature” software process from software processes in general, including the prevalent “ad hoc and chaotic approach.”

Table 3 Definitions of Function, Process, and Maturity

<i>function</i>	One of a group of related actions contributing to a larger action (or process). [from <i>Webster’s Third New International Dictionary, Unabridged, 1986</i>]
<i>process</i>	The series of steps used to bring about a desired result. [from the <i>Concise American Heritage Dictionary, revised edition, 1987</i>] A progressively continuing operation that consists of a series of controlled actions or movements systematically directed toward a particular result or end. [from <i>Webster’s Third New International Dictionary, Unabridged, 1986</i>]
<i>software process</i>	The set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products. [from SEI’s 93-TR-24, p.3] [7] A set of activities that transforms a user’s computing requirement into a program which operates within the intended user environment.
<i>mature software process</i>	A set of systematic activities that transforms a user’s computing requirement into a program which operates satisfactorily within the intended user environment.

3.1 Maturity and Software Quality

If the maturity of a software process determines the quality of the software produced, then the following statements are also true:

- Immaturity lowers software quality.
- Trying to improve quality after software has been produced is a questionable strategy.
- Maturity must be put into a process before high-quality software can be expected from it.

Many senior managers, unfamiliar with these quality/maturity relationships or simply not believing them, think software quality depends solely on the capabilities of their staff. Having capable people is always an advantage, but a recent study of success factors in software engineering projects [6] put “staff competence” in seventh place, lower than factors such as the following:

- User involvement
- Executive management support
- Clear statement of requirements
- Proper planning

Think about it: executive management support is more important to the success of software projects than the competence of practitioners who develop the software! That insight is worth noting and exploring. In fact, all four of the success factors listed above are involved in rating the maturity of a software process.

So what is a “mature” software process? What does “maturity” make possible?

Simply put, a software process is “mature” when—in addition to producing software—it efficiently finds defects and repairs them, seeks proactively to prevent defects, produces objective indicators of the software’s quality, and uses the indicators to change the software process and improve software quality.

In terms of this definition, it can be said that a mature software process is one that is both complete and effective. To be complete, it must have four parts:

1. The first part produces software—executable code plus its associated materials.
2. A second locates and repairs defects and proactively looks for ways to prevent them.
3. Another periodically produces objective indicators of the software’s quality.
4. The fourth repeatedly identifies, plans, and makes process changes to improve quality.

In addition to being complete, a mature process is also effective. Effectiveness is measured by increases in three indicators: software quality, customer satisfaction, and staff productivity.

This definition gives meaning to the statement “maturity puts quality into software” by “continuously improving the software process.” Even though the statement is very simple and clear, implementing maturity is not easy (see Section 4); nor is it without risks (see Section 5). Appendix B provides information on staffing and organizing to support software quality improvement through continuous process improvement.

But before moving on to implementation, process maturity must be examined in more detail. For example, if maturity is the key to software quality:

- What are characteristic outcomes of a “mature” process?
- What is used to rate or measure the “maturity” of a software process?
- Can a “mature” process be prescribed—not hypothetically, but in a way that fits one’s particular organization and situation?

These questions are answered below.

3.2 Outcomes of a Mature Process

“What we want from our software group is executable code,” many senior managers say. And they insist on schedules, set budgets, and allocate staff and other resources in a manner that expresses that view. But a *mature* software process produces various materials in addition to code—and the quality of the outcomes steadily increases. Table 4 lists several of the additional outcomes—what is not produced, as well as what is. For example, producing “fewer defects” is an important result because it indicates increasing quality.

The last two outcomes listed in Table 4 are especially important. Both are used in defining process maturity. Having objective, quantified evidence of quality is important because it means you have an agreed-on definition of quality (remember, your customers help to define quality), and you have established a metric for quantifying the level of quality. With these, you and your

customers can communicate about quality clearly, simply, and accurately. The policy presented in Section 2 establishes two metrics as objective evidence of software quality, and two metrics that measure process maturity.

Objective evidence is important for a second reason; it provides a basis for continuously improving quality. Continuous improvement is the most reliable road to high quality software—in fact, it is the only road. Quick, one-shot fixes—known as “silver bullets”—cannot solve your software problems. Continuous improvement puts a high priority on figuring out what must be done, planning the required changes, and executing the plans. This calls for energetic leadership as well as time and effort by capable and experienced staff. If either is lacking, it will not happen.

Yes, a mature software process is comprehensive. If your organization’s process is not, something is missing. It is safe to say that “something is missing” for many organizations.

Table 4 Characteristic Outcomes of a Mature Software Process

In addition to executable code, a “mature” software process also

- Delivers software on schedule and within budget—regularly
- Keeps defects-released-to-customers and mistakes-made-with-customers low
- Locates defects, and fixes them quickly—in priority order
- Handles software baselines, evolutionary versions, and audit trails of change
- Produces the various materials that, combined with code, form useful software
- Provides objective evidence of the software’s level of quality
- Persistently identifies ways to improve the software produced and the process

3.3 Rating the Maturity of a Process

The outcomes of a process reveal much about its maturity (see above). In fact, the most comprehensive tool yet developed for rating software process maturity uses, as its basis, outcomes (expressed as goals) along with practices that accomplish the goals. This tool is the Capability Maturity Model (CMM) for Software [7], developed and supported by the Software Engineering Institute (SEI) at Carnegie Mellon University. The SEI, a major force in the widespread improvements being made in software engineering, “is committed to the evolution of software engineering from an ad-hoc, labor-intensive activity to a managed, technology-supported engineering discipline” [8].

The SPI Project is using the CMM as a de facto standard in improving the quality of supplier software executing in SEMATECH member company fabs. The CMM is comprehensive and very detailed, but its architecture is simple, logical, and quick to grasp. This section profiles that architecture, explains what the CMM brings to the work of improving quality, and shows how the CMM for Software relates to other improvement guides—namely, the SSQA method (Module 3) and ISO 9001 (as interpreted for software by ISO 9000-3).

The CMM for Software models process maturity in terms of the following:

- Selected *areas* of software activity on which the model is focused
- Specific *goals* to accomplish in the activity areas selected (goals = results, outcomes)
- Various *practices* for accomplishing the goals (producing the results or outcomes)

The CMM has four parts, each of which is a construct of areas, goals, and practices. One part models “full maturity.” The other three depict maturity at stages or levels leading up to “full maturity.” Figure 3 shows this progression as “five levels of maturity” with “full maturity” at Level 5, and Levels 2, 3, and 4 leading up to it. Maturity Level 1 has no areas, goals, or practices associated with it, so it represents the original or ad hoc stage.

These four parts of the CMM—used individually or as a unit—serve as templates for determining how mature an organization’s software process is. Such a rating is the result of an objective assessment in which CMM templates are compared with evidence collected by a team of appraisers. This comparison determines which of the model-prescribed areas of activity are present within the organization’s software process, which goals it achieves on a regular basis, and which practices it uses consistently.

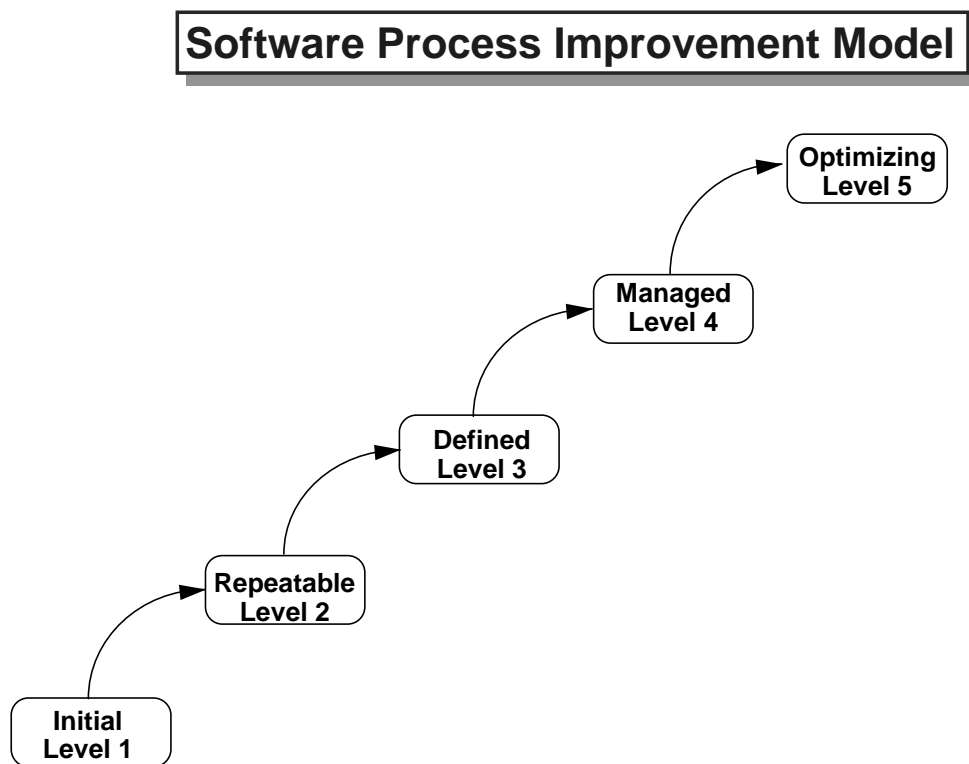


Figure 3 Maturity Levels in the Capability Maturity Model from the SEI

Not only is the CMM used to rate process maturity, it also is a roadmap charting a path from low maturity to high. Named areas of activity define the scope of the model at each level of maturity; within each area, goals with practices that achieve the goals give it focus. The progression from one maturity level to the next charts the path. In Table 5, a profile of each level reveals this path and the CMM's underlying strategy for improving software.

Table 5 Profiles of the Five Maturity Levels in the Capability Maturity Model

<ul style="list-style-type: none"> • Maturity Level 1: Initial This first level of the CMM is unique; it has no model of maturity—no areas, goals, or practices. That void means that a software process which fails to qualify at any of the other four levels—those that are unpredictable and poorly controlled, even ad hoc and chaotic—are referred to as Level 1 processes.
<ul style="list-style-type: none"> • Maturity Level 2: Repeatable At this level the model's focus is on basic management control of software operations. Goals and practices in six key process areas identify what is required to repeat previously mastered tasks or projects in a way that gets similar results. For that reason, this level is referred to as "repeatable." By effectively planning and managing each of its software projects, an organization controls this most basic level of its software process.
<ul style="list-style-type: none"> • Maturity Level 3: Defined At this level, the model focuses on an established software process that is used throughout the organization. Goals and practices in seven areas clarify and characterize the entire process, making it understood and regularly used organization-wide. Once established, the process is institutionalized: it is documented, staff and managers are trained in all aspects of it, and it is verified that projects are conducted in a manner suitably consistent with it.
<ul style="list-style-type: none"> • Maturity Level 4: Managed At this level, the model's focus is on mastering techniques for measuring the software process itself, and the quality of software produced by the process. Goals and practices in two key process areas add instrumentation to the organization-wide process that was established at Level 3, allowing that process to be quantitatively understood and controlled, and also facilitating fact-based decision-making and priority-setting by managers.
<ul style="list-style-type: none"> • Maturity Level 5: Optimizing At this final level, the model's focus is on full control of the software process, including the ongoing improvement of both software quality and process maturity. Goals and practices in three key process areas identify important aspects of defect prevention and continuous improvement. At this level, the organization's software process routinely operates well, freeing a larger share of the effort for continuous improvement and process adaptation.

Figure 4 gives more details about the CMM. In addition to the maturity levels, this figure graphically shows the CMM's underlying improvement strategy. It also names (and numbers) the Key Process Areas. They set the scope of the model at each level of maturity. The model's four levels have a total of 18 key process areas, which in turn have a total of 52 goals (or outcomes). For the 52 goals, the CMM lists 150 key practices that accomplish the goals. The model also lists 166 practices for "institutionalizing" the key practices, i.e., ensuring that the organization uses the key practices regularly and consistently. The four types of institutionalizing practices are those that

- Commit the organization to perform the key practices
- Create an ability to perform them
- Ensure that measurement and analysis are performed
- Show diligence about verifying implementation

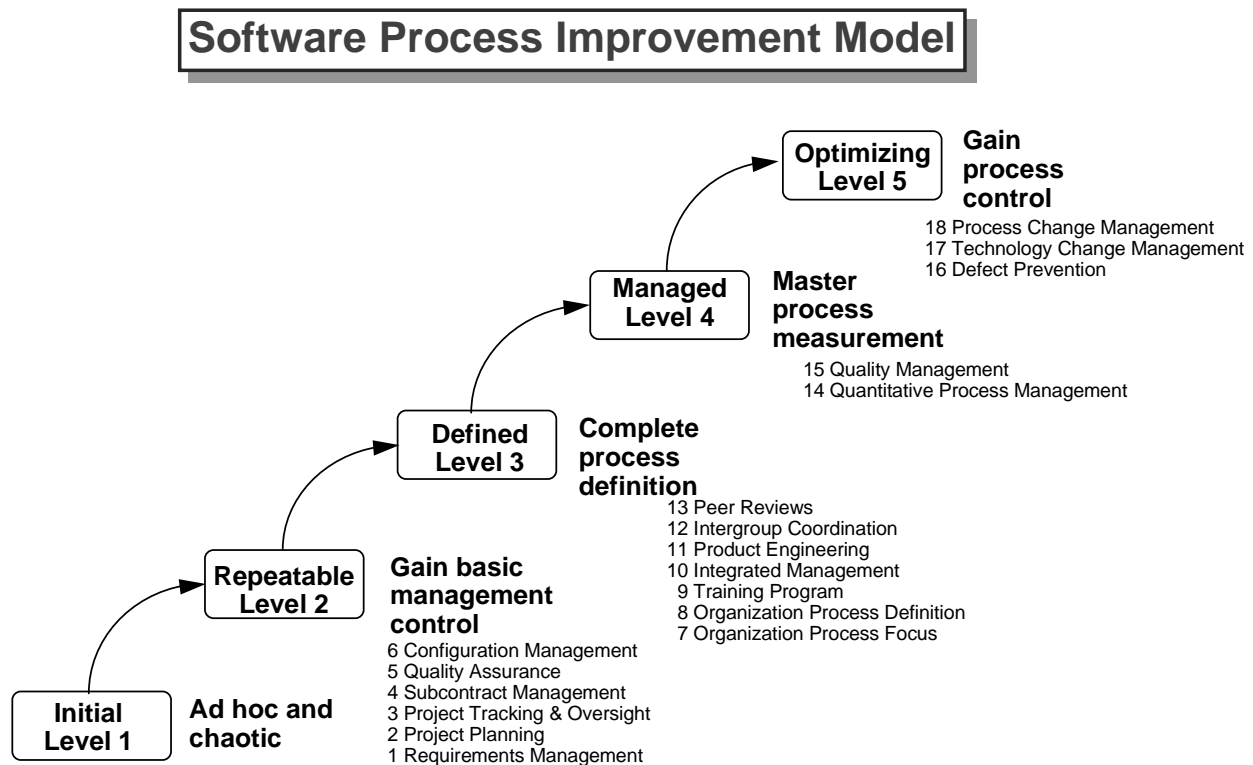


Figure 4 Improvement Strategy and Key Process Areas of the CMM

Table 6 details the CMM's makeup at each maturity level in terms of areas, goals, key practices, and institutionalizing practices. For example, achieving Maturity Level 2 requires work in six areas from which 20 outcomes (goals) are important. The model's checklist of 62 key practices suggest ways to produce the outcomes, with 59 additional suggestions for how to bring the key practices into use. Section 4 presents an approach to implementing practices in areas of activity that accomplish improvement goals.

Table 6 Composition of the Capability Maturity Model at Levels 2, 3, 4, and 5

Maturity Models	L2	L3	L4	L5	Totals
Areas of activity (KPAs)	6	7	2	3	18
Goals (area outcomes)	20	17	6	9	52
Key practices (achieve goals)	62	50	12	26	150
Institutionalizing practices	59	58	19	30	166

The CMM, as a model of progressive maturity, drives an organization to steadily raise its improvement goals and practices. It does this by guiding an organization into new areas of improvement activity as it progresses from one level of maturity to the next.

In contrast, ISO 9001 and SSQA are static standards. The former is effective in that it defines a floor of good practice below which an organization should not fall—namely, establish a software process, document it, and use it with consistency. SSQA sets the quality bar a bit higher with its unstructured list of 12 requirements. Even though the SSQA doesn't model increasing maturity, its 12 requirements have significant overlap with the six CMM areas at level two, plus five of the seven at level three.

The SPI Project can provide more information about any of these three improvement guides. Whichever you choose, you'll need to implement within your software process certain functions that the guide prescribes. Before answering the third question (Can a mature process be prescribed?), it is important to consider the full range of functions involved in maturing a software process.

3.4 Functions of a Mature Process

Two out of three organizations have software processes rated at CMM Level 1. That means they are ad hoc and chaotic, with typically undisciplined uses of the following functions:

- Gather requirements (if offered)
- Create a design (if needed)
- Produce executable code (as fast as possible)
- Test code (if time permits)
- Deliver code (on the due date, even if it is incomplete)

Only extremists still regard coding as the only important function, but far too many organizations are immature software producers because they do not balance priorities, capabilities, and schedules among the five functions listed above, and do not implement other aspects of a mature software capability.

The purpose here is to contrast the limited and inadequate (but popular) approach to software indicated above with a mature software capability. This mature capability is expressed as a range of organizational functions in the five categories named in Figure 5. The various functions cited as aspects of maturity, although not explicitly named by the CMM, are implied by that de facto standard [7]. When an organization implements any of these functions, the form it takes must be suited to the particular organization, and to the function's use and place within the software process. This is an important matter, but beyond the scope of this document.

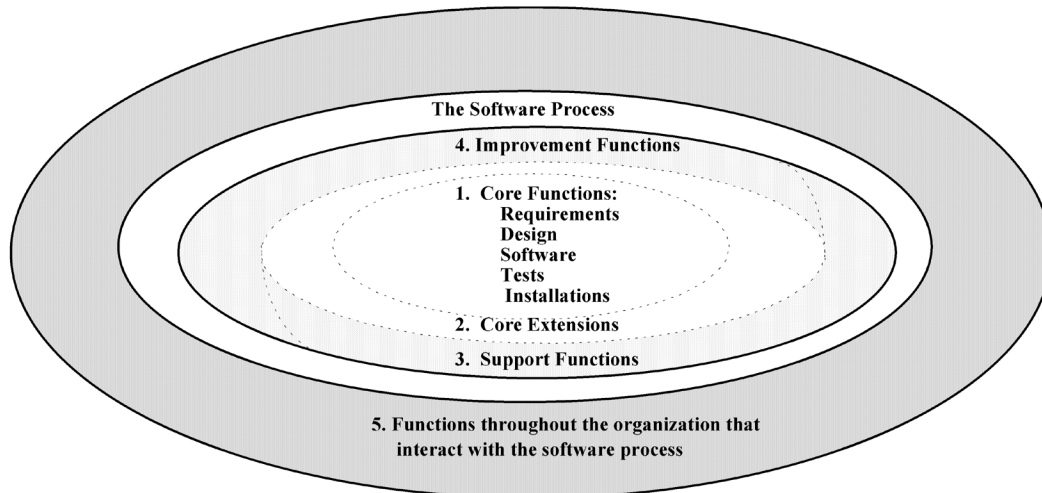


Figure 5 Categories of Functions for Maturing a Software Process

Also, it is important to note that the listing of functions below does not define a software process; rather, these functions are components to use when building a process. Senior managers have an important role in selecting functions to implement, determining the form in which each is implemented, and deciding how to arrange implemented functions into a process. That management responsibility is touched on in Section 4.

1. *Core Functions.* Five functions form the core of software engineering:

- **Requirements.** Gather requirements and requests from customers, proactively when necessary; when received after development has started or after software is released to customers, manage the process that determines what to implement and when, and for those that are selected, see that they impact all software materials, not just the code.
- **Design.** Based on requirements, create and establish an architecture and a design for the product; maintain agreement between requirements and the architecture/design.
- **Software.** Produce code, and with it related materials as required, e.g., documented requirements/designs, user training/reference materials, and release notes; maintain agreement between requirements and all software materials.
- **Tests.** Verify that software is free of defects, and validate that it satisfies customer requirements, at each of several established checkpoints in the software process.
- **Installations.** Install, gain acceptance, and support users of the product in various customer facilities.

- **Installations.** Install, gain acceptance, and support users of the product in various customer facilities.

Establishing this core means determining, for your particular setting, a workflow among the five functions that can be properly scheduled, staffed, and managed. Each function is important, and an organization must find an appropriate balance for the set of functions.

A second important aspect of establishing the core is determining which software materials are needed, and then seeing that all core functions deal appropriately with them. As used here, software refers to code plus some or all of the associated materials listed in Figure 6.

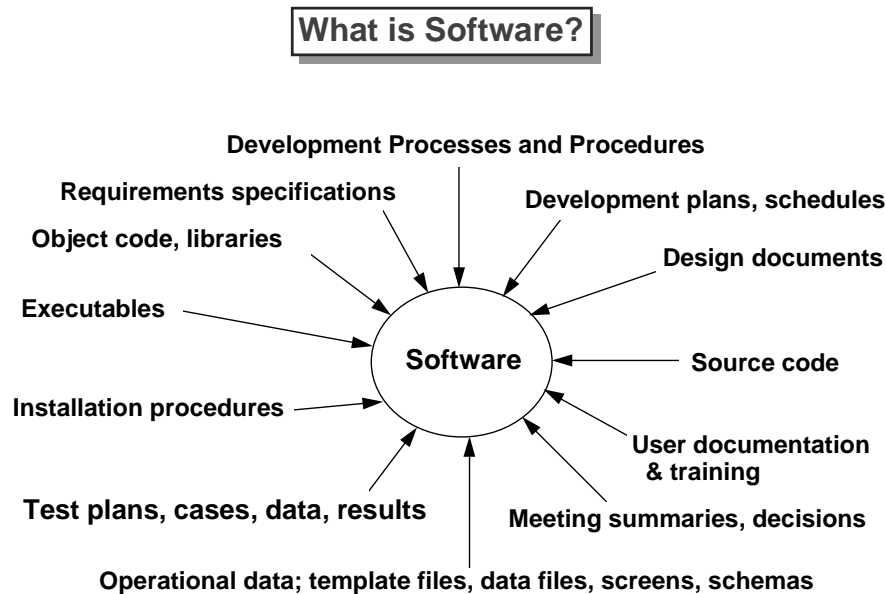


Figure 6 Materials That Constitute Software

The test function within the core finds and fixes software defects. Determining what forms of testing to use, and the points within the process at which tests are used, is a third important aspect of establishing the core of software engineering [9].

Finally, defects, as they are identified, form a basis for the work of defect prevention. Root cause analysis relates defects to the function in which they originated and, when possible, to the cause that produced them. The final step is to adjust the process in a way that prevents, or at least reduces, the injection of such defects in the future. Integrating this activity into core functions is the fourth aspect of establishing the core.

2. *Core Extensions.* This second category is not additional functions, but rather business settings in which the core of software engineering must operate. Following are examples of such settings:

- New development (from scratch) of products, systems, and/or software
- Product evolution and enhancement, sustaining engineering
- Software re-engineering and reuse
- Software maintenance and customization (defect repair and adding specials)
- Software acquisition (and customization)

Each setting requires a form and sequencing of the core functions that are tailored to the needs of that setting. Versions of the core functions are implemented as a process for as many of these settings as are relevant to your business.

3. *Support Functions.* These functions are integral to the software process, but outside the software engineering core described above. Their purpose is to handle all software in a manner that:

- Allows only appropriate access
- Keeps an audit trail as changes are applied
- Assures that quality standards are met
- Builds and transfers to customers configurations of the software

Here, each of three support functions is simply named, and a list of representative responsibilities indicates its scope.

- Problem reporting and change control
 - Custodian of all problem reports and the work of resolving them
 - Defect correction
 - Product enhancement and customization
 - Priority setting, task scheduling/tracking/reporting
 - Change control board administration
 - Root-cause-of-problems analysis
- Configuration management and release control
 - Custodian of all software that is under formal access and distribution procedures
 - Internal distribution and control of software modules during development
 - Product-build and signoff on software releases
 - External distribution and control of software releases
 - Exact reproduction of particular software releases
- Objective verification and validation of products and process
 - Product testing
 - Peer review coordination
 - Validation of process use
 - Project debriefings (lessons learned, postmortems)

Each function involves specialized skills, methods, and tools. Implementing any of the three involves decisions related to tool selection, staff selection and training, and procedures. Staffing will be on a part-time basis in some cases, full-time in others.

4. *Improvement Functions.* These functions, like the support functions, are an integral part of the software process, but outside its core. They provide the leadership, experience, and drive needed to guide and sustain an SPI initiative. The functions can be implemented in any of various forms, but are essential to improving software quality. They ensure that quality improves in a manner beneficial to the organization and consistent with the CMM for Software. Listed with each

function below are representative responsibilities [10], including the CMM KPAs for which the function is responsible.

- Vision for Software, Focus on Process, Assure/Manage Continuous Improvement
 - Software engineering process leader (chief change agent)
 - Key function: KPA 7, Organization Process Focus
 - Management Steering Committee for Software Improvement (includes sponsor)
- Policy and Procedures Working Group
 - Key functions: KPA 8, Organization Process Definition
 - KPA 10, Integrated Software Management
 - KPA 18, Process Change Management
 - Additional functions: risk management plus the following
 - KPA 1, Requirements Management
 - KPA 2, Software Project Planning
 - KPA 3, Software Project Tracking and Oversight
 - KPA 4, Software Subcontract Management
 - KPA 12, Intergroup Coordination
- Training and Staff Development Working Group
 - Key function: KPA 9, Training Program
 - Additional functions: position descriptions with training requirements
- Tools and Methods Working Group
 - Key function: KPA 17—Technology Change Management
 - Additional functions:
 - KPA 5, Software Quality Assurance
 - KPA 6, Software Configuration Management
 - KPA 11, Software Product Engineering
 - KPA 13, Peer Reviews
- Metrics and Database Working Group
 - Key functions: KPA 14, Quantitative Process Management
 - KPA 15, Software Quality Management
 - KPA 16, Defect Prevention
 - Additional functions: management of the data repository

Most organizations charter their software engineers to spend a specified percentage of time on these working groups. In larger organizations, one or more full-time staff are required to provide the leadership and experience needed to improve software quality.

5. *Organizational Context.* Maturity within a software process is not possible without a matching level of maturity in the various organizational functions that frequently interact with the software staff. This category includes four areas of organizational activity that directly impact software quality. Here again, each area is labeled, and representative responsibilities are listed to indicate the scope.

- Organization Management (senior manager responsible for software operations)
 - Policy statements and communication
 - Business plans, goals, and strategies
 - Resource allocations and priorities
 - Process management
 - Compensation, rewards, incentives, recognition
 - Authorizations for staff training
- Product and System Engineering (engineering and manufacturing management)
 - System design with allocation of functions to software
 - Coordination of the various engineering disciplines
 - Mechanical and electrical engineering
 - Product problem resolution and resource sharing
- Interactions with customers (marketing, sales, field support, senior management)
 - Product requirements and schedules
 - Definition of quality and defects
 - Product acceptance and support
 - Product performance and problem reporting
 - Determination of customer satisfaction
- Interactions with suppliers (purchasing, senior management, finance, legal)
 - Dealing with contract software engineers
 - Making and managing subcontracts for software
 - Acquiring software for use within products
 - Acquiring tools and equipment for software engineering

Clearly, senior management has a role in the activities listed above. An organization that does not handle such interactions with maturity cannot grow, cannot mature, and will not be competitive.

3.5 Recap and Preview

This section lists organizational functions that “put maturity into a software process.” Maturity means that the process produces software that is high in quality and low in defects. It also periodically produces objective quality-indicators, and a steady flow of changes to the process that, over time, cause the indicators to show that quality is increasing. Functions that do that “ensure quality in the software produced.”

The material also answers two of the three questions raised at the outset:

- It lists the kinds of outcomes that characterize a mature software process.
- It describes the CMM for Software, a gauge for rating process maturity and a guide for continuously improving software quality.

This section concludes by answering the third question: Can one prescribe a mature software process—not hypothetical, but one that fits a particular organization and situation?

The answer is “No;” each organization must develop its own process. Ready-made, easy-to-use, one-size-fits-all software processes planted within an organization do not work. Even a custom-made process cannot be defined for you by an outside party. The only way to establish a mature software process within an organization is for that organization itself to author it, first by examining and establishing its current process, and then maturing that “as-is” process by enhancing the effectiveness of existing functions and/or incorporating new functions as the need for them arises.

That is the challenge each organization producing software faces, and Section 4 is an aid for meeting that challenge.

Section 5 completes this document with information about ways to

- Reduce the risk of failure
- Increase the incentive for change
- Speed up the pace at which changes are made

4 USE “PROCESS” TO IMPROVE SOFTWARE QUALITY

A company’s success in the semiconductor industry does not depend on what the company knows or can do at any given moment. It rests, instead, on an ability to learn, change, and mature those capabilities that most impact the company’s business. Bluntly put, no one today stays competitive long-term without being intentional about how they learn, change, and mature as an organization—and without being very effective at working the approach to change that they use [11].

Meeting this industry’s software quality challenge (Section 2) requires a company to learn, change, and mature how it produces software. Section 3 describes the mechanism needed: a mature software process that is complete and effective. If your organization’s software process is already mature, or at least well down that road, keep at it. If you, like many of your colleagues, have some distance to go or have not yet started, this section charts a way to produce high quality software, beginning from wherever you are. So, please read on.

A mature software process can improve quality only if quality is clearly defined. Customers define quality, but proactive suppliers will prompt them to identify the attributes with highest priority. The policy presented in Section 2 (from SEMATECH member companies) puts highest priority on “reliability,” and the general purpose path to high quality defined in this section uses “reliability” as the quality attribute to improve. If you have customers with other priorities, you will need additional quality indicators, and it might also change other aspects of your company’s improvement initiative.

A definition from Table 3 bears repeating:

A process consists of “...actions *systematically* directed toward a particular result or end.”

Once quality (the target) is defined (as reliability or some other attribute), the first step to improving quality is to establish your process—don’t try to implement new functions, simply establish a process—and aim it as best you can at your target, as you best understand the target. That simple act turns chaos into systematic, results-oriented action. Clarify who participates in the process and, for each participant, what they do, when they do it, and why.

Establishing such a base process makes each subsequent addition, adjustment, and change to the process easier to do, and more likely to succeed. The following paragraphs sketch a sequence of moves that bootstrap an organization up the maturity scale—from Level 1 to Level 5. Ideally, each stage will be completed before continuing on to the next. While that is not mandatory, it is highly recommended because work not completed at a stage increases the risk of failure for work at the later stages.

4.1 Stage 1: Complying with the Policy

This first stage literally puts an organization on the road to high quality software. The actions required are not difficult, but the motives, insights, and determination of the people taking the actions are critical. There must be:

- Conviction about the need, and the benefits
- Insight into methods, and the need for persistence in applying the methods

The CMM labels this starting point *Level 1—Initial*.

Many organizations are painfully aware that their “as-is” approach to software is not really a process at all. Their people work ad hoc and unmanaged, and in the worst cases software activities are chaotic and out of control. Management has little insight into projects, and budgets and schedules are always changing. But, moved by their customers’ need for very reliable, high-quality, on-time software, and inspired by their own vision for software and its role in the success of their business, the organization commits to putting its software house in order.

Four steps, all responsive to the Software Quality Improvement Policy, launch an organization’s ongoing SPI initiative:

- Step 1: (on a best-efforts basis) At regular intervals, provide the 4-ups data defined by the metrics in the Software Quality Improvement Policy, Item 7 (see Figure 2).
- Step 2: (on a best-efforts basis) Discipline your “as is” software activity into a process so you can begin to analyze, evaluate, and revise it.
 - Study the flow and balance of your core functions (requirements through installations)
 - Consider what materials are produced in addition to code
 - Settle on a definition of quality, with metrics and indicators for level of quality
 - Identify the business settings in which software is produced (e.g., new development, maintenance, evolutionary development, acquisition)
- Step 3: Determine what it means for you to be “totally compliant” with the policy (as discussed in Appendix A) and your status with respect to each item of the policy.

- Step 4: Declare your organization’s commitment to improving software quality to your staff and customers, and also to the SPI Project at SEMATECH.

Completing these four actions prepares an organization for Stage 2.

4.2 Stage 2: Software Process 101

After the launch, this second stage attends to the basics: staffing, support, and orientation to the work at hand. Again, the actions are quite simple, but the capabilities of those selected and the priority put on completing the actions are critically important. The CMM label *Level 1—Initial* still applies for Stage 2.

Frequently, an organization launches an improvement initiative, then lets it die. It is only when senior management takes ownership of the issue, puts it on the organization’s agenda, and gives it priority that you move into Stage 2. The other essential ingredient is having a recognized leader of the initiative—a chief change agent who has a vision and passion for software that motivates the entire organization. With capable people in these two roles, the CMM as a resource, and open channels of communication with customers and the SPI Project at SEMATECH, an organization is in position to improve software.

Five steps follow up the launch and give the SPI initiative focus and substance:

- Step 1: Designate a senior manager as the SPI sponsor and reaffirm the organization’s commitment to improve software quality. This step is essential; continuing without a committed sponsor puts the undertaking at high risk of failure.
- Step 2: Name the chief change agent and equip the person selected with a charter, time allocation and budget, and training if needed. This step, also essential, provides the leadership so critical to any effort that implements change.
- Step 3: Understand the three goals of the CMM KPA #7 (Organization Process Focus) and have a strategy (enough for now) for reaching those goals.
 - Goal 1: Software process development and improvement activities are coordinated across the organization.
 - Goal 2: The strengths and weaknesses of the software process are identified relative to a process standard (initially it is SSQA, later the CMM).
 - Goal 3: Organization-level process development and improvement activities are planned.
- Step 4: Define the organization’s norms for software “projects” and for “business settings” that produce software.
- Step 5: Identify specific projects as candidates for improvement activities.

Completing these five actions prepares an organization for Stage 3.

4.3 Stage 3: Addressing the “Big 6” Issues

After committing resources to SPI, this third stage ensures some quick benefits. By concentrating in areas that customers have identified as troublesome, improvements will be visible and well received. The CMM label *Level 1—Initial* still applies for Stage 3.

A poll of SEMATECH member company representatives identified two areas of major concern about supplier performance—configuration management and testing. Four other areas identified are defect tracking, peer reviews, requirements, and project planning/tracking. Improved performance in any of these area has customer visibility and importance, strengthens customer/supplier interactions, and begins the move toward Level 2 maturity, the objective of the next stage. Findings from the SSQA assessment (see Stage 1) can inform the plans for change in each of the six areas, and influence priorities that order the work. Each change made must be evaluated in terms of its potential for improving product quality, as reflected by the quality indicators being used.

Improving software quality is a strategic initiative, but success over the long term consists of a progression of short-term successes that individually fuel everyone’s motivation to continue, and that cumulatively have greater significance than the sum of their individual benefits.

Four steps begin the work of improving software quality and building an infrastructure to support the ongoing effort:

- Step 1: Apply the findings from the SSQA assessment (see Stage 1, Step 1, 4-Ups Metrics) to ensure that they cover the current conditions and capabilities in each of the “Big 6” areas:
 - Configuration management, version, and release control
 - Product and software testing
 - Defect tracking, e.g., the Failure Reporting, Analysis, and Corrective Action System (FRACAS) method
 - Peer technical reviews and inspections
 - Requirements management
 - Project planning and tracking
- Step 2: Define your organization’s norm for coordinating work on software with other aspects of product production and delivery (reflect this norm in the “as is” process)
- Step 3: Develop plans for and priorities on improvements in each of the “Big 6” areas
- Step 4: Complete first-step improvements in at least four of the “Big 6” areas

Completing these steps prepares an organization for Stage 4.

4.4 Stage 4: Gaining Basic Management Control

This stage makes the transition from “getting underway” (the first three stages) to gaining effective management control of all software production (the work performed in software projects). The CMM label for this level of maturity is *Level 2—Repeatable*.

When an organization’s maturity qualifies as Level 2, its software process is repeatable and under basic management control. This means that project managers are able consistently to make reasonable estimates and plans, and to track and control project performance via these plans. The organization’s best software practices are accumulating at the project level, and there is an easy to recognize difference in the style of work, compared to the earlier stages. Successful projects—in terms of cost, schedule, and requirements—are the norm. The disciplines in place track cost, schedule, and functionality, and allow new projects to draw on earlier, similar projects in ways that repeat their success.

Maturity at this level and beyond is implemented within the norms the organization established for software projects and business settings (in Stage 2) and for relating its software work to all other aspects of product production (in Stage 3).

Representative accomplishments at this level of maturity include the following:

- Policies for managing projects and procedures that implement the policies are established.
- Planning and managing new projects is based on experience with similar projects.
- Successful practices developed on earlier projects are reused on subsequent projects.
- Project commitments are realistic, based on results from previous projects and requirements of the current project.
- Software project managers track software costs, schedules, and functionality.
- Problems in meeting commitments are identified when they arise.
- Software requirements and the work products developed to satisfy them are baselined, and their integrity is controlled.
- Software project standards are defined, and the organization ensures they are faithfully followed.
- The software project establishes strong customer-supplier relationships with subcontractors.
- The project process is institutionalized, meaning it is practiced consistently, documented, trained, and measured, and is able to improve.

For a software process to be rated at this level of maturity, it must be producing results that satisfy the 20 goals (in six Key Process Areas) listed in Table 7.

Table 7 Key Process Areas and Goals for Maturity Level 2 (Stage 4)

1. Requirements Management (KPA 1)	An established baseline of software requirements exists and is used. Software (all aspects of it) is kept consistent with the requirements.
2. Software Project Planning (KPA 2)	Software estimates are documented for use in planning and tracking. Software project activities and commitments are planned and documented. Groups and individuals agree to the commitments that impact them.
3. Software Project Tracking and Oversight (KPA 3)	Actual results and performances are tracked against software plans. Corrective actions are taken that deal with significant deviations from plans. Changes to commitments are agreed to by those affected by the change.
4. Software Subcontract Management (KPA 4)	The subcontractors that are selected are qualified. The contractor and subcontractor agree to their commitments. The contractor and subcontractor maintain ongoing communications. The contractor tracks actual results and performance against commitments.
5. Software Quality Assurance (KPA 5)	The SQA activities are planned. Adherence to standards, procedures, and requirements is objectively verified. Affected groups and individuals are informed of SQA activities and results. Senior management handles noncompliance issues not resolved in projects.
6. Software Configuration Management (KPA 6)	Software CM activities are planned. Selected work products are identified, controlled, available. Changes to identified work products are controlled. Affected groups and individuals are informed about software baselines.

4.5 Stage 5: Establishing Your Process

This stage, building on prior accomplishments, establishes a well-defined, consistently-used, organization-wide software process. The CMM label for this maturity level is *Level 3—Defined*.

At maturity Level 3, the best practices from projects have been generalized for use by the organization as a whole. That means the organization has found a systematic way to share best practices, namely codifying them as the organization's standard software process. Both management and engineering activities that develop or maintain software are documented, generalized, and integrated into a standard process for the organization.

The organization's defined process does not restrict projects, because there is also a systematic way to adapt the accumulated process knowledge to the needs of each new software project that comes along. That means each software project follows an approved, tailored version of the standard process.

Representative accomplishments at this level of maturity include the following:

- There is an organization-wide understanding of the roles and responsibilities for all participants in the software process.
- An organization-wide training program is implemented for managers and technical staff, ensuring they have the knowledge and skills their roles require.
- Projects tailor the organization's standard software process to develop their own defined software process.
- The software process is defined in terms of readiness criteria, required inputs, standards and procedures for performing the work, verification mechanisms (such as peer reviews), expected outputs, and completion criteria.
- Management has good insight into technical progress on all projects.
- Within established product lines, cost, schedule, and functionality are controlled, and software quality is tracked.

For a software process to be rated at this level of maturity, it must be producing results which satisfy the 17 goals (in seven Key Process Areas) listed in Table 8.

Table 8 Key Process Areas and Goals for Maturity Level 3 (Stage 5)

7. Organization Process Focus (KPA 7)	Process development and improvement is coordinated across the organization. Strengths/weaknesses of the software process are identified by a process standard. Organization-level process development/improvement activities are planned.
8. Organization Process Definition (KPA 8)	A standard software process for the organization exists and is maintained. Information about process use by projects is collected, reviewed, maintained.
9. Training Program (KPA 9)	Training activities are planned. Training is provided for both management and technical staff. Individuals involved with software receive the training they need in their roles.
10. Integrated Software Management (KPA 10)	Each project's defined process is tailored from the organization's process. Each project is planned and managed according to the project's defined process.
11. Software Product Engineering (KPA 11)	Software engineering tasks are defined, integrated, and consistently performed. Software work products are kept consistent with each other.
12. Intergroup Coordination (KPA 12)	The customer's requirements are agreed to by all affected groups. Commitments between the engineering groups are agreed to by affected groups. The engineering groups identify, track, resolve intergroup issues.
13. Peer Reviews (KPA 13)	Peer review activities are planned. Defects in the software work products are identified and removed.

4.6 Stage 6: Mastering Process Measurement

Based on consistent, organization-wide use of a software process, this stage masters the use of metrics to measure product quality and process maturity. The CMM label for this level of maturity is *Level 4—Managed*.

At Level 2, the best practices tend to be in projects. By Level 3, the organization has mastered the technique of spreading the best practices across the organization in the form of a standard process. Now, at Level 4, all the process assets accumulated from Level 2 and Level 3 practices are used by the Level 4 organization to support projects with a quantitatively understood, stable process. Detailed measures of the software process and product quality are collected and used in managing the projects.

Representative accomplishments at this level of maturity are as follows:

- The organization sets quantitative quality goals for both software products and processes.
- Productivity and quality are measured for important activities across all projects as part of an organization-wide measurement program.
- Both the software process and its products are quantitatively understood and controlled.
- An organization-wide database is used to collect and analyze data reflecting uses of the defined software process within projects.

- All software projects report prescribed measurements that are well defined and consistent, and provide a basis for evaluating the process used and products produced.
- Projects exhibit control over their products and process by narrowing the variation in their performance, causing it to fall within acceptable quantitative boundaries.
- The risks involved in moving up the learning curve in a new type of software project are known and carefully managed.
- The organization is able to predict trends in process maturity and product quality.
- When the limits set on product quality are exceeded, corrective actions are taken.
- Software products are of predictably high quality.

For a software process to be rated at this level of maturity, it must be producing results that satisfy the six goals (in two Key Process Areas) listed in Table 9.

Table 9 Key Process Areas and Goals for Maturity Level 4 (Stage 6)

14. Quantitative Process Management (KPA 14)
<ul style="list-style-type: none"> The quantitative process management activities are planned. The performance of the project's defined process is controlled quantitatively. The capability of the organization's standard process is known quantitatively.
15. Software Quality Management (KPA 15)
<ul style="list-style-type: none"> The software quality management activities are planned. Measurable goals for product quality and their priorities are defined. Actual progress made to achieving the quality goals is quantified and managed.

4.7 Stage 7: Gaining Full Process Control

When this final stage has been completed, the organization is capable of consistently producing extremely reliable software within predictable cost and schedule limits, so its major focus is on continuous process improvement. The CMM label for this level is *Level 5—Optimizing*.

By the time it has reached Level 5, an organization's process mechanisms operate routinely to deliver its software products. These mechanisms work so reliably that the process of producing software is virtually automatic. There is even a process for handling deviations in processes. Everyone knows what to do next and whose job it is to do it.

So, what do the people in a Level 5 company do? They produce software that is reliable, and they do it with high productivity. The process is so effective they can readily adapt it for strategic business reasons, e.g., to gain competitive advantage in quality, in productivity, and/or in timely delivery—or in combinations of these three. When a process is effective, efficient, and under control, people can focus on improving it and adapting it to new situations.

Representative accomplishments at this level of maturity are as follows:

- The organization has the means to identify weaknesses and strengthen the process proactively, with the goal of preventing the occurrence of defects.
- Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.
- Data on software process effectiveness is used to perform cost benefit analyses of new technologies and proposed changes to the organization's software process.

- Innovations that exploit the best software engineering practices are identified and transferred throughout the organization.
- Software project teams analyze defects to determine their root causes.
- A project’s software process is evaluated to prevent reuse of any deficiencies that are recognized.
- Lessons learned during a project are disseminated to other projects.
- Improvement occurs both by incremental advancements in the existing process and by innovations using new technologies and methods.

For a process to be rated at this level of maturity, it must be producing results that satisfy the nine goals (in three Key Process Areas) listed in Table 10.

Table 10 Key Process Areas and Goals for Maturity Level 5 (Stage 7)

16. Defect Prevention (KPA 16) Defect prevention activities are planned. Common causes of defects are sought out and identified. Common causes are also prioritized and systematically eliminated.
17. Technology Change Management (KPA 17) Incorporation of technology changes are planned. Effects of new technologies on quality/productivity are evaluated. Appropriate new technologies are put into use across the organization.
18. Process Change Management (KPA 18) Continuous process improvement is planned. Participation in the organization’s SPI activities is organization-wide. The organization’s standard and defined software are improved continuously.

4.8 Recap and Preview

A company that continuously improves software quality has changed its organizational culture. Its software process is established organization-wide and is owned and managed by its senior management sponsor. The chief change agent inspires, drives, and coordinates the ongoing work of improvement and adaptation to change. Software engineers throughout the organization are empowered within the “continuous improvement cycle” to propose and evaluate change, deploying and institutionalizing those changes that have merit. Changes made to the software process are evaluated in terms of their impact on the quality indicators selected. Improvement goals are expressed in terms of these quantitative indicators, and managers and staff are both incentivised to achieve the goals that are set.

Appendix C shows how the progress suppliers make toward continuous software quality improvement is being reported.

A supplier has full responsibility for the quality of all software delivered to its customers, but customers influence the level of quality achieved. Although a supplier must establish its own software process, customers and the SPI project at SEMATECH have resources that can accelerate progress and increase the success that is achieved. Section 5 discusses these matters in more detail.

5 USE “PARTNERING” TO REDUCE RISK AND SPEED PROGRESS

From the recent book *Leading Change*:

Available evidence shows that most public and private organizations can be significantly improved, at an acceptable cost, but that we often make terrible mistakes when we try because history has simply not prepared us for transformational challenges. [12]

The author continues with the observation that “a globalized economy is creating both more hazards and more opportunities for everyone, forcing firms to make dramatic improvements not only to compete and prosper, but also to merely survive.”

As to errors, the author says: “By far the biggest mistake people make when trying to change organizations is to plunge ahead without establishing a high enough sense of urgency in fellow managers and employees. This error is fatal because transformations always fail to achieve their objectives when complacency levels are high.”

Actually, this is the first of eight common mistakes cited. The other seven are these:

- Failing to create a sufficiently powerful guiding coalition
- Underestimating the power of vision
- Under-communicating the vision by a factor of 10 (or 100 or even 1,000)
- Permitting obstacles to block the new vision
- Failing to create short-term wins
- Declaring victory too soon
- Neglecting to anchor changes firmly in the corporate culture

Throughout, this guidelines document communicates a vision for software and an approach to improving its quality that avoids these misakes. The vision cannot be static because software engineering is still a maturing discipline [13]. For that reason, the SPI Project maintains a working relationship with the SEI, an organization on the forefront of change in this area. For example, the SEI’s CMM for Software, a key component of the SPI Project’s software vision, gives all SEMATECH/SPI participants access to a continuing stream of “software best practices.”

Secondly, improving software as prescribed by the CMM and outlined in this document involves serious risks that cause some to stumble or even fail. In a competitive industry like this one, companies are sometimes willing to share “lessons learned” but are unlikely to openly discuss their mistakes and failures. Even so, one can listen to those who observe and report on change initiatives, determining what risks and pitfalls are common and most serious. Following is a representative list of such pitfalls [14], each with a summary statement about managing the risk in a way that is consistent with the SEMATECH/SPI approach:

- **Lack of management commitment.** Be sure commitments from management translate into resources for SPI, realistic expectations, and accountability for results.
- **Unrealistic management expectations.** Educate managers to understand the realities of what serious process improvement costs, what benefits to expect, and how long it takes.
- **Time-stingy project leaders.** Senior management must make it clear that project leaders will be evaluated on the effectiveness of their process improvement activities, as well as on the success of the software projects themselves.

- **Inadequate time and resources devoted to training.** Inadequate knowledge leads to false starts, well-intentioned but misdirected efforts, and a lack of progress. Therefore, provide training in the improvement framework you select (e.g., CMM), setting up a software engineering function, establishing a metrics program, assessing the process capability of a project team, and action planning.
- **Making achievement of a CMM maturity level the primary goal.** Make sure your software process improvement effort is aligned with corporate business and technical objectives, and is meshed with other improvement efforts, e.g., ISO registration.
- **Failing to scale formal processes to project size.** Know the types and sizes of your software projects, and rationally scale the practices recommended by your improvement framework (e.g., CMM) to the size and nature of your projects.
- **Excluding project team members from assessments.** Process change involves cultural change, and the ease/success of making the change happen depends on the extent of staff involvement with process assessments and change planning activities.
- **Stalling on action plan implementation.** Plans turn into actions when each effort becomes a mini-project, you concentrate on just two or three areas at a time to avoid overwhelming the staff, and then report status at quarterly management reviews.
- **Turning process improvement into a game.** The idea that SPI is a fad is dispelled if the process change leader illustrates behaviors expected in each improvement area as superior processes are successfully implemented, internalized, and institutionalized; and the manager makes clear he/she is serious about continuously improving the way software is built—that methods, when replaced, are gone for good.
- **Expecting defined procedures to make people interchangeable.** Process improvements do not equalize the large difference that exists in the capabilities of software engineers. Instead, aim for creating an environment in which all team members share a commitment to quality and are enabled—through superior processes, appropriate tools, and effective team interactions—to each reach their own peak performance.

A final feature of the SEMATECH/SPI approach is the teamwork, sharing, and trust-building among participants that it fosters. The lists of common mistakes and pitfalls cited above underscore the importance of such teamwork, certainly within an organization, but also involving other stakeholders—the customers, in some cases suppliers, and often one or more SEMATECH programs. This is essential because suppliers and manufacturers must each contribute to software improvement. For example, suppliers produce the reliable software manufacturers need, but to produce it suppliers need a complete statement of requirements from the manufacturers, a vast amount of information about fab environments, and actual data from equipment performance and malfunctions. No single group can deal with the software challenge single-handedly; meeting it successfully means all parties are working together. The benefits of success, sketched in Appendix D, show that the returns available are considerable—for both suppliers and manufacturers.

The balance of this section outlines the teaming that has emerged. It is beyond this document's scope to get into details, but the highlights clearly show the nature of the program in place.

5.1 Role of a SEMATECH Member Company

The cost of semiconductor fabs increased sharply in recent years. Gordon Moore says it will be \$2 billion in the near term, and well beyond that within ten years [15]. Equipment for handling, processing, and testing wafers accounts for more than 70% of a fab's cost. During recent years, embedded software has become a significant factor in the cost of such equipment (in some cases 50% or more) and projections show software's share of the cost continuing to increase. The software and communications networks that are created to plan, manage, and analyze fab operations pushes the capital that relates to software still higher. And then there are operating costs tied to software, such as the ramp time needed to bring up a new fab or to reconfigure a production line for a new product, the COO for equipment in general, and OEE during ongoing production. Work stoppages and product breakages due to software malfunctions, plus the time and effort required for recovery—these with the other costs mentioned put a manufacturer's total software bill into some very large numbers.

To gain the strategic advantage that very reliable, high-quality software offers, SEMATECH members are doing, and must do more of, the following:

- Deploy (with other SEMATECH members) the Software Quality Improvement (SQI) Policy and a roadmap for complying with it.
- Incorporate the SQI Policy into your equipment procurement operations, making it the basis for evaluating a supplier's product quality and software capability before buying.
- Use the SQI Policy's metrics (e.g., SSQA results) to track the SPI progress your key suppliers make improving their products and software process.
- Be software-literate and aware of software quality improvement, and expect the same of the senior managers in your supplier companies.
- Put high priority on software issues in the 1997 *National Technology Roadmap for Semiconductors*.
- Form strategic partnerships with key suppliers and focus your software improvement efforts with them on specific tools, both current and future.
- Change the measurements captured in fab operations to include MWBI and/or MTBF.
- Improve the practices used in fabs to capture and analyze failure data so that it recognizes and labels software failures.
- Define (with the SPI Project) and implement software failure reason codes, and do a Pareto analysis of the data you collect.
- Dedicate staff with software expertise to pursuing SPI in the fabs and by suppliers.
- Normalize your product requirements to reduce specials and customization.
- Apply the SQI Policy within your own operations that produce software for the fabs.
- Work with and continue to support the SPI Project to accomplish the above.

5.2 Role of a Supplier

A reputation for sustained excellence in software is increasingly important for suppliers in this industry. Product functionality gives competitive advantage, and more than ever that functionality is based on software. In many cases, the software staff now outnumbers all other engineers, and

software costs are an increasing share of overall product development cost. Building your company's reputation in software calls for blending quality principles with sound business practices. Four guidelines [16] are important:

- Provide the greatest possible *value to customers* by providing product functionality that is responsive to their needs, expectations, and definition of quality.
- Focus on *preventing software problems*, not on firefighting to correct problems.
- *Measure progress* toward software quality goals using established metrics, objective benchmarks, and appropriate standards.
- *Continually improve* software skills, processes, technologies, products, and services.

Adhering to these guidelines—as you change your company's culture to one that learns, changes, and matures systematically—requires a senior management sponsor and guide, and widespread buy-in from the software practitioners. Many suppliers are seeking that accomplishment by taking the following actions:

- Subscribe to the Policy for Software Quality Improvement and commit to complying with all aspects of it.
- Implement the SEMATECH/SPI approach to software quality improvement within your organization.
- Track the progress you make maturing your software process and improving the quality of your software, beginning with the 4-ups metrics in the SQI Policy.
- Designate and charter a senior manager as the sponsor and owner of your initiative to improve software, with accountability and rewards for success.
- Designate and charter a software practitioner as the champion and chief change agent for SPI, with accountability and rewards for success.
- Work with customers to be clear about their meaning of software quality and defects, and then focus your improvement initiative on providing quality as they define it.
- Set improvement goals (e.g., 25% per year reduction in customer-discovered defects) and then take actions that achieve your goals.
- Provide incentives for and give recognition to project managers who improve software quality.

A software organization that is isolated from the rest of product development or that is out of management's control can produce software, but odds are that its cost is high and its quality low. High quality, cost-controlled software results from teamwork within your organization, as well as between your organization and others. And that, in simplest terms, is what makes “process” so important. Without it, the teamwork that is required just cannot happen.

5.3 Role of SEMI/SEMATECH

SEMI/SEMATECH's mission is to achieve continuous improvement among U.S. users and suppliers of semiconductor manufacturing equipment, processing materials, and software. Under this mission, its objectives are as follows:

- To work collectively with the membership's customers for mutual benefit
- To facilitate continuous improvement of the members to achieve world-class competitiveness

- To work with SEMATECH on its programs and deliver effective program-related communications to its members

In addressing the industry's software challenge, SEMI/SEMATECH's role is presenting clearly to its members the importance of software reliability and overall quality, the realities of what it takes to deliver high quality in software, and the benefits available to those who successfully confront the software challenge. SEMI/SEMATECH is encouraging its members' involvement in SPI in many ways:

- Promoting the adoption and use of the SQI Policy and its metrics, and the SPI roadmap
- Challenging SEMI/SEMATECH member companies to make a substantial commitment to SPI
- Encouraging SEMI/SEMATECH members to have an SPI sponsor (senior manager) and chief change agent (senior software practitioner)
- Working with SEMATECH, SEMATECH members, and SEMI/SEMATECH members to identify software issues for the 1997 *National Technology Roadmap for Semiconductors* and give those issues high priority
- Cooperating with the SEMATECH SPI Project to help meet the software challenge

5.4 Role of SEMATECH's SPI Project

The SEMATECH member companies have chartered the SPI Project to provide leadership and to facilitate networking and cooperation that support software improvement in companies that are members of SEMI/SEMATECH. Table 11 gives the project's mission and objectives. These are supported by the project's key activities, which are:

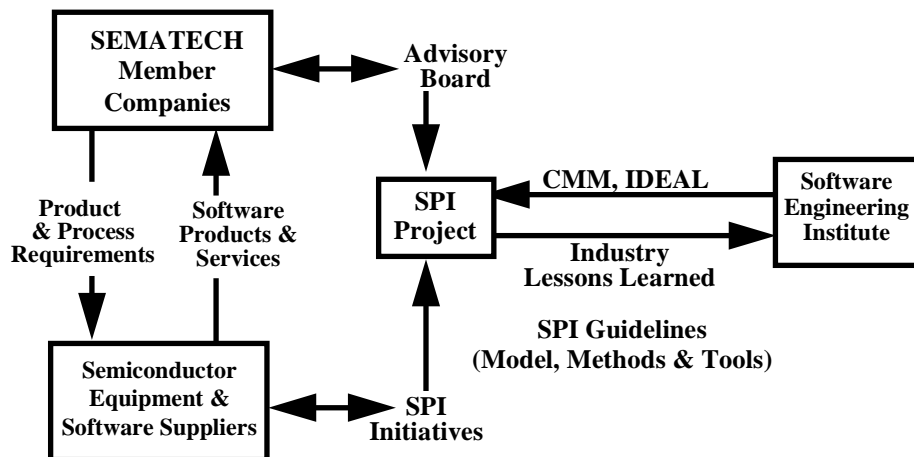
- Articulate the industry's software issues.
- Draft a Policy on Software Quality Improvement and coordinate its review and adoption by the members of both SEMATECH and SEMI/SEMATECH.
- Develop a consensus on supplier metrics and milestones for SPI.
- Deploy a roadmap to software reliability and overall quality for the industry that is based on the policy, its metrics and milestones.
- Define an approach to capturing fab software failure data.
- Define software-failure-reason codes.
- Monitor the progress suppliers make improving software quality in their products and their software processes.
- Periodically baseline the industry for progress made in process and product improvement.
- Facilitate cooperation in SPI activities between the member companies and suppliers.

Table 11 Mission and Objectives of the SPI Project

The SPI Project Mission	
Increase the operational reliability of SEMATECH member company fabs by improving software acquired from suppliers.	
SPI Project Objectives	
Member Companies	Suppliers
<ul style="list-style-type: none"> • Reduce cost of ownership for fab equipment 	<ul style="list-style-type: none"> • Increase software MWBI in delivered equipment
<ul style="list-style-type: none"> • Increase reliability of fab operations 	<ul style="list-style-type: none"> • Reduce the number of defects in delivered software
<ul style="list-style-type: none"> • Require software improvements in acquisition process for supplier procurements 	<ul style="list-style-type: none"> • Improve profitability by reducing the cost of software development and maintenance
	<ul style="list-style-type: none"> • Improve schedule adherence for delivered software

Figure 7 shows the context in which the project works by identifying its major relationships. From this position, the project provides selected forms of on-site technical support and assistance to suppliers, together with management reviews, using the following resources and events:

- Training courses in various aspects of software improvement (east and west coast)
- Guidelines for improving software (published annually)
- Case studies of significant achievements in improving software (one or two a year)
- Workshops that foster software improvement (twice a year)
- A process asset library (electronic form, paper form, World Wide Web)
- Resources from R&D centers working on software improvement (SEI and others)
- Information about consultants with relevant SPI experience

**Figure 7 Key Relationships of the SPI Project**

The project sponsors training courses on important software engineering topics, such as software inspections and software project management. These courses are taught at both West and East Coast locations.

The project conducts semiannual supplier workshops, giving suppliers an opportunity to share lessons learned in software process improvement and hear of best practices developed throughout the membership.

The project publishes what-to-do and how-to materials, together with important results of software process improvement activities in the semiconductor industry and elsewhere.

The training courses, workshops, and case studies all offer opportunities for networking among the supplier companies. By contributing to the process asset library, you and other contributors share documents, checklists, spreadsheets, and similar tools, making it unnecessary to develop each from scratch. The SPI Project also provides information about conferences, public training courses and workshops, and local groups that promote software improvement.

If you have questions about any aspect of how to improve the software you deliver to your customers, or want to discuss a working relationship with the SPI Project, please contact:

Harvey Wohlwend
SEMATECH
2706 Montopolis Drive
Austin, Texas 78741

Phone: 512/356-7536
Fax: 512/356-3575
E-Mail: harvey.wohlwend@sematech.org

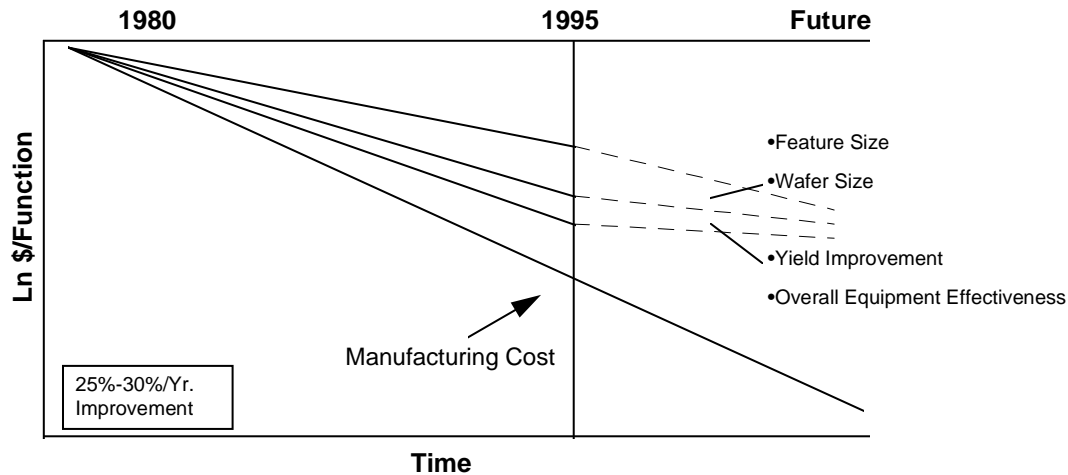
5.5 A Final Word

For three decades now, the number of transistors that engineers squeeze onto a silicon chip has doubled with machine-like regularity every 18 months. This cramming of ever more transistors on a polished slice of sand makes chips more powerful, computers and other devices made from the chips more versatile, and the range of applications for semiconductor devices more extensive [15].

Figure 8 shows that this phenomenon, known as the industry's cost-reduction curve, depends on four contributing factors [1]. To keep on its downward path, two of the four factors—decreasing feature size and increasing overall equipment effectiveness—must provide an increasing share of the 25%-30% per year improvement. That is because the other two factors are approaching limits. The two factors that must increase rely in part on improved software quality for that increase. That means the increase can only be realized through software process improvement.

Over the years, the semiconductor industry has become famous for addressing challenges from the hard sciences—physics, chemistry, and electronics—and for solutions worked out in hardware. But now, the challenge of providing high quality software comes from a different direction, one involving human factors more than science and hardware. Three keys to success in this case are

1. Having the will to do it and persevering
2. Getting and applying the experience and techniques needed to do it
3. Being diligent about communications among the participants while doing it



Factor	1980	1995	Future
Shrinking feature sizes	12%	12–14%	12–14%
Larger wafer sizes	8%	4%	<2%
Yield improvements	5%	2%	<1%
Overall equipment effectiveness	3%	7–10%	>9–15%

Figure 8 Keeping the Productivity Engine on Track

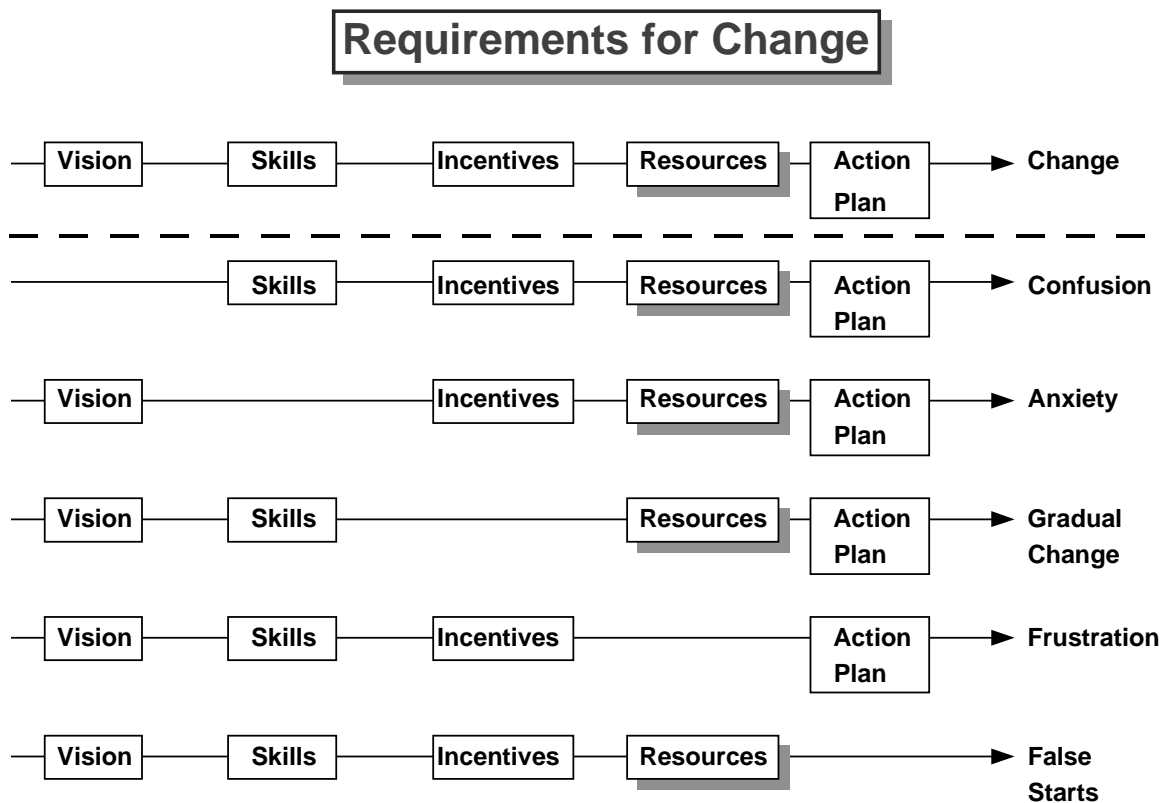
The different nature of this challenge is quite apparent when you look at the reasons organizations fail to get benefit from software improvement initiatives. The top ten reasons reported by one consultant familiar with the industry [17] are the following:

1. No consensus on the problem and the need to change
2. Lack of education on the issues, e.g., what is process, maturity, software, software engineering? Why care?
3. Lack of the leadership, commitment, and resolve that successful long-term change requires
4. Attitude and priority problems (e.g., fear of and resistance to change)
5. Lack of culture-change skills
6. Lack of vision and focused objectives
7. No plan of action for establishing an improvement roadmap
8. Failure to measure benefits and provide feedback
9. Selection of the wrong maturity model, or incorrectly using the selected model
10. Organizational listening and learning disabilities

These success blockers are all human factors—challenging, but very different from the challenge of producing 0.18 μm circuits.

Figure 9 illustrates this point another way. It shows the five requirements for organizational change—all human-factors issues—and the outcomes to expect when any one of the five is missing. This document supports the “vision” that is needed and provides an “action plan” framework. The “skills” needed to improve software quality are readily available externally and can be developed among your own staff. That leaves “incentives” and “resources,” which each organization must provide—or deal with outcomes that result when these elements are missing.

The SPI Project predicts that high software quality, and the capability required to produce it, will be major supplier differentiators before the end of this decade.



Adopted from Ambrose 1987

Figure 9 Five Requirements for Organizational Change

6 REFERENCES

- [1] J. Owens, *The Productivity Challenge*, SEMATECH, Austin, TX, a paper presented at SEMICON/West, July 1995

The following excerpt from this presentation is most relevant: “Historically, four major factors have contributed to the economics which allow the semiconductor industry to maintain its productivity: shrinking feature sizes, adopting larger wafer sizes, improving yields, and improving other productivity factors, which are primarily equipment productivity improvements. ... My conclusion after doing this study is that equipment productivity needs to become the star of this industry.” Equipment reliability is a major factor in equipment productivity, and software reliability is, of course, a significant component of overall product reliability. Software reliability can be dependably increased only by the type of software process and quality improvement initiative outlined in this document.

- [2] H. Wohlwend, et al., *Software Issues, Practices and Reliability Within the Semiconductor Industry: A SEMATECH White Paper*, Technology Transfer #96033106A-TR, SEMATECH, Austin, TX, May 31, 1996 (SEMATECH Confidential)
- [3] SEI’s study of 13 companies is reported in the document:

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

Additional information on 48 companies is reported in the document:

W. Hayes, D. Zubrow, *Moving On Up: Data and Experience Doing CMM-Based Process Improvement*, Software Engineering Institute, Pittsburgh, PA, CMU/SEI-95-TR-008

- [4] *Process Maturity Profile of the Software Community, 1996 Update*, Software Engineering Measurement and Analysis Team, Software Engineering Institute, Pittsburgh, PA, April 1996
- [5] Watts Humphrey is the originator of the statement that software quality is a function of the maturity of the process producing it. The statement was based on observation and anecdotal data, and since it was first made many informal reports from individual companies have supported it. More recently, some disciplined studies have been completed, confirming it. One such study is:

D. Goldenson, J. Herbsleb, *After the Appraisal: A Systematic Survey of Process Improvement, Its Benefits, and Factors that Influence Success*, Software Engineering Institute, Pittsburgh, PA, CMU/SEI-95-TR-009, August 1995

- [6] J. Johnson, *CHAOS: The Dollar Drain of IT Project Failures*, Application Development Trends, January 1995, pp 41-47

- [7] Following are references for the Capability Maturity Model for Software from the Software Engineering Institute, the SPI Project’s primary model of software capability:
- 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

Material from the following handbook on the CMM for Software used in the discussion of the CMM in Section 3, for example the profiles of the maturity levels in Table 2:

K. Dymond, *A Guide to the CMM*, Process Inc US, Annapolis, MD, 1995

Following are references to the other two models of software capability referred to in this document:

ISO 9000-3, Guidelines for the Application of ISO 9001 to the Development, Supply, and Maintenance of Software, International Organization for Standardization, June 1991

Standardized Supplier Quality Assessment Workbook, SEMATECH non-Technology Transfer document, January 15, 1996

Standardized Supplier Quality Assessment Training Participant Guide, SEMATECH non-Technology Transfer document, May 27, 1996

- [8] *SEI Strategic Plan: 1997-2001*, Software Engineering Institute, Pittsburgh, PA, CMU/SEI-96-SR-006, August 1996
- [9] S. Fulton, M. Messer, *Tactical Software Reliability Guidebook*, Technology Transfer #95092967A-GEN, SEMATECH, Austin, TX.
- [10] The Software Engineering Process Group of the Raytheon Equipment Division was the 1995 winner of the IEEE Computer Society Software Process Achievement Award. This award recognizes outstanding achievements in software process improvement. The report cited here was written to document the achievements recognized by the award. Material from the report contributed to the Improvement Functions listed in Section 3, and to the organizational structure depicted in Appendix B.
T. Haley, B. Ireland, E. Wojtaszek, D. Nash, R. Dion, *Raytheon Electronic Systems Experience in Software Process Improvement*, Raytheon Electronic Systems, Sudbury, MA, and Software Engineering Institute, Pittsburgh, PA, CMU/SEI-95-TR-017, November 1995
- [11] E. Quann, *Transforming the Workplace into a Learning Organization*, Fastrak Training, Inc., Columbia, MD, presented at the Software Engineering Process Group Conference in Boston, MA, May 1995, and published in *Software Quality Matters*, Vol. 4, No. 1, Spring 1996
- [12] J. Kotter, *Leading Change*, Harvard Business School Press, Boston, MA, 1996
- [13] W. Gibbs, *Software's Chronic Crisis*, *Scientific American*, September, 1994
- [14] K. Wiegers, "Software Process Improvement: 10 Traps to Avoid," *Software Development*, May 1996, pp 51-58

Karl Wiegers is a software process improvement leader in a large product software division at Eastman Kodak Co. He is the author of *Creating a Software Engineering Culture*, a book being published by Dorset House Publishing in 1996

- [15] P. Ross, *Moore's Second Law*, Forbes, March 25, 1995, pp 116-117
- [16] H. Krasner, *Imagining Life in a Mature Software Organization*, Krasner Consulting, Austin, TX, Software Quality Matters, Vol. 4, No. 1, Spring 1996
- [17] H. Krasner, *Why Organizations Fail to Benefit from Software Improvement Initiatives*, Krasner Consulting, Austin, TX, a presentation at the Austin Software Process Improvement Network (SPIN) Meeting on April 18, 1996

Herb Krasner has been involved in software process improvement activities within the semiconductor industry for several years, working with manufacturers as well as supplier companies.

- [18] *SEMATECH Cost of Ownership (COO) Model*, Technology Transfer #91020473B-GEN, January 24, 1992, SEMATECH, Austin, TX.

APPENDIX A

Complying with the Policy on Software Quality Improvement

In Figure 1, the Policy on Software Quality Improvement lists four “compliance” metrics (see Item 7). Those suppliers who regularly take the measurements the metrics prescribe, and use the data to drive quality improvement in their organizations, are in “basic compliance” with the policy.

The Policy also states that these four metrics plus some additional data—Assessment Findings (Item 1); Quality Goals and Action Plan (Item 2); and Test Plan, Test Report, Release Notes, and Standards Compliance Status (Item 6)—are the “indicators of quality” selected for communicating with customers. A supplier making all of these indicators available to customers (in addition to using them to drive quality improvement) is “generally compliant” with the policy.

To be “totally compliant,” a supplier develops and uses “objective indicators of quality” for *every* state, activity, and accomplishment cited in the policy. Compliance in this broadest sense means creating and using quality indicators for each subject listed below:

<u>Item</u>	<u>Subject</u>
1(a).	Commitment
1(b).	Models of Maturity
2.	Goals, Plans, and Metrics—Continuous Improvement
3.	Problem and Corrective Action Reporting System
4.	Software Revision Control System
5.	Staff and Budget for Quality Improvement Functions
6.	Best Practices and Standards for Software Quality
7.	Compliance Metrics

“Total compliance”—where the supplier has a mature software process, consistently follows it, and is proactively improving the quality of all its software—is the intent of the policy.

This appendix shows the scope of a mature software process to be quite broad by giving examples of quality indicators over the complete range of subjects the policy addresses. After sketching all eight subjects, the examples are repeated as a single list to illustrate the flow of results it takes to drive a mature software process. Section 3 pictures a mature software process in terms of organizational functions—the functions that produce the flow of results presented here. Sections 4 and 5 offer guidelines for putting the various quality functions into play. Once they are operational, your software process is mature and your organization is “totally compliant” with the policy.

Item 1a: Commitment. A commitment to improve software quality must be substantive, specific, focused ... something like the following:

“I” and “my organization” will produce and deliver better and better software over time, based on verifiable metrics, and *will do whatever it takes* to make that happen in ways that measurably benefit our customers and our own company.

A commitment that is vague, generic, or generalized is not a commitment at all.

The actual wording in Item 1 of the Policy is: “Establish an organizational commitment to improving software quality ... “ An “organizational commitment” implies that each major participant in the software process:

- Sees the need for improvement and its benefit potential
- Shares an understanding of how to go about doing it
- Is committed to personal and active participation
- Is prepared to perform their role competently
- Has motives/incentives for making a success of the effort

The major players include at least the following:

- Senior managers
- Software staff (managers, developers, maintenance/support people, field representatives)
- Staff that interfaces with customers in matters that impact software
- Engineers responsible for aspects of the product other than software

Two essentials in an “organizational commitment” are the active involvement of senior management, and teamwork among all participants. The commitment and spirit of the team must be cultivated and reinforced by repeated successes. “Learn to crawl and walk before you try to run or fly” is advice applicable to this effort.

Examples of “indicators” that show an organization is committed are the following:

- Connections that managers make between improving software quality and reaching the goals set for the business.
- A public statement that commits the organization to improving the quality of its software.
- Authorizations for staff time and budgets to improve software quality.
- Records of “training in quality improvement methods” for various participants in the software process.
- Established incentives/rewards/recognition for success in software improvement.
- Evidence of a connection between the organization and the SPI Project at SEMATECH.

Item 1b: Models of Maturity. What is required to produce and deliver software with higher and higher quality is well known:

- A mature software process
- Capable software engineers
- Proactive and involved management of the software process

But it is not widely practiced because

- Most organizations have an immature software process
- Few staffs are trained/experienced in quality/continuous-improvement practices
- Middle and senior managers are often not actively involved with software

A model of process-maturity-and-staff-capability can be invaluable as a resource that informs an organization's commitment-to-improve.

The SPI Project recommends two maturity/capability models: SSQA Module 3 and the SEI's CMM for Software. ISO 9001 (with 9000-3 for software) is compatible with both, and will begin moving an organization toward the SSQA and the CMM goals.

Item 1 of the policy directs suppliers to use findings from SSQA ratings and/or SEI maturity levels to drive software quality improvements. To improve software quality, a "maturity model" has two roles. First, it is the basis for showing how an organization measures up as a producer of software. A detailed and objective assessment of actual practices determines the organization's position along the model's maturity/capability continuum. The quantified summary of assessment findings is a score or rating. The findings themselves are the basis for the model's second role, which is to indicate next steps for maturing the software process, increasing staff capabilities, and enhancing the practices of those managing the software process—thereby improving software quality.

To use a maturity model with effectiveness, key staff members must understand it, so training is required. Objectivity in the assessment is crucial, so steps that ensure a comprehensive and unbiased view of the organization are important. Analysis and interpretation of assessment findings provide the insight and understanding needed to prioritize and implement the improvements the model suggests. Implementing change can vary from simple actions (like a decision to begin doing something or to do it differently), to executing improvement projects that take planning and managing like any other kind of project.

"Indicators" of an organization's effective use of one or both models include the following:

- An organization-wide announcement identifying the maturity model(s) selected
- Names of assessment team members with qualifications and training provided
- Organization's profile as a software producer, SSQA-based and/or CMM-based
- The report to staff and managers of assessment findings and recommendations
- The organization's product-delivery records, showing adherence to schedules

Item 2: Goals, Plans, and Metrics—Continuous Improvement. Commitment and maturity models (Item 1) are means to an end. The *end itself* is knowing how your customer defines software quality, measuring the level of quality (as defined by your customer) in your existing products and releases, and producing a software release with higher quality than the release it replaces—or a new product with higher quality software than any of your existing products. *That* is product quality: quality that is *improving*.

The policy states, "Demonstrate continuous improvement by adopting ... goals and ... plans that achieve the goals." The policy also cites two SEMATECH member company goals:

- Operational reliability, measured by how many "units of productive work" are completed between "failures" of the software
- Freedom from defects, measured by counts of "defects" detected and removed, balanced by "known defects" not yet removed plus an estimate of "defects yet to be detected"

For equipment that processes wafers, MWBI is the preferred measure of "units of productive work" between software failures. The reliability of software in other kinds of equipment and

systems is measured by MTBF. In both cases it is important to have clearly defined meanings for “interrupt” and for “failure,” and a discipline for determining that software caused the interrupt or failure.

Improvement action plans are based on the findings of an assessment of your software process. Staff and budget must be committed to complete the assessment and develop the plans (and a high priority given to plan execution), or nothing significant will be accomplished.

Some “objective indicators” of continuously improving software quality include the following:

- A customer-validated definition of software quality and defect
- A measurement (or estimate) of software reliability for each of your current products (MWBI/MTBF)
- A trend chart (for sequences of releases) that shows software quality (as defined by your customers) is increasing
- A plan for implementing a change in the software process that will improve software quality

Item 3: Problem and Corrective Action Reporting System. One way to continuously improve software quality is a systematic approach that tracks to completion each reported problem and proposed improvement. Initially, this system may deal exclusively with product defects, then include product enhancements, and finally handle fixes and improvements to the software process itself. In this way, a supplier moves from an initial focus on product defect *detection and correction*, to defect *prevention*. Trying to improve software quality without a disciplined reporting and tracking system is a waste of time, effort, and resources.

Item 3 states, “Establish and maintain a (software) Problem and Corrective Action Reporting System ...” Such a system determines how critical each reported problem is, assigns priorities and selects the tasks on which to work, tracks each task to completion, determines when a product is ready for release to customers, and analyzes the software process to identify areas that need improvement.

Some “indicators” that show a problem and corrective action reporting system exists and is used effectively are:

- Meeting announcements and minutes of the group resolving the problems reported
- Existence of a software problem report form
- The current set of software problem reports, with criticality ratings and resolution priorities
- Trend charts showing problems reported, problems resolved, problems yet to be resolved

Item 4: Software Revision Control System. The policy advocates continuous improvement in the sense that each new module of software developed has higher quality than those developed before it. But it also promotes improving existing modules of software by making defect repairs and implementing enhancements that evolve the software both functionally and qualitatively.

A software revision control system, by performing configuration management for software engineers, supports both types of software improvement. It also coordinates access to software modules during development (multiperson projects and developer/tester coordination), and records the exact configuration of software distributed to each customer. This is important when a defect is reported in software that has many different configuration options and/or has been

customized for the particular customer. Several commercial products on the market offer a range of options and prices.

Some “indicators” that show a Software Revision Control System exists and is used effectively are as follows:

- The standards that dictate what materials the organization puts under configuration management
- The procedures for software engineers to use when accessing materials that are under configuration management
- Standards for determining when a software product is released and ready for distribution to customers
- Records that show exactly what software has been distributed to (is installed at) each customer site
- The name of the commercial product selected, installed, and used for configuration management

Item 5: Staff and Budget for Quality Improvement Functions. A commitment to quality—even one that’s informed by a maturity model—accomplishes nothing without the following:

- A *vision* for software in the particular organization
- *People* who are chartered, funded, and accountable for implementing the vision

Two functions are key:

- The SPI function provides a vision, and then plans/champions an initiative that realizes the vision, also serving as chief change agent for making the changes required.
- The SQA function validates results and processes, ensuring that products meet the organization’s quality standards and that the organization’s quality practices are used consistently.

Providing both functions in an appropriate form is important. In smaller settings, the staff will be part-time; in larger ones, staff will be full-time members of established groups.

Item 5 declares, “Charter staff and allocate budget to the work of continuously improving ... and ... validating ...” Typical SPI responsibilities are as follows:

- To visualize, launch, sustain, and report the accomplishments of an initiative that continuously improves software quality, process maturity, and staff capability
- To achieve consistent use of a mature, established software process—one that applies to all types of projects and throughout the software product life cycle
- To identify and meet training needs of staff participating in the software process
- To examine the staff’s motivations for producing high quality software, and to champion the use of additional incentives and rewards as appropriate
- To evaluate the methods, tools, and platforms the organization uses to produce software, and then propose, evaluate, and inaugurate improvements as appropriate
- To champion fact-based planning/decision-making throughout the organization by leading in the use of metrics and measures in all aspects of software improvement

- To enable key support functions within the software process, such as peer reviews, configuration management, software quality assurance, and pathfinding

Typical responsibilities of an SQA function include the following:

- To objectively validate that software procedures the organization uses correspond to the standards and procedures that have been established
- To objectively assure the quality of each product delivered to a customer

Some “indicators” that show the two functions exist and are effective include the following:

- Management-approved charters for both functions
- Staffing rosters for both with time allocated versus actual time spent
- Actual expenditures for improvement versus the budgets allocated
- Organization-wide announcements of actions by each of the quality functions

Item 6: Best Practices and Standards for Software Quality. The ultimate test of software quality comes when the customer uses it. Data delivered with software serve as indicators of quality.

Item 6 of the policy directs suppliers to indicate what level of quality they are delivering. The items named in the policy relate to testing, defects, and standards—three areas that are keys to producing quality software. Using best-known-practices in all three areas leads to high quality.

The data selected to be “indicators” of quality are as follows:

- Software test plan and test report (results from execution of the test plan)
- Documentation (release notes) for all changes made since the last release
- List of the known defects with a procedure for recovering from the problem
- Status of compliance with the current interface standards

Item 7: Compliance Metrics. All quality indicators identified in these notes help drive a supplier’s initiative to improve software quality. But SEMATECH and its member companies have selected four metrics as the key indicators: two are direct measurements of software products, and two measure the software process a supplier uses. Positive trends in all four serve as first indicators of progress toward software that is more reliable and in other ways higher in quality.

Item 7 of the policy states, “Show compliance with this policy by using the following metrics.” Metrics specified are the following:

- Software quality or the absence of defects in software delivered to customers
- Software reliability or the time between occurrences of downtime due to a software problem
- On-time delivery of software and overall consistency meeting commitments
- Software process maturity as determined using an approved model of maturity

A.1 Examples of Quality Indicators: All Eight Subjects

Item 1a— Commitment

Evidence that an organization is committed to improve software quality:

1. Connections (by management) between software improvement and the organization's business goals
2. A statement of policy that commits the organization to improving the quality of its software
3. Authorizations for staff time and budgets to improve software quality
4. Records of "improvement training" for various software process participants
5. Established incentives/rewards/recognition for those responsible for success in software improvement
6. Explicit connections between the organization and the SPI Project at SEMATECH

Item 1b— Models of Maturity

Evidence the organization has selected and is using SSQA and/or CMM as a software maturity model:

1. An organization-wide announcement that identifies the maturity model (or models) selected
2. A list of the assessment team members with their names, qualifications, and roles
3. **The organization's profile as a software producer, based on an SSQA- or CMM-based assessment¹**
4. The full report to staff and management of assessment findings and recommendations
5. **The organization's product delivery record, showing schedule adherence as on-time, early, or late**

Item 2—Goals, Plans, and Metrics—Continuous Improvement

Evidence the organization is continuously improving the quality of its software:

1. Customer-validated definitions of software quality and software defect
2. A measurement of software reliability for current products and software releases (MWBI/MTBF)
3. A trend chart (for a sequence of releases) that shows increasing software quality
4. Quality improvement goals with metrics for measuring progress to the goals
5. Action plans for improving software quality, with schedules and recent accomplishments

Item 3—Problem and Corrective Action Reporting System

Evidence the organization is reporting product/software problems and tracking corrective actions:

1. Charter, membership list, meeting schedule, and minutes of the steering committee for product change control.
2. Forms and procedures for reporting and prioritizing problems, and for tracking the corrective actions taken.

¹ Examples in **boldface** are explicitly mentioned in the policy statement.

3. Current list of known product/software defects with priorities/schedules for correcting them.
4. List of recently completed actions taken to correct software defects in the product.

Item 4— Software Revision Control System

Evidence the organization is managing access to and the distribution of software:

1. Name of the commercial product selected, installed, and used for configuration management
2. Standards for what materials the organization puts under configuration management
3. Procedures for accessing materials under configuration management or submitting materials for such management
4. Standards for determining when to release a software product for distribution to customers
5. Records that show exactly what software configuration has been distributed to (installed at) each customer site

Item 5—Staff and Budget for Quality Improvement Functions

Evidence the organization has committed significant resources to work on software improvement functions:

1. Management-approved charters for organization-wide improvement and objective verification/validation
2. Staffing rosters for the improvement functions established, with time allocated versus actual time spent
3. Actual expenditures for improvement, versus the budgets allocated
4. Organization-wide announcements of and results of work done by the established quality functions

Item 6—Best Practices and Standards for Software Quality

Evidence the organization is using some of the best practices and standards related to software quality:

1. Software test plan and test report (annotated results from execution of the test plan)
2. Release notes (annotated list of changes since last release and known defects with workarounds)
3. Status of compliance with the GEM/SEM and SECS interface standards

Item 7—Compliance Metrics (selected by SEMATECH member companies)

1. For the two product metrics: see Item 2, Examples 2 and 3.
2. For the two process metrics: see Item 1b, Examples 3 and 5.

APPENDIX B

Organizing to Support Software Quality Improvement

Section 3 identifies functions that make a software process mature. This appendix discusses two subjects related to establishing such quality functions within your organization. These include:

- What form and structure to give them
- How to staff them

A more thorough discussion of these topics is important, but would go beyond the scope of this document.

One way senior management expresses a commitment to SPI is by helping establish the necessary supporting infrastructure. When doing this, it is important to recognize that an SPI infrastructure must provide “leadership” more than “management.” The following excerpt from *Leading Change* shows what the difference is:

Management is a set of processes that can keep a complicated system of people and technology running smoothly. The most important aspects of management include planning, budgeting, organizing, staffing, controlling, and problem solving.

Leadership is a set of processes that creates organizations in the first place, or adapts them to significantly changing circumstances. Leadership defines what the future should look like, aligns people with that vision, and inspires them to make it happen despite the obstacles. ... Successful transformations (such as an SPI initiative) is 70 to 90 percent leadership and only 10 to 30 percent management. Yet for historical reasons, many organizations today don’t have much leadership. And almost everyone thinks about the problem as *managing* change. [12]

Table 12, also from *Leading Change*, brings out the difference even more clearly. So whatever organizational form and staffing pattern you adapt, be sure it is able to exercise the leadership its mission requires.

Against this background of “leadership first,” following are brief comments on each of the five categories of functions covered in Section 3 (see Figure 5).

B.1 Core Functions

The five functions in this category are staffed and performed by software engineers, or in some cases by people with an appropriate level of technical competence. Some companies choose to have a software group, while others distribute software people into the various projects, programs, product line organizations, or departments. Either approach is workable. What is most important is to properly determine the workloads and skill requirements across all five functions, and then staff to an adequate level and with the skills required by all five functions.

B.2 Core Extensions

This is not a category of “functions,” so there is no staffing to do nor organizational structure to create. However, the business settings (for the five core software functions, discussed in Section 3.4) that exist in your company might well influence the approach you take in staffing and organizing the core functions.

Table 12 Management versus Leadership

<i>Management</i>	<i>Leadership</i>
<ul style="list-style-type: none"> • Planning and budgeting: establishing detailed steps and timetables for achieving needed results, then allocating the resources needed to make it happen. • Organizing and staffing: monitoring results, identifying deviations from the plan, then planning and organizing to solve these problems. • Controlling and problem solving: monitoring results, identifying deviations from the plan, then planning and organizing to solve these problems. 	<ul style="list-style-type: none"> • Establishing direction: developing a vision of the future—often the distant future—and strategies of producing the changes needed to achieve that vision. • Aligning people: communicating direction in words and deeds to all those whose cooperation may be needed so as to influence the creation of teams and coalitions that understand the vision and strategies, and that accept their validity. • Motivating and inspiring: energizing people to overcome major political, bureaucratic, and resource barriers to change by satisfying basic, but often unfulfilled, human needs.
<ul style="list-style-type: none"> • Produces a degree of predictability and order, and has the potential to consistently produce the short-term results expected by various stakeholders. 	<ul style="list-style-type: none"> • Produces change, often to a dramatic degree, and has the potential to produce extremely useful change.

Source: From a *Force for Change: How Leadership Differs from Management* by John P. Kotter. Copyright 1990 by John P. Kotter

B.3 Support Functions

These three functions are of two kinds. The first—Problem Reporting and Change Control—collects problem reports and enhancement requests from throughout the organization into a central repository. A group of people selected to represent the interests of various parts of the organization then prioritizes and manages the work of responding to the work requests collected. This group—often called the Change Control Board (CCB)—meets periodically to review progress, make decisions about the work at hand, and set schedules. The staffing required to manage the reports and requests and otherwise support the work of the CCB depends on the volume of work and size of the organization.

The other two support functions are centralized, specialized services for the rest of the organization. The people that staff these functions perform the prescribed services for individuals and groups throughout the organization, using established procedures and, as appropriate, automated tools. Therefore, they must be proficient with the procedures and tools, have a service orientation, and be able to acquaint and coach the rest of the organization about the services they provide. Again, the level of staffing depends on the volume of work and size of the organization.

B.4 Improvement Functions

The chief change agent and SPI champion has the lead role in this set of functions. Each organization must decide whether to staff the position with a full-time specialist, or with a part-time effort for a suitably trained and experienced software engineer. This position, called by some the software engineering process (SEP) leader, must provide a combination of leadership and management along the lines discussed above.

Larger organizations often form an SPI Steering Committee to work with and support the SEP leader. The members of this committee represent the various organizations involved with and impacted by software, including senior management. In smaller organizations, the senior management group acts as the SPI Steering Committee. In either case, this committee sets quality improvement goals, and manages the overall initiative to improve software quality.

Staff that functions as work groups plan and implement changes that achieve the organizations quality goals. Because such activity impacts the core and support functions, it is important that the work groups be staffed by people from the core and support functions.

The Software Engineering Process Group of the Raytheon Equipment Division [10] is organized as shown in Figure 10.

The general form of this infrastructure, as Raytheon originally envisioned it, consisted of four entities:

1. An executive committee to provide direction and oversight
2. Working groups specializing in each of the major disciplines involved in process improvement
3. Task teams to develop the actual process changes that achieve the improvements
4. An SEP leader to monitor and coordinate day-to-day progress

This organizational structure has stood the test of time for Raytheon and continues today in this form. Each organization must create its own form of this infrastructure to support its SPI initiative.

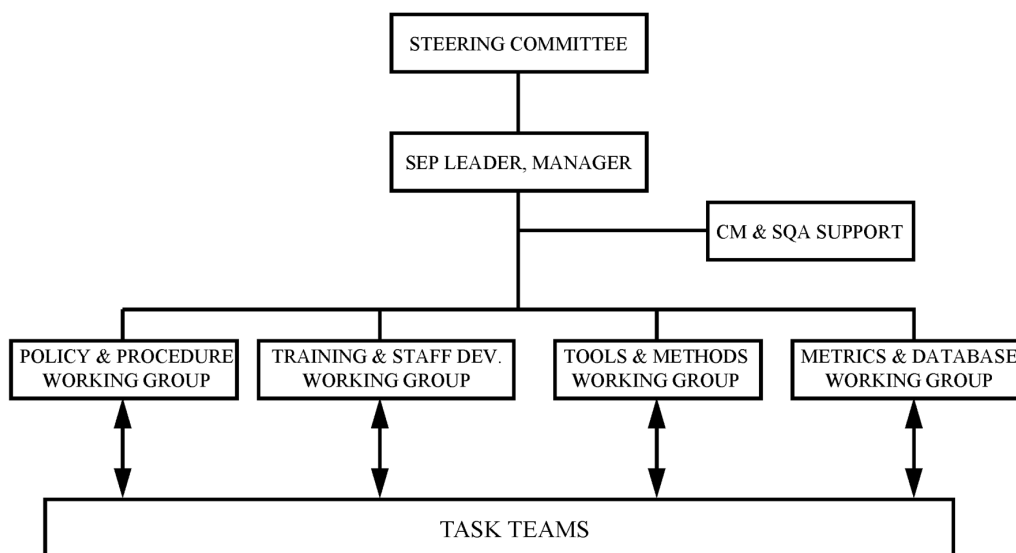


Figure 10 Example of a Software Engineering Process Group

B.5 Organizational Context

This is not a category of “functions,” so there is no staffing to do nor organizational structure to create. However, the groups in your company that interact with the software staff, and their particular patterns of interactions, might well influence the approach you take in staffing and organizing the five core software functions, as well as the support and improvement functions.

APPENDIX C

Reporting Improvements in Software Quality

Section 4 shows the continuous improvement of a software process as a progression of seven stages. By moving through these stages, an organization transforms its work on software from an ad hoc, labor-intensive activity to a managed, technology-supported engineering discipline. The material in this appendix introduces two topics related to this structure: recording a supplier's overall position along this scale, and consolidating those recordings for the supplier community as a whole (or at least a significant subset of it).

Producing objective quality indicators at regular intervals is the lifeblood of a quality improvement initiative. Management that is committed to SPI recognizes this and supports production of such indicators to the same degree it supports software production and field support. The committed organization uses these indicators first and foremost to drive its own improvement effort, and as a basis for staff incentives—technical staff and managers alike. Using the indicators to communicate with customers about the status of the quality improvement initiative, while very important, is secondary.

The following quality indicators are best suited to showing a supplier's overall progress.

Stage 1, Step 1: the 4-Ups Data, the fourth of which is the SSQA rating.

An SSQA covers many organizational activities. Module 3 deals with software; Modules 1 and 2 relate to other activities such as manufacturing, product engineering, purchasing, finance, and management. The software module is 12 key requirements:

1. **Documented Process:** An approved, documented process shall be used to guide the development and maintenance of all software that impacts total customer satisfaction.
2. **Software Project Planning and Control** mechanisms shall be in place and followed.
3. **Software Development System:** Software shall be developed as part of a total system using a phased development approach, intermediate deliverables, and review and approval shall be based on criteria for entry and exit from each phase.
4. **Documented Requirements:** Software shall be developed in support of documented (formal, written, approved, available) requirements, with conformance to the requirements verified.
5. **Configuration Management and Change Control:** Software shall be maintained under documented plans for configuration management and change control, including installation and customer configurations.
6. **Software Development Security:** Software shall be developed using proper tools and shall be documented, with approved procedures for security and information recovery, including disaster protection.
7. **Independent Testing of Software:** Software shall undergo system/acceptance testing by individuals or organizations not directly involved in the design or implementation or the product being tested. Testing shall reflect customer usage.
8. **Software Quality Goals:** Goals shall be established for software quality that are supported by measurement systems, with provisions for tracking progress toward these goals and highlighting issues from a customer perspective.

9. **Software Quality Organization** shall act as a customer advocate in software matters by assuring conformance to customer requirements and specifications, and proper execution of the approved development process.
10. **Continuous Software Quality Improvement:** A mechanism shall be used to ensure continuous quality improvement of software and of the software development process.
11. **A Software Capability Improvement Program** that includes deployment and assessment of training shall be in place for all software organizations.
12. **Software Subcontractors:** The processes used by software contractors shall be controlled, and conformance to requirements of subcontracted software is verified.

Each requirement is followed by several detailed examples of actions that satisfy the requirement. For instance, some of the examples given for the first requirement are as follows:

- a. A documented process for software development is approved by senior management.
- b. The process is followed with measurable and observable milestones established.
- c. Training is in place to educate people on the process.

Each of the 12 requirements is scored on a scale of 0-10 on each of the following four factors, using the SSQA scoring matrix:

- **Management Commitment:** Is management committed to this requirement?
- **Systems Approach:** Is there a systematic approach in place that defines the process?
- **Deployment:** Is the process deployed and being used throughout the organization?
- **Results:** Are there measurable results that can be attributed to the process?

The supplier organization is required to do a thorough self-assessment of its practices, and to document the results using the SSQA method at least once a year. The SEMATECH member companies periodically request an independent validation of the self-assessment scores. The scores are presented in the form of a bar chart, as illustrated in Figure 11.

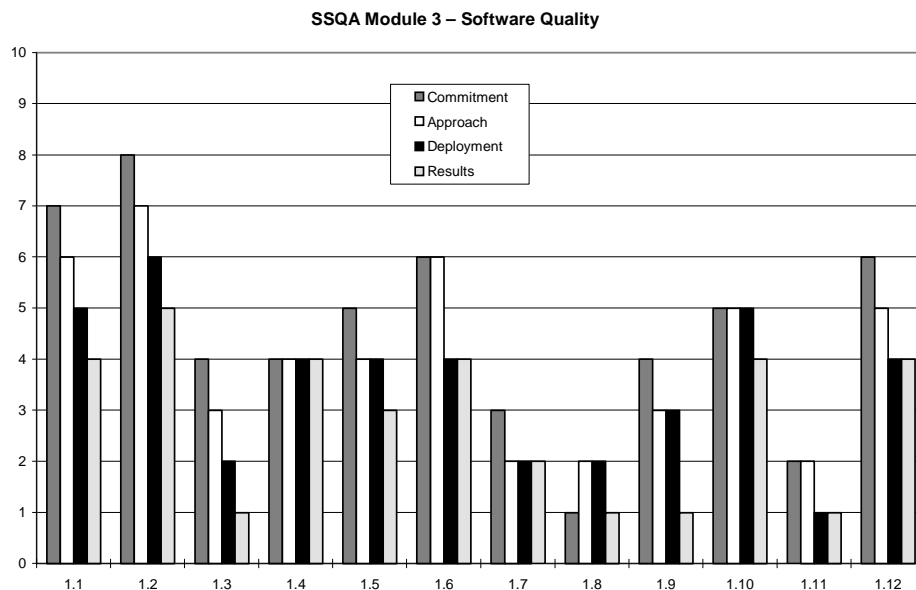


Figure 11 SSQA Module 3: Process Maturity

Scores for multiple suppliers are consolidated for each of the four factors, as illustrated in Figure 12. The plot for a specific supplier can be superimposed on consolidated data for the industry, showing how one organization stands relative to the community as a whole.

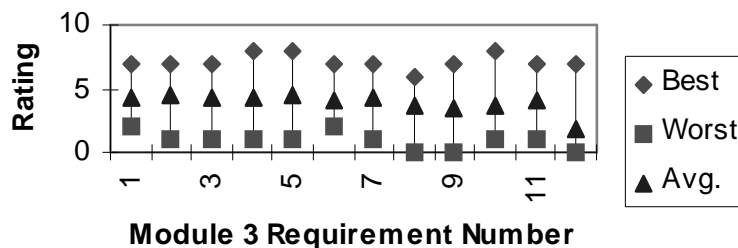


Figure 12 Management Commitment Composite Rating

Stage 3, Step 1: SSQA findings applied to (extended into) each of the Big 6 areas

The Big 6 areas of activity are directly addressed by the SSQA requirements, so findings from an SSQA assessment, and the improvement plans developed from those findings, address the needs for improvement in all Big 6 areas. To ensure this connection, an “easy to take” inventory has been developed. A copy of this inventory is provided as Table 13. For each item listed, three questions are asked:

1. Was that item done last year at this time?
2. Has it been done during the past year (is it in place now)?
3. Does the organization plan to have it done by this time next year?

Table 13 SPI Composite Index

Software Process Improvement Composite Index	1/1/96	1/1/97	1/1/98
<i>Configuration Manager</i>			
SCM policy and procedures document exists and is followed			
Configuration control board exists			
Configuration management library system in use			
Work products under SCM identified			
Release procedures followed			
Change request/problem reporting procedures followed			
Archiving and disaster recovery is done			
Subtotal Configuration Management			
<i>Testing</i>			
Test plan is made and used, managed, and controlled			
Test criteria are developed and reviewed with customers			
Testing (unit, integration, system, acceptance) is used			
Regression testing is performed			
Test coverage is measured and recorded			
Subtotal Testing			
<i>Peer Technical Reviews/Inspections</i>			
A written policy for peer reviews exists and is followed			
Adequate resources exist to perform peer reviews on each work product			
Peer review leaders and reviewers are trained			
Peer reviews are planned and the plans are documented			
Actions identified in peer reviews are tracked until they are resolved			
Subtotal Peer Technical Reviews/Inspections			
<i>Project Planning and Tracking</i>			
Project planning, tracking, and oversight policy and procedures exist			
Planning activities managed; statement of work exists			
Software development plans made and used; changes managed			
Project schedule made and tracked			
Estimates for size, effort, cost, and resources made and tracked			
Planning and replanning data available			
Monitoring of key milestones done			
Project/Senior management review of activities			
Subtotal Project Planning and Tracking			
<i>Requirements Management</i>			
Requirements Management policy and procedures exist			
Requirements documented and reviewed			
Requirements used as the basis for plans and work			
Changes to requirements reviewed and incorporated			
Requirements are tracked to test cases			
Subtotal Requirements Management			
<i>Defect Tracking</i>			
Defect tracking policy and procedure exists			
Defects are reviewed and tracked to closure			
Defect tracking tool is used			
Number and severity of defects in requirements is recorded			
Number and severity of defects in code is measured and recorded			
Subtotal Defect Tracking			
TOTAL			

Scorings from this assessment (0 = nothing done, 1 = partial implementation, 2 = fully implemented) focus on quality improvement in the six areas SEMATECH members view as critical. Consolidated assessment results can be plotted according to Figure 13.

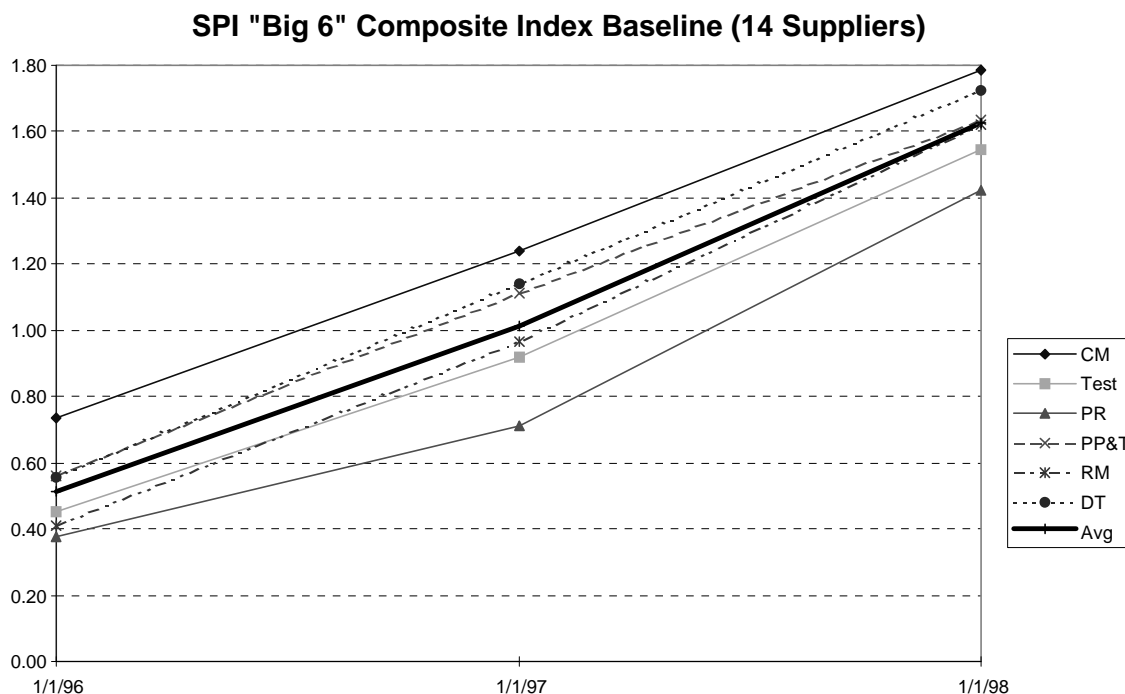


Figure 13 SPI Composite Index Baseline

Beyond Stage 3: CMM levels of maturity, with optional reports on the KPA level

Taking the steps that define the first three stages puts key elements of the supporting infrastructure for SPI in place, and initiates the work needed to satisfy many of the CMM's requirements for a Level 2 maturity rating. Completing that work and satisfying all requirements for the six KPAs on Level 2 is the work of Stage 4. A CMM-based assessment verifies when that work has been completed.

This process continues through the remaining three stages: completing the work of Stage 5 is marked with a Level 3 maturity rating, Stage 6 with a Level 4 rating, and Stage 7 with a Level 5 rating. Progress during a stage is indicated by the KPAs being implemented completely. Even partial implementation of a KPA can be indicated by which of its goals are currently satisfied.

APPENDIX D

Benefits of Improving Software Quality

Section 5 describes several partnering opportunities and types of teamwork that reduce the risk of failure inherent in any SPI initiative and cut the time needed to reach an initiative's goals. This appendix adds incentives for following the guidelines by citing several ways the participants can benefit from a successful SPI initiative, and highlighting the cost impact on this industry that unreliable software represents.

D.1 Reported Benefits

To provide better data about the results of SPI in settings that include high technology companies, the SEI conducted a study of 13 organizations and their experiences with CMM-based improvement initiatives [3a]. The study, completed in 1994, included four organizational components of SEMATECH member companies.

The results from this study are summarized in Table 14. These quantities were measured with different data definitions in the various organizations, so that changes within each organization over time are reported. Each category is presented with the range of reported data values. An in-depth discussion of this data is provided in Chapter 3 of the published report [3a].

Table 14 Reported Benefits of SPI Among 13 Organizations

Type of Benefit	Number of Companies	Benefit Range Annual	Benefit Median Annual
Productivity growth	4	9%–67%	35%
Increase in pre-test defect detection	3	6%–24%	22%
Decrease in time to market	2	15%–23%	19%
Decrease in field error reports	5	10%–94%	39%
Return on investment (ROI)	5	4.0–8.8	5.0

D.2 Cost of Unreliable Software

The cost of unreliable software is very significant for semiconductor manufacturing companies. In development fabs, it approximates 40% of the equipment interrupts. Although effective measurements of software unreliability in production fabs are not available, software is viewed as a critical problem. Software accounts for perhaps 3% of the interruptions in a production fab. The cost benefit of improving software reliability can then be expressed in the four ways, as summarized in the following paragraphs.

D.2.1 Decreased Downtime

A fairly conservative estimate of the cost of unplanned downtime caused by software is \$1.2M to \$7M per year for a typical fab, using the assumptions and estimates in Table 15 and Table 16.

Table 15 Assumptions Used in Assessing Software-Related Unplanned Downtime

Wafer starts		5000 per week
Operating time	7 days x 24 hours/day	168 hours/week
Hourly production	5000/168	29.76 wafers/hour
Good production (90% yield)	0.9 x 29.76	26.79 wafers/hour
Value of good production		\$1000 per wafer
Value per hour	1000 x 26.79	\$26,790 per hour

Table 16 Estimated Costs of Software-Related Outages

	Minimum	Typical
Typical software outages (unplanned downtime)	0.5% (50 minutes per week)	3.0% (5 hours per week)
Cost of outage per hour	0.005 x 26790 = \$133.95	0.030 x 26790 = \$803.70
Cost of outage per week	133.95 x 168 = \$ 22,503.60	803.70 x 168 = \$135,021.60
Cost of outage per year	22503.60 x 52 = \$1,170,187.20	135021.60 x 52 = \$7,021,123.20

Each member company can multiply these costs by the number of fabs it operates to get its total cost of unplanned downtime caused by software outages. The member companies operate approximately 150 production fabs, so the maximum possible total return is approximately \$1.05B (\$7.0M/fab x 150 fabs) per year.

How, one might ask, will SPI decrease unplanned downtime? A major contributor to unplanned downtime is software defects. The number of defects in shipped software will be substantially lower after SPI.

D.2.2 Decreased Cost of Ownership

The cost of ownership (COO) for a piece of semiconductor manufacturing equipment is a complex function of cost, yield, and equipment reliability characteristics. Following is a list of variables in the COO model [18] that can be affected by better software to reduce the cost of ownership:

- Decreased scheduled maintenance time
- Increased MTBA
- Decreased redo (rework) rate
- Decreased defect density
- Decreased original equipment cost
- Decreased system prove-in costs
- Decreased supply costs per system
- Decreased maintenance costs/system
- Increased MTBF
- Increased throughput
- Increased throughput yield
- Increased number of systems per operator
- Decreased other depreciable cost
- Decreased training requirements
- Decreased material costs/system
- Decreased personnel per system

D.2.3 Decreased Start-Up Costs

The traditional way of starting up a new fab is to conduct marathon runs on each new tool. If the run consists of 10,000 wafers, each worth \$1,000, then the cost of each run on each tool is \$10M.

Now, suppose a 300 mm fab has 25 new tools and each tool requires two marathon runs. The total start-up cost would be \$500M ($\$10\text{M}/\text{tool}/\text{run} \times 2 \text{ runs}/\text{tool} \times 25 \text{ tools}$).

What if an SPI program could shorten the marathon run by improving the quality of the software in each tool? A 2% improvement then would save \$10M.

To speculate further, suppose that most of the problems in the first run are caused by defective software (e.g., 40%). If software quality approached zero defects, only one run would be necessary, producing a \$250M savings per new fab!

D.2.4 Decreased Supplier Costs

Unreliable software is also very expensive for suppliers. The author identified one supplier whose equipment has 150 hours MTBF. Given the number of machines installed worldwide, that means one of that supplier's machines goes down every two minutes. If the MTBF were improved by an order of magnitude (10X), the need for field engineers would be lessened.

Unreliable software hurts the suppliers in other ways. Poor reliability means a need for more field support for the tool population. Given the explosion in the number of tools worldwide, there is little wonder the supplier companies cannot staff enough field engineers to support unreliable equipment.

If the equipment supplier depicted in Table 17 had 1000 tools in 1991, it would have needed 100 field engineers. However, if the company had 5000 tools in 1995, it would have needed 1250 field engineers. The number of field engineers would have gone up by a factor of 12.5, while the number of tools would have increased by a factor of five. Assuming each field engineer costs the equipment supplier \$100K per year (a conservative figure), field service costs would have risen by \$115M $((1250-100) * \$100\text{K})$.

Table 17 One Supplier's Ratio of Field Engineers to Process Tools

Year	Number of Machines per Field Engineer
1991	10
1995	4

**SEMATECH Technology Transfer
2706 Montopolis Drive
Austin, TX 78741-6499**

<http://www.sematech.org>