Software Process Improvement Handbook

A Practical Guide

Karl E. Wiegers

PROCESS IMPACT

11491 SE 119th Drive
Clackamas, OR 97015-8778
phone: 503-698-9620 fax: 503-698-9680
www.processimpact.com

Contents

Introduction

I have never met anyone who could truthfully say, "I am building software today as well as software can ever be built." Unless you can legitimately make this claim, you should always be looking for better ways to manage and execute your software projects. This is the essence of software process improvement (SPI).

The fundamental objective of process improvement is *to reduce the cost of developing and maintaining software*. It's not to generate a shelf full of processes and procedures. It's not to comply with the dictates of the most fashionable process improvement model or framework. Process improvement is a means to an end, and the "end" is superior business results.

The word *process* has a negative connotation in many organizations. Developers think of the "process police" (a phrase I detest) imposing seemingly arbitrary changes on an organization that's already enjoying business success. Developers are afraid that defined processes will cramp their style and stifle their creativity. Managers fear that following a defined process will slow the project down. Sure, it's possible to adopt inappropriate processes and follow them in a dogmatic fashion that doesn't add value or allow for variations in projects and people. But this isn't a requirement! Sensible, appropriate processes help software organizations be successful consistently, not just when the right people come together and pull off a difficult project through heroic efforts. Process and creativity aren't incompatible. I follow a process for writing books, but this process doesn't restrict the words I put on the page. Nor is process improvement at odds with the agile software development movement that is popular at present. Agile is simply one strategy for process improvement that complements the more traditional, heavier approaches.

Despite its apparent simplicity, process improvement is hard. It's hard to get people to acknowledge shortcomings in their current ways of working. It's hard to get busy technical people to spend time learning and trying new techniques. It's hard to get managers interested in future strategic benefits when they have looming deadlines. And it's very hard to change the culture of an organization, yet process improvement involves at least as much culture change as it does changes in technical behavior.

This handbook addresses many issues that can help software organizations implement and sustain a successful process improvement program. I've led process improvement efforts in small teams building information systems and process control software, in a division of 500 software engineers building embedded and host-based digital imaging products, and in a world-class Internet development organization. This handbook summarizes many insights I gained from these diverse experiences. True stories from real projects are flagged with a newspaper icon in the margin. Worksheets at the end of each chapter give you an opportunity to apply the principles and practices to your own projects. I also warn you about several common traps to avoid; these are marked with an icon of a mousetrap in the margin. Refer to Chapter 4 for more details on some of the most treacherous traps. Because it's not a comprehensive tutorial on every aspect of SPI, this handbook contains many references to the extensive body of process improvement literature.

Many software and systems organizations have enjoyed the benefits of systematic process improvement over the past 15 years. With patience, steadfastness of purpose, and a little help from this handbook, so can you.

Chapter 1. Why Is Process Improvement So Hard?

In 1995 I began a new job leading the software process improvement efforts in a large engineering division. Soon I met a woman who had had the same job for about a year in another division. I asked my counterpart what attitude the developers in her division held toward the program. She thought for a moment then replied, "They're just waiting for it to go away." If process improvement is viewed simply as the latest management fad, most practitioners will just ride it out, trying to get their real work done despite the distraction.

What software team wouldn't want to improve its productivity, ship better products sooner, reduce its maintenance burden, and enhance their employer's bottom line? The software literature has an abundance of success stories of companies that substantially improved their software development and project management capabilities. However, many organizations that tackle process improvement fail to achieve significant and lasting benefits.

Why is something that seems so simple so difficult to achieve in practice? After all, we have the role models of successful companies that have published their process improvement experiences, and we have a growing body of software industry best practices to draw on. Organizations like the Software Engineering Institute (http://www.sei.cmu.edu) have given us many—perhaps too many—frameworks for guiding our process improvement efforts. As a consultant, I meet a lot of smart software developers and managers, and yet many are having a hard time getting a decent return on their investment in SPI. This chapter presents five major reasons why it's difficult to make process improvement work and provides some suggestions for avoiding these pitfalls (Wiegers 1999b). We'll address the challenges of:

- Not enough time
- Lack of knowledge
- Wrong motivations
- Dogmatic approaches and
- Insufficient commitment.

Challenge #1: Not Enough Time

Insane schedules leave insufficient time to do the essential project tasks, let alone to investigate and implement better ways to work. No software teams are sitting around with plenty of spare time to explore what's wrong with their current development processes. Customers and senior managers are demanding more software, faster, with higher quality, and incidentally we have to downsize a bit and we can't pay for overtime, so sorry. One consequence is that a team might deliver release 1.0 on time, but then they have to ship release 1.0.1 immediately thereafter to fix the most egregious problems.

Part of the problem is a mindset that views the time to initial release as the only important factor in project success. While sometimes this is the case, I think it's really true far less often than is claimed. I’ve used some well-known commercial software products that might have met their ship dates (I don’t know for sure), but I’d be embarrassed to admit I was on the development team. I would have preferred the developers spent more energy on improving quality, a goal of many process improvement efforts.

Quality shortfalls that are addressed late in the project can cause schedules to slip, but processes that build quality in from the outset can actually shorten cycle times. This benefit is counterintuitive to many people, who don’t appreciate the potential 3- to 10-fold return from investing in quality (Jones 1994). A holistic cost-of-quality perspective looks at the life cycle costs of a product, including maintenance and customer support costs, not just the time to initial release. Even that perspective doesn’t account for the time customers waste working around defects in the software and their annoyance toward the company that created it.

Manufacturing industries realize they need to take their equipment off line periodically for retooling, performing preventive maintenance, and installing upgrades that will improve productivity or quality. The product managers and marketing people accept this planned downtime as a necessity and adjust their production schedules around it. Process improvement is the software industry’s equivalent of machinery upgrades. It upgrades the capabilities of both individuals and organizations to execute software projects. You don’t ordinarily shut down the software development machine, so you have to implement the upgrades in parallel with your production work.

It’s always difficult to find the time, but I know of one Internet development project that takes a few weeks between major releases to explore new technologies, experiment, and implement improvements. A periodic release schedule provides an opportunity to reflect on what went well and not-so-well on the last release, to focus some improvement energy in a few high-priority areas, and to quickly flow improved practices into the next development cycle or iteration.

You need to integrate your process improvement activities with development as a routine way you spend some of your time on every project. Devote a specific percentage of your development budget and effort to ongoing process improvement. Think about these possible levels of commitment:

- If you spend less than about three percent of your budget on process improvement (including staff effort, training, process assessments, and outside consultants) you’re dabbling.
- If you invest six or eight percent you’re serious about making some significant headway.
- Investing 10 or more percent indicates a major commitment to aggressive improvement.

Include these resources and tasks in your project plans and schedules. Don’t commit the technical staff 100 percent to project work and hope the process improvement gets done by magic; it won’t. If you don’t spend the time to improve how you perform technical and management work starting tomorrow, you shouldn’t expect your next project to go any better than your current one.

See Trap #3, time-stingy project managers, in Chapter 4.

Challenge #2: Lack of Knowledge

A second obstacle to widespread process improvement is that many software practitioners aren't familiar with industry best practices. I informally survey audiences at conferences and training seminars, asking how many attendees perform one practice or another and how many people are familiar with certain topics I'm discussing. The unscientific results suggest that the average software developer doesn't spend much time reading the literature to learn about the best known software development approaches. Developers might buy books on Java and .NET, but don't look for anything about process, testing, or quality on their bookshelves.

The industry awareness of established process improvement frameworks has grown in recent years. These frameworks include the Capability Maturity Model® for Software or SW-CMM® (Paulk 1995) and the CMM® Integration for Systems Engineering and Software Engineering or CMMI® (Chrissis, Konrad, and Shrum 2003).[1] Effective and sensible application often could be better, though. Too few developers and managers are actively applying the experience collected in bodies of software knowledge such as the IEEE Software Engineering Standards Collection (http://www.computer.org/cspress/CATALOG/st01121.htm) or the Software Engineering Body of Knowledge (Abran and Moore 2004). Many recognized industry best practices simply are not in widespread use in the software development world. These best practices include inspections, risk management, change control, metrics-based project estimation and tracking, and requirements management (Brown 1996; Brown 1999).

To overcome this barrier to successful SPI, read the literature! Learn about industry best practices from sources such as Steve McConnell's *Rapid Development* (McConnell 1996). Leverage the sharing of local best practices among the people in your team or department, so all can learn from those individuals who have strengths in particular areas. Facilitate a periodic lunch 'n' learn session (lubricated with pepperoni, sugar, and caffeine) in which team members select articles or book chapters to read, discuss, and seek to apply to your project. As a manager, reward those team members who take the initiative to learn about, apply, and share practices that can boost the group's technical and management capabilities. The information is out there; find it and use it.

See Trap #6, inadequate training is provided, in Chapter 4.

® CMM, CMMI and Capability Maturity Model are registered in the US Patent & Trademark Office by Carnegie Mellon University.

[1] The SW-CMM (often simply called the CMM) is a 5-level framework for guiding a software organization's strategic process improvement initiative. Each level except Level 1 requires that the organization's projects master a defined set of *key process areas*. The successor to the SW-CMM is the CMMI, which integrates system and software engineering process improvement activities. Vast quantities of information about these—and other—capability maturity models and process improvement issues are available at the Web site for the Software Engineering Institute, where they were developed, http://www.sei.cmu.edu.

Challenge #3: Wrong Motivations

Some organizations launch process improvement initiatives for the wrong reasons. Maybe a customer demanded that the development organization achieve CMM Level X by date Y. Or perhaps a senior manager learned just enough about the CMM to be dangerous and directed his organization to climb on the CMM bandwagon. Someone once told me that his organization was trying to reach CMM Level 2. When I asked why, all I got was a blank stare. Striving for Level 2 (and beyond!) is just what you do if you're playing the CMM game, right? I don't think so. He should have been able to describe some problems that the culture and practices of Level 2 would help solve or the business results his organization hoped to achieve as a benefit of reaching Level 2.

The basic motivation for SPI should be to make some of the current pain you experience on your projects go away (see "Focus on the Pain" in Chapter 2). Team members aren't motivated by seemingly arbitrary goals of achieving a higher maturity level or an external certification just because someone has decreed it. However, most people should be motivated by the prospect of meeting their commitments, improving customer satisfaction, and shipping excellent products that meet customer expectations. Many process improvement initiatives encounter immediate practitioner resistance when couched in terms of "doing the CMM," without a clear explanation of why improvement is needed and the benefits the team hopes to achieve.

Start with some introspection. What aspects of your current ways of working cause the most late nights at the keyboard? Which situations lead to frustrating rework of completed effort? Are you struggling most with meeting schedules, applying new technologies effectively, developing the right product, or achieving the desired level of quality? You probably already know what the lurking undercurrents of pain are, but an external process assessment can bring focus and legitimacy to these problem areas. A post-project or mid-project retrospective also provides a structured way to reflect on what has gone well so far (keep doing it) and what has not (do something different next time) (Kerth 2001).

Challenge #4: Dogmatic Approaches

A fourth barrier to effective process improvement is the checklist mentality, a rigid and dogmatic implementation of the SW-CMM, CMMI, or other process improvement framework. The checklist mentality focuses on the spurious objective of gaining some kind of process certification, not the real objective of improving your project results. I have heard managers proclaim that because their team has created the proverbial big honkin' binder of development procedures, they've achieved process improvement. Yay? Nay! The bottom line of process improvement is that the members of the team are working in some new way that gives them collectively better results. You can have fabulous processes, but if no one follows them, if they collect dust on the shelf, if people still work as they always have, then you haven't succeeded with process improvement.

Process change is culture change. Managers and change leaders must realize they need to change the organization's culture, not just implement new technical practices. Changing the way people think and behave is a lot harder than installing a new tool that runs twice as fast as the old one. The organization's leaders must steer the team toward improved ways of working by setting realistic (not unattainable) improvement goals, tying process improvement to business results, leading by example, and recognizing those who contribute to the change initiative.

Don’t feel that you have to comply with every expectation presented by a process improvement framework like the CMM. These are intended to be guidelines that users must thoughtfully adapt to their environments (see Chapter 6). For example, many development organizations have problems with their requirements development practices, but this topic isn’t addressed at CMM Level 2. If it hurts, fix it, no matter what the checklist says.

See Trap #5, achieving a maturity level becomes the primary goal, in Chapter 4.

New processes must be flexible and realistic. One of my consulting clients mandated that all software work products would undergo formal inspection, which is a great idea but not a realistic expectation for most organizations (Wiegers 2002). Their waiver process will be running overtime and practitioners might play games to appear as though they’re complying with this unrealistic policy. If your processes don’t yield the desired results when applied in good faith by informed practitioners, they aren’t the right processes for you. People will find ways to bypass unworkable processes—and they should.

Team members have to recognize that not every new process will benefit them personally and directly. Most people’s initial reaction to a request to do something new is “What’s in it for me?” This is a natural human reaction, but it’s the wrong question. The correct question is “What’s in it for *us*?” Foster a mindset of collaborative teamwork to show how a change in the way Person A works can provide substantial benefits to Persons B and C in the downstream development and maintenance activities, even if it costs Person A more time today. There needs to be a net gain for the project, organization, company, and customers, but not necessarily for every project participant.

Challenge #5: Insufficient Commitment

One more reason why SPI fails is that organizations claim the best of intentions but the managers and practitioners lack a true commitment to process improvement. They hold a process assessment but fail to follow through with actual changes. Management sets no expectations of the development community around process improvement. They devote insufficient resources, write no improvement plan, develop no roadmap, and implement no new processes.

It’s very easy to achieve a zero return on your investment in process improvement. You simply invest in a process assessment, train the team, write process improvement action plans, and develop new procedures, but the people in the organization never actually change the way they do their business. As a result, managers lose confidence and interest; practitioners conclude that, just as they suspected, the whole thing was an idle exercise; and frustrated process improvement leaders change jobs.

Sometimes, initial enthusiasm and commitment wane when quick results are not immediately forthcoming. Some improved practices, such as the use of static code analysis tools, can yield better results immediately. Other practices, such as metrics-based estimation, may take awhile to pay off. Respect the reality of the learning curve (Figure 1-1), in which you take a short-term performance hit in the early stages of any change initiative. Your investment in process improvement has an impact on your current productivity because the time you spend developing better ways to work tomorrow

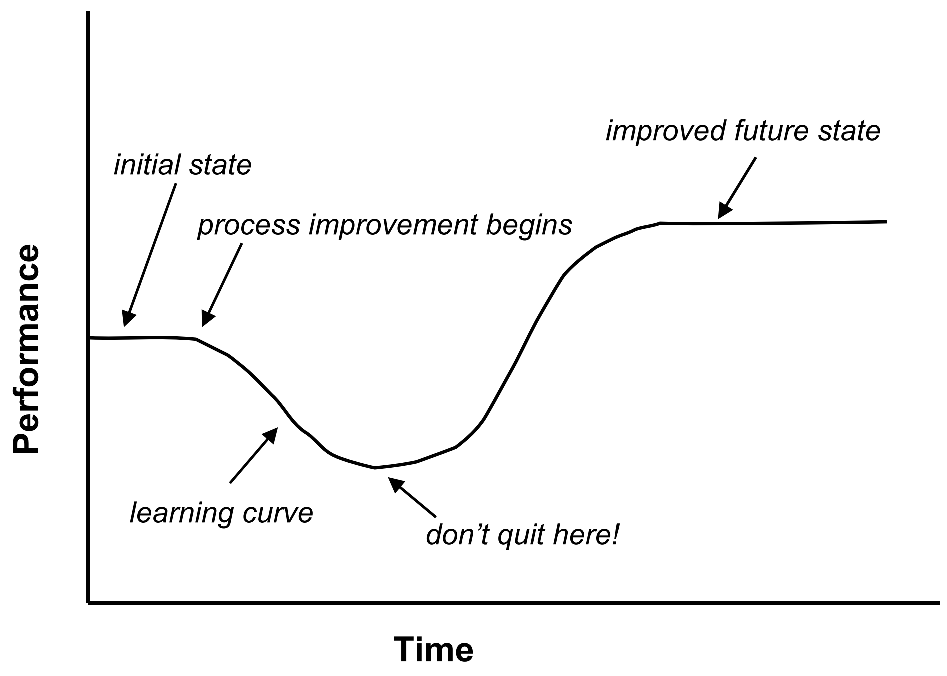


Figure 1-1: The learning curve

isn't available for today's assignment. If you can push through the learning curve—avoiding the temptation to give up before you start seeing results—eventually you'll wind up at a higher level of performance SPI is a bit like highway construction. It slows everyone down a little bit for a while, but once it's done, the capacity is much greater and the ride much smoother. It can be tempting to abandon the effort when skeptics see the energy they want devoted to immediate demands being siphoned off for the hope of a better future. Don't give up! Take motivation from the very real, long-term benefits that many companies have enjoyed from sustained software process improvement initiatives. For some examples, see Haley (1996) and Diaz (1997).

What Should You Do?

You can improve the way you manage and implement your software projects; you have to. Keep the following tips in mind to avoid having these pitfalls sabotage your SPI effort.

- Regard SPI as an essential strategic investment in the future of your organization. If you don't start now, you'll have a harder time keeping up with the companies that do. Companies in India, for example, have aggressively embraced process improvement as a way to gain a competitive edge in development efficiency and product quality. The current popularity of outsourcing American software development to offshore companies suggests that this strategy is working.
- Focus on the business results you wish to achieve, using SPI as a means to that end, rather than being an end in itself.
- Expect to see improvements take place over time but don't expect instant miracles. Remember the learning curve.

- Thoughtfully adapt existing improvement models and industry best practices to your situation. Pick an established approach and use it as a guide, not a straightjacket.
- Treat process improvement like a project. Develop a strategic process improvement plan for your organization and tactical action plans for each focused improvement effort (see Chapter 2). Allocate resources, manage risks, establish schedules, track progress against the plans, measure results, and celebrate your successes.

Practice Activities

1. Complete Worksheet 1-1.

Worksheet 1-1: Your Project's Challenges

Analyze the threat that the five challenges from this chapter pose to your process improvement initiative on a scale from 0 (don't worry about it) to 5 (a program-killer). Think about the possible impact each challenge could present and brainstorm some strategies for controlling these impacts.

Challenge	Threat to Your SPI Success (0-5)	Likely Impacts	Mitigation Strategies
Not enough time			
Lack of knowledge			
Wrong motivations			
Dogmatic approaches			
Insufficient commitment			

Chapter 2. An Effective Process Improvement Approach

When I worked at Kodak, two information-systems divisions had particularly successful process improvement programs. Both organizations had sincere management commitment at all levels, developed both strategic and tactical plans for the process improvement project, established small groups of people dedicated to SPI on a full-time basis, and engaged nearly all members of the organization on a part-time, periodic basis. These organizations achieved their process improvement goals and enjoyed the intended business benefits.

Many development organizations are climbing onto the software process improvement bandwagon, but too many of them are falling off again. The failure scenario looks like this:

1. Spend money and time on process assessments, consulting services, and training seminars.
2. Create a big honkin' binder of procedures and tell all team members they have to follow all of the procedures starting immediately.
3. Have senior management decree, "Make it so!"
4. Watch the binders gather dust as team members never really change the way they work.

This chapter presents recommendations that can help you avoid this exercise in futility (Wiegers 1999a). I present practical tips for how to approach SPI, describe the basic process improvement steps, and show you how to set up the organizational structures for a successful SPI program. This chapter addresses a management-driven, organization-wide process improvement initiative. Chapter 3 describes the types of personal improvements that all software professionals can make on their own.

The Process Improvement Cycle

At its core, process improvement is simple: consistently apply the practices that give you good results and modify the practices that cause problems. This requires honest introspection and analysis to identify the reasons behind your previous projects' successes and shortcomings. You also need to treat your process improvement program as a project, giving it the structure, planning, resources, goals, and respect that any project deserves.

Figure 2-1 illustrates an overall SPI cycle. After defining your desired business objectives, evaluate your current processes, problems, and project outcomes through an assessment. Armed with the assessment insights and knowledge of software industry best practices, you can then set realistic improvement goals. Select a small set of appropriate practices that will address your current process shortcomings and move you toward the goals. Identify a project or two on which to pilot new processes and make adjustments before officially rolling them out.

Planning how you'll implement new ways of working greatly increases your chance of success. The hardest part is actually implementing an action plan; if you don't do this part, nothing will really change. Give the new processes some time, then see whether they're easing the problems those

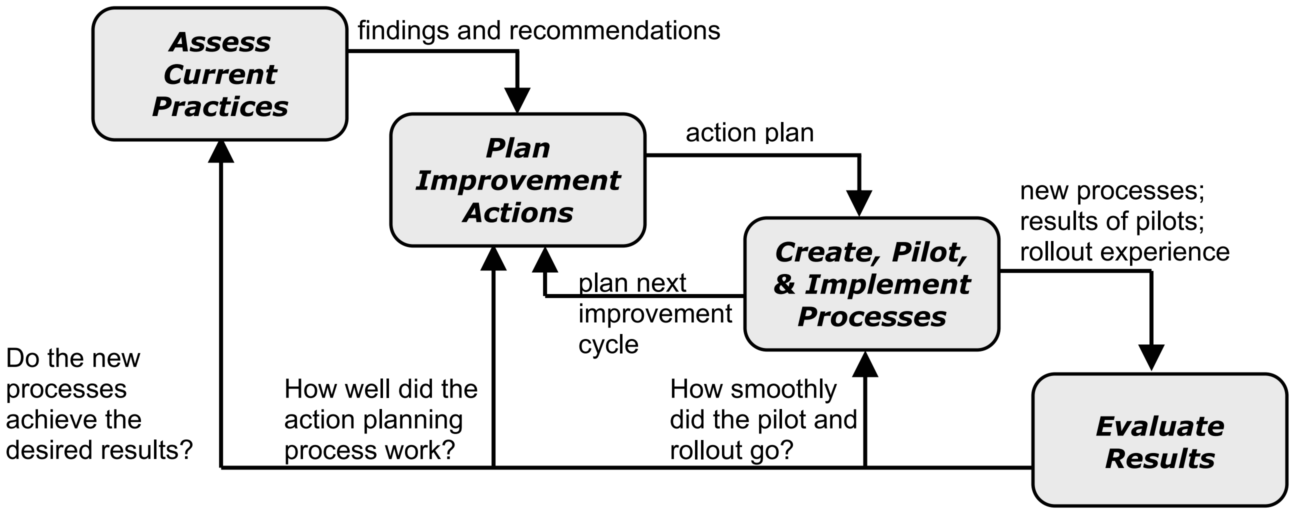

Assess Current Practices
findings and recommendations
Plan Improvement Actions
action plan
Create, Pilot, & Implement Processes
new processes; results of pilots; rollout experience
Evaluate Results
plan next improvement cycle
Do the new processes achieve the desired results?
How well did the action planning process work?
How smoothly did the pilot and rollout go?

Figure 2-1: The software process improvement life cycle

improvement actions targeted. Hard data about process benefits is more convincing than subjective perceptions. Then continue the improvement cycle with the next most pressing need. Process improvement is a journey, not a destination.

Focus on the Pain

Pain is the best motivation for changing the way people work. I don't mean external or artificially induced pain, but the very real pain we sometimes experience from our current methods. One way to encourage people to change is to persuade them that the grass will be greener in the intended future state. A more compelling motivation is to point out that the grass behind them is on fire.

The assessment (or appraisal) helps reveal the points of pain and major risks facing your projects, as well as highlighting the team's strengths. Assessments range from a simple brainstorming session, to a structured internal mini-assessment (see Chapter 7), to a semi-formal evaluation by an external consultant. The most rigorous appraisals are conducted according to the requirements of an established process model such as the CMM or CMMI. These formal appraisals are expensive and time-consuming, but they thoroughly evaluate your current practices against the process standard.

A brainstorming session is the cheapest, quickest way to identify problem areas. One of my small teams took this approach when we launched a grassroots improvement effort in 1990 (Wiegers 1996a). Our goal was to think of as many improvement opportunities (a euphemism for "problems") as we could, then envision possible solutions and commence working on them in earnest. For half an hour, everyone silently wrote problems and concerns on index cards and threw them in a pile on a table. As ideas dried up, we took each other's cards out of the pile and used them to stimulate new thoughts. When the idea-generating period was over, we weeded out duplicate ideas, leaving thirty-three unique issues. In a second pass through the cards, each participant wrote possible solutions to the problem described on each card. After some discussion to focus on the major items, we had identified several tangible improvement opportunities to tackle.

In another experience, a group of 18 software developers went through a two-part exercise to identify current barriers to our software productivity and quality. In session one, the participants wrote their thoughts about this topic on sticky notes, one idea per note. A facilitator collected and grouped the ideas as they were generated, a technique called affinity mapping. We came up with a dozen major categories, which we recorded on large flipchart sheets. In the second session, the same participants wrote ideas for overcoming these barriers on sticky notes and attached them to the appropriate flipcharts. Further refinement led to a handful of action items that could be addressed as mini-projects to help us achieve superior software quality and productivity.

In my experience, a process assessment rarely reveals major surprises. Most groups are already aware of their problems. Having an external party conduct the assessment legitimizes it because the external party is free of the organization's politics, historical tensions, and personalities. The assessment lets you confront the uncomfortable problem situations directly. An assessment absolutely is *not* a forum for finding fault or scapegoats.

Performing an assessment also can demonstrate management's commitment to process improvement. Be sure to follow through on the assessment's findings and recommendations. If you don't, you've lost the money and time you invested in the assessment, as well as losing credibility with frustrated team members who conclude that management isn't serious about making changes. An assessment is an investment, not an achievement.

Most assessments identify more improvement opportunities than you can address at once. "Focus" is a key word in process improvement. It takes longer than you might think to devise better processes and make them part of the way your group routinely operates. I once saw a highly motivated 20-person project team tackle seven improvement activities at the same time. Resources were spread thin, priorities were unclear, and the team accomplished little despite their enthusiasm.

State your process improvement objectives in terms of the desired business results. For example, a goal could be to eliminate incorrect builds being handed off from development to system testing. The goal is *not* to write a build-and-promote procedure. The SPI activities are a means to an end, not an end in themselves. Your managers should be able to describe how they expect the group's behaviors and results to be different if the SPI program succeeds.

Based on your top priority goals, pick two or three improvements to address initially. If you complete them quickly, great; go back to the assessment report and select the next area to work on. Take baby steps and don't try to do too much right out of the gate. Large organizations can undertake several change initiatives concurrently across multiple projects, but each project team should focus on just a few areas at a time.

Communicate, Communicate, Communicate

People naturally push back when they're asked to change the way they work because they have a comfort level with familiar (if not always effective) practices and some fear of the unknown. Concern about burdensome process overhead that stifles creativity and delays delivery is also common, but the fear normally exceeds the reality. Your team members might experience a sense of failure because the assessment identified process shortcomings, even though they've been working as hard as they can. Customers and other external groups might view the SPI program as raising barriers that will make it harder for them to get what they want from the developers.

To address these issues, permeate process improvement with communication. Articulate the price being paid for the current process shortcomings. This price might include blown schedules, missing functionality, massive overtime, high product support costs, unhappy customers, or low morale. Explain to skeptics what benefits the improvement activities will provide to individuals, the project team, the company, and its customers. Allay the fear of overwhelming change by stressing that new processes will be selected thoughtfully, created by the team members themselves, and rolled out gradually. Look for "allies" who are willing to try new procedures and document templates, provide feedback, and help lay the groundwork for successful change.

Trumpet your achievements to the team members and other stakeholders. Publicly recognizing the individuals who contributed to each successful change demonstrates that constructive participation in the SPI program is a desired behavior. Analyze unsuccessful change attempts to understand why they struggled and adjust your approaches accordingly.

A key SPI operating principle is "Gentle pressure, relentlessly applied." Keep your improvement objectives and status visible to the entire team. Stress how the SPI goals align with your organization's business goals. Set aside time during team meetings to review the process improvement progress. Periodically report successes to management to bolster their commitment to the effort.

A kiss of death is to launch an improvement program with great fanfare at the beginning of the year, then never mention it again until you see if you reached your goals at year's end. You won't. Working on project activities will always be viewed as more important than contributing to process improvement. Managers must frequently remind the team that SPI work also is necessary and valued.

Organize for Process Improvement

Organizations that are serious about SPI often set up the three-level structure shown in Figure 2-2. This structure shouldn't be any more complicated than necessary to insure that sensible improvement actions are identified, initiated, implemented, and effective.

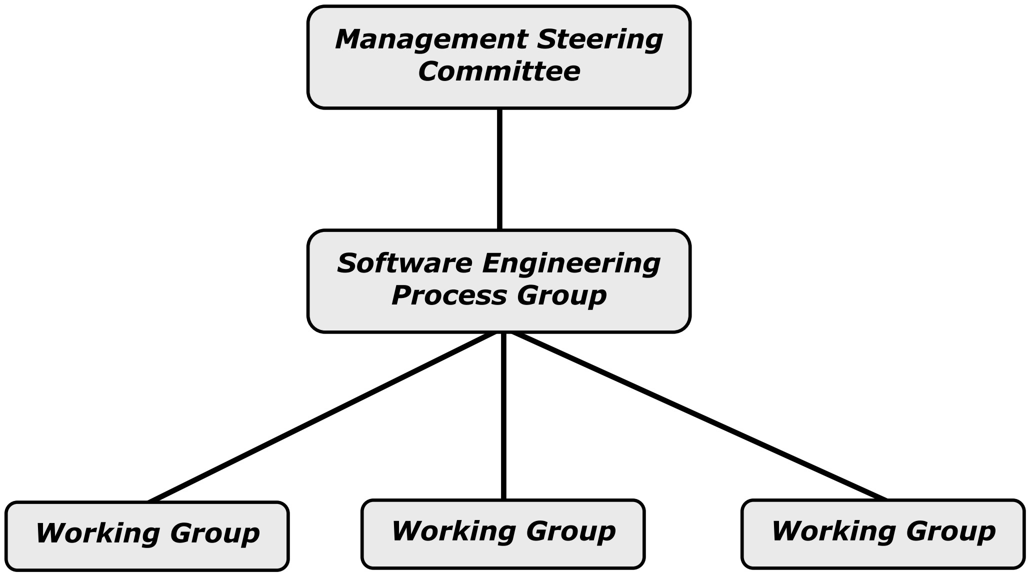

Management Steering Committee
Software Engineering Process Group
Working Group
Working Group
Working Group

Figure 2-2: A typical process improvement organizational structure

Management Steering Committee

The management steering committee commits resources and sets direction and priorities. Typical MSC members include the organization's senior manager, the manager who's responsible for the SPI program, the individual leading the SPI effort, and selected project or department managers. Getting several management levels to participate in the MSC indicates that the organization takes improvement seriously. The MSC's responsibilities include:

- Setting improvement area priorities
- Chartering working groups to address specific process areas
- Monitoring activities and status
- Assessing the impact of the improvement actions completed to date
- Managing process improvement risks and breaking down barriers

The MSC might also identify a "process owner" for each process area it addresses (such as requirements engineering, peer reviews, testing, or estimation). The process owner is a manager who charters and leads working groups. He provides continuity as individual working groups come and go. The process owner serves as the point of contact for requests to improve that area's processes and supporting process assets (process descriptions, procedure, document templates, policies, forms, checklists, examples, and the like). Without ownership, processes can decay over time and fail to meet the organization's current needs. The ideal process owner is an enthusiastic, committed champion, a sustaining sponsor who serves as the organization's authority for that process area. As senior management's delegate, he sets expectations for application of the process through an organizational policy. Figure 2-3 lists some typical responsibilities for a process owner.

Software Engineering Process Group

The software engineering process group or SEPG (variously pronounced "S-E-P-G," "sep-gee," "see-peg," and—more whimsically—"seepage") coordinates the various process improvement actions (Fowler and Rifkin 1990). It's not the SEPG's process improvement program, though; it's management's program. The SEPG acts as management's agents to implement the program. A large organization's SEPG should have a full-time manager, a few full-time software process specialists, and a rotating cast of practitioners who participate on a part-time basis for a year or so at a time.

Process specialists often come out of the testing and quality assurance ranks. At least some of the SEPG members should have substantial development experience, preferably in the same organization they're supporting. Project management experience also is valuable. These qualifications will give the SEPG more credibility with developers and managers than if it appears to be a bunch of ivory-tower theoreticians who don't know how real software development gets done.

SEPG members learn a lot about assessments, process improvement frameworks, how to write good processes and procedures, and how to influence change. The human and organizational issues of change are at least as important as the technical aspects. Effective SEPG members are well-organized, patient, flexible, and effective communicators who can adapt their approaches to individual situations. They are skilled facilitators, able to lead fruitful discussions on sensitive topics with diverse participants, some of whom would rather be somewhere else.

1. Maintain expert knowledge of the process area and the details of the local process.
2. Establish and enforce management policies for the process area.
3. Convene improvement working groups for that process area and coordinate their activities.
4. Lead the writing of tactical process improvement action plans for that process area.
5. Arrange for training as needed, both in the technical domain of the process area and in the local process specifics.
6. Determine how to reward the early adopters of the new processes.
7. Monitor the use of the process on projects and evaluate its effectiveness.
8. Communicate successful process results throughout the organization.
9. Use metrics data to evaluate process effectiveness and to identify further improvements.
10. Define pertinent process metrics. Set expectations for collecting and using these metrics.
11. Review process modification requests and implement approved requests.
12. Ensure that the organization's process assets library contains the process descriptions, process assets, and training materials.
13. Maintain information about tools that support the process.

Figure 2-3: Possible process owner responsibilities

The SEPG serves as a resource to all those involved in the process improvement program. Their expertise, resources, and outside connections accelerate the change initiative because practitioners involved in the effort have a place to go for help. SEPG members typically perform the following functions:

- Develop strategic and tactical improvement action plans
- Lead or participate in process assessments
- Facilitate and coordinate process improvement working groups
- Collect information about industry best practices
- Accumulate process assets and share them across the organization (see Chapter 5)
- Review new processes, procedures, and templates that working groups create
- Lead improvement activities that span the entire organization, such as metrics and training programs

Working Groups

It's not a good idea to have the SEPG simply write all the new procedures and inflict them on the project teams. This classic "throw it over the wall to land on the victims' heads" approach usually fails. Instead, get practitioners involved in developing realistic new procedures through their participation in process improvement working groups, also called process action teams (PATs) or process improvement teams (PITs). A working group is an ad hoc team of perhaps three to six project representatives who address a specific improvement area. Their deliverables typically include a description of the processes currently being used in that area, as well as new processes and process assets. A working group's efforts might lead to new processes for just one project, or they could develop processes for an entire organization to use.

The working group shouldn't invent every new process asset from scratch. They should begin by looking through your local process assets library—a shared collection of process-related resources—for relevant starting points. Many sample procedures and document templates are available through the Internet, some free and some for sale. Unfortunately, good examples of actual project deliverables are hard to find because they're usually proprietary. Kim Caputo provides several sample process descriptions in her book, *CMM Implementation Guide* (Caputo 1998). Consultants will gladly build customized processes for your organization; I've done many of those myself for my clients. However, all externally-obtained materials need to be tailored to suit your project needs. Practitioner input into the working group's products develops a sense of shared ownership and confidence that the process materials will work for your local projects.

Scope each working group's objectives such that it can complete its tasks within two to three months. If the working group goes on for too long, the members can lose their enthusiasm and turnover is likely. Try to get all practitioners and project managers involved in a working group at some point (although not all at once). A member of the SEPG can launch each working group and facilitate its initial meetings. However, it's important for the development organization to own its improvement program and sustain its own working group efforts after the SEPG removes the training wheels.

Plan Your Actions

As with any project, your SPI program needs to be planned. Two useful planning components are an overall strategic software process improvement plan and a tactical action plan for each working group (Potter and Sakry 2002). Some people balk at writing plans, viewing them as wasteful overhead effort. However, writing the plan isn't the hard part. The hard part is thinking, asking, listening, understanding, negotiating, and thinking some more. Actually writing the plan is mostly transcription at that point. I've found that plans help keep even simple projects on course and provide something against which I can track progress.

Figure 2-4 suggests a template for a strategic SPI plan, which guides the organization's long-term improvement activities. This strategic process improvement plan template is included with this handbook. The organization's process improvement leader typically is the prime author of this plan. Members of the SEPG and MSC should review and approve it. Adapt this template to serve your own SPI planning needs. The various sections of this template provide guidance on how to complete it. Section 6 on risks and barriers is particularly important. Be sure to identify the possible obstacles you might encounter and ponder how you might remove them.

Each working group you convene to address a specific process area should write a tactical action plan using a standard template, which is included with this handbook. The action plan describes what the group intends to accomplish and how. It should identify:

- The business or technical goals to be accomplished, which should trace back to the overall goals expressed in the strategic SPI plan
- Measures that will indicate whether the process changes are yielding the desired effects
- The organizational scope of the process changes
- Working group participants, their roles, and their time commitments

1. **Purpose**
   1.1. Acronyms
   1.2. Reference Documents
2. **Overview**
3. **Executive Summary**
4. **Business Case for SPI Initiative**
   4.1. Business Rationale
   4.2. Guiding Principles
5. **Process Improvement Goals**
   5.1. Short-Term Goals
   5.2. Long-Term Goals
6. **Assumptions, Risks, and Barriers**
7. **Organization for Process Improvement**
   7.1. Organizational Scope
   7.2. Management Steering Committee
   7.3. Software Process Improvement Leader
   7.4. Software Engineering Process Group
   7.5. Working Groups
   7.6. Process Owners
   7.7. SPI Consultant
8. **Criteria for Success**
9. **Improvement Agenda**
   9.1. Selection of SPI Activities
   9.2. Current SPI Activities
   9.3. Process Improvement Roadmap

Figure 2-4: Template for a strategic software process improvement plan

- How often and to whom the working group will report status, results, and issues
- Any external dependencies or risks
- The target completion date for all activities
- A list of action items, identifying for each the individual owner, date due, purpose, activities, deliverables, and resources needed

Limit the number of action items in each plan to nine or ten specific, fairly small actions. This will help you scope the working group's efforts to a couple of months, rather than being a life sentence for the participants. When the group fully implements one action plan, you can always move on to the next area that's ripe for improvement.

See Trap #4, stalling on action plan implementation, in Chapter 4.

Steer to Success

Process improvement leadership is a steering function, guiding the organization first to accept that better practices are needed and then to successfully adopt new practices. This steering requires stable goals and constancy of purpose. Ever-changing improvement objectives confuse and frustrate practitioners, who throw up their hands and say, “Tell me when you figure out what you really want.” Avoid being distracted by a quest for higher CMM maturity levels or process certifications, instead focusing on improving your business results by selectively and creatively applying the guidance from existing process frameworks. Track the time people actually spend on SPI activities to see if the planned work is really getting done and whether your current resource level is realistic for your objectives.

Recognize the reality of the learning curve discussed in Chapter 1, the short-term performance drop that results as you learn new work methods and incorporate them into everyone’s personal processes. It will take time for individuals to internalize new and better ways of working, and for the organization as a whole to institutionalize new methods as routine practice. The time and money you spend on SPI is a strategic investment in the long-term success of your organization, and those resources aren’t available for immediate project use. Take satisfaction in small victories and celebrate your successes. Keep your teams trying to do their project work better tomorrow than they did it yesterday and you’ll all come out ahead.

Practice Activities

1. Complete Worksheet 2-1.
2. Complete Worksheet 2-2.
3. Complete Worksheet 2-3.
4. If you’ve held some type of assessment of your current processes (or if you otherwise have insights into specific process issues), complete the strategic process improvement plan template provided with this handbook. Describe the results you want to obtain from your SPI effort and how you plan to approach it.
5. Select one of the specific process areas you want to work on and complete the action plan template to define how a working group would approach the improvement activities. These plans should be fairly simple, not highly detailed.

Worksheet 2-1: Your Management Steering Committee

Identify candidates for your Management Steering Committee, what you expect them to contribute, and what attitude they're likely to have toward the SPI program.

Name	Contribution Expected	Attitude

Worksheet 2-2: Your Process Owners

Identify people who might be willing and able to serve as process owners for the various process areas that are important to the success of your projects. See Table 5-1 in Chapter 5 for some possible process areas to consider.

Process Area	Possible Process Owner

Worksheet 2-3: Your Software Engineering Process Group

Identify appropriate candidates for your Software Engineering Process Group. These could be the names of individuals or they could be project or organizational roles (such as a project's QA manager, or an experienced technical lead). Describe the role you expect each participant to perform. One should be the leader of the process improvement program, another might take the lead in performing internal process assessments, a third might head up a metrics program, some might be responsible for certain process areas or for liaising with certain sectors of your organization, and so on.

Name or Role	Percent Time Participation	Likely Responsibilities

Chapter 3. Personal Process Improvement

I sometimes meet software developers who don't think they can change the way they work unless their manager approves. I've also met people who believe they don't need to improve the way they work unless their management implements a formal process improvement effort. These attitudes both miss the mark. As professionals, we're all responsible for continually learning and applying new techniques that improve our productivity and the quality of what we produce. You don't need a manager's permission to pump up your own software development knowledge, skills and practices. Nor should you need a manager to demand that you change. Learning, growing, and improving is one of your job responsibilities.

Many developers don't work in an organization with a management-driven process improvement program. Perhaps you're surrounded by process skeptics who succumb to the relentless daily project pressures, never investing energy in making tomorrow's project go better than today's. What's a lonely process enthusiast to do?

You can take several actions to improve your personal software engineering approach and perhaps your workgroup's processes, even without official management sanction or leadership. You'll eventually hit a glass ceiling, a limit beyond which you can't influence other people. Nonetheless, anything you do to upgrade your own capabilities will save time, reduce rework, and improve your productivity. The Personal Software Process provides an aggressive and formal approach to enhancing an individual software engineer's capability (Humphrey 1995; Humphrey 1996). Not everyone is ready for the time commitment and rigor the PSP demands, though.

This chapter identifies many possible areas you could consider for improving your own performance (Wiegers 2000). You can't tackle all of them at once, even if they all pertain to you. Pick out one or two areas that you want to work on during each project or life cycle phase. Think of it as growing your bag of tricks, your professional tool kit. It will take you some time to learn and apply these new practices but you don't need a lot of money for tools or training. Selectively adopt some of these practices as personal development habits and set an example for your fellow team members to follow. When you achieve better results than you did before, other people will notice.

Project Planning and Tracking

Even if your job title isn't project manager, you doubtless need to plan and track the work you do. For example, most software team members are asked to prepare estimates for their work but few are skilled estimators. Someone once asked me, "If we're supposed to use historical data for preparing estimates, where do we get historical data?" The answer is, "If you write down what you did today, then tomorrow that is historical data." It's not more complicated than that. To get better at estimation, begin by recording your estimates (schedule, effort, budget) for individual tasks and the actual results. Comparing the two sets of numbers and understanding the differences will help you improve your estimates. You can also develop estimating heuristics based on counts of program elements such as requirements, classes, methods, or user interface screens and widgets. Only data and experience can help you produce estimates better than those you'll get from sticking a wet finger in the wind. Using data also helps you justify a project timeline to management. They might not like the schedule, but estimates based on empirical evidence are easier to defend than those based on guesses.

To further enhance your personal estimation skills, create planning checklists for common project activities such as implementing a class, executing a system test cycle, or building a product from the components stored in your configuration management system. The checklists will help you avoid overlooking necessary tasks, a common cause of underestimation. Update the checklists as you gain experience. I use simple planning checklists to help me remember tasks to perform and to track when I planned to complete them and when they were really done. I reuse these lists the next time I perform a similar project or activity. Figure 3-1 illustrates such a checklist, the one I used when creating this handbook. I based it on the one I used for my earlier *Project Initiation Handbook*.

Because no one really spends 40 hours per week working on an assigned project, keep track of how you actually use your time. Record the hours you spend every day on each development or maintenance activity phase. Don't try to account for every minute, but discover how many effective project hours you actually have available in an average week. Several groups I've been in tracked their time in this way. The results are sobering. The best we ever did was to account for an average of 31 hours of project time per week (Wiegers 1996a). Use this information to help you translate your task effort estimates (in labor-hours) into calendar time estimates. Remember to leave time for quality activities, meetings, your other responsibilities (like SPI activities!) and the inevitable rework.

As another project management tip, maintain an issues list to make sure that you track each one to closure, rather than leaving loose ends when you think you've completed an activity. Perform a retrospective with your team members when you complete a project or phase to understand what went well and what could be improved (Kerth 2001). Additionally, identify risk factors you can anticipate and try to control on future projects.

Requirements

Individual developers generally have little control over the software requirements they receive. Strive to form collaborative relationships with those who supply your requirements, be they marketing staff, dedicated business analysts, managers, or actual users. Ask to review a set of requirements before you commit to implementing them. During the review, carefully examine each functional requirement to determine whether it contains enough information for you to design and implement it. Look for ambiguities and vague language that would lead to subjective interpretations. These weaknesses greatly increase the chance that you'll fall short of customer expectations. Watch out for these fuzzy terms: support, such as, and/or, etc., including, several, many, user-friendly, simple, typical, standard, usual, fast, robust, flexible, efficient and improved (Wiegers 2003). You can find checklists for reviewing software requirements specifications, design documents, and other typical software development work products at http://www.processimpact.com/pr_goodies.shtml.

If you aren't sure of the intent behind a requirement, ask for clarification. Point out that you aren't sure what the customer needs and you don't want to keep questioning people or to rebuild the system if your first guess turns out to be wrong. Look for possible exception conditions that weren't addressed. Consider whether you could verify that each requirement was correctly implemented through testing or some other approach. As a prime 'customer' of the requirements documentation, your critical eye will spot problems with requirements that an analyst or customer likely will not see.

Status	Est. Finish	Actual Finish	Task
			1. Write introduction
			2. Write 7 chapters
			3. Develop handbook layout and design
			4. Design cover page
			5. Accumulate process assets
			6. Review and edit handbook myself
			7. Have entire handbook reviewed by others
			8. Revise after review
			9. Add marketing page in back
			10. Final proofread and edit
			11. Create release PDF
			12. Create release zip with handbook and process assets
			13. Establish pricing
			14. Develop description and marketing materials
			15. Develop TOC and sample chapter for web site
			16. Set up PayPal mechanism
			17. Backup current versions of web site files
			18. Install new files on web site (index, goodies, products, order_form, handbooks, pubs, /handbooks files)
			19. Remove related articles and process assets from web site. (SPI so hard, PI that works, personal PI, SPI traps, good practices, molding CMM, MMA)
			20. Proofread web page changes
			21. Advertise to my contact list
			22. Make backup CD of handbook materials

Figure 3-1: A sample planning checklist

You might find that the requirements documents you receive from marketing or the business analyst don't meet your needs. They could be poorly organized, lack essential information, or include spurious material that distracts you from the key content. Too often, critical documents that are intended to bridge the interface gap from one community to another are written only from the document author's point of view. But it's far better to write those key bridging documents from the perspective of the *consumer* of the information. Marketing might produce a requirements document that seems reasonable to them but has too many shortcomings to be useful to the downstream consumers of the document. It might be missing a clear statement of the product's scope and limitations, lack information about the product's intended usage, or describe conflicting quality characteristics.

Work with the people who supply you with such key documents to agree on a document template that meets development's needs. Explain why a different document structure or content will accelerate development by reducing false starts and rework. This is the first step toward a collaboration that will make those critical documents as concise and valuable as possible.

A change-control mechanism is one of the highest-yield processes any software team can implement. Don't work on proposed requirements until they've been approved by the decision-makers, baselined, and allocated to a specific product release or build number. If customers request some surreptitious changes, politely steer them to the project's defined change-control process. This helps ensure that the right people makes sound business decisions to incorporate the right changes, which improves your team's chance of meeting its commitments.

Design and Coding

The hard part of programming is thinking, not typing. Take the time to design your software before you implement it, to avoid having to build it over and over again to meet functional and performance needs. Having never produced an optimum design on my first try, I learned long ago the value of iterating my designs. Refactoring an existing design can add value, but it's no substitute for doing some design work in the first place. Learn about, adopt, and share with your colleagues some standard ways to represent designs. The Unified Modeling Language or UML is the current standard modeling notation (Booch, Rumbaugh, and Jacobson 1999), but the classical structured modeling techniques still have their place.

Models are communication mechanisms. Avoid the temptation to invent your own design notations. Using established conventions facilitates effective communication; contributing to the software Tower of Babel does not. Lead your group to agree on the design notations you'll use and the tools that will help you iterate your way to robust designs.

Get to know your programming tools. Really understanding the capabilities and limitations of your language, editor, compiler and other tools can improve your efficiency and effectiveness, enhancing your ability to develop high quality, industrial-strength code.

All programmers think their coding style is the best. Indeed, why would they continue to follow it if they knew of a better way? However, we can all learn from the masters and improve our own programming approaches. Read some of the classic programming books, such as Steve McConnell's *Code Complete, 2nd Edition* (Microsoft Press, 2004), Jon Bentley's *Programming Pearls, 2nd Edition* (Addison-Wesley, 1999) and Donald Knuth's three-volume *The Art of Computer Programming, 3rd Edition* (Addison-Wesley, 1998). Apply the guidance in these books to develop—and consistently follow—sensible coding standards and sound construction techniques. As a simple example, I modified my commenting style after I spotted a better approachin *Code Complete*. I knew my old style was awkward to maintain, but I liked the way it looked. Finding a better way encouraged me to confront the shortcomings in my current technique and overcome my intrinsic resistance to change.

Quality Practices

Each developer is responsible for the quality of the work he does . Unfortunately, few developers are adequately trained in the arts of quality analysis and measurement, testing, and technical review. Read a book on software testing, such as Brian Marick's *The Craft of Software Testing* (Prentice Hall, 1997) and actively apply the information to your own work. Decide what you mean when you say you are finished testing a program. Are you simply out of time? Did the program survive whatever tests you tossed at it? Or did you consciously define an acceptable quality level and select the quality filters and testing coverage that will let you reach it?

Effective unit testing is another crucial skill. I once assisted a programmer who had spent three weeks testing a program but hadn't recorded a single one of his tests. We had to start the testing all over because we didn't really know what he'd done and we couldn't repeat it.

Simply thinking of tests often reveals errors in the code, even before you execute the program and sometimes even before you write the code. Test-driven development is a well-established practice that's enjoyed renewed popularity thanks to the agile methodology movement (Beck 2002). You can document your tests in the form of code comments to make them repeatable. Develop automated test harnesses to accelerate regression testing, such as test functions or methods that are invoked through conditional compilation. When you modify a program, execute these regression tests to ensure that nothing else broke. Document your integration and system test cases and procedures to avoid having to redevelop them in the future and to enable someone else to properly test your programs if necessary. As defects are discovered, add new test cases to help you confirm each fix and grow a robust regression test suite.

Use quality tools to scour your code for as many potential problems as you can. At the least, use static code analyzers such as Lint (public domain for various languages or Gimpel Software PC-Lint and FlexeLint for C/C++), Compuware DevPartner and Parasoft's CodeWizard to find subtle defects the compiler and a casual visual scan might miss. A developer once told me that Lint reported 10,000 errors and warnings when his team ran their program through it, so they never used Lint again! The ostrich approach doesn't lead to better software.

Even if you don't use a static code analyzer, respect and correct compiler warnings and errors. The best programmers I've ever known taught me to make the compiler as fussy as possible and listen to everything it tells you. Your development team members should agree to use the same compiler settings. If something's wrong with your code, fix it now when it's cheap, rather than later when it's painfully expensive.

Another class of quality tools is the dynamic code analyzer, exemplified by IBM Rational's Purify, Compuware BoundsChecker, and Parasoft's Insure++. These tools can reveal run-time errors such as memory leaks, writing past the end of arrays, and assorted pointer problems. Tools won't detect missing requirements or logic errors, but they can help you avoid a lot of debugging time and user grief.

Peer reviews are a valuable technique that can improve the quality of any work product (Wiegers 2002). Find some colleagues you respect professionally and trust personally, and get in the habit of looking over each other's requirements, designs, code, and tests. Take a class on software inspection[2] and introduce your team to these formal reviews. Many people complain that reviews consume too much time. While they do take time, anyone who has experienced the review benefits of finding defects efficiently will never want to return to programming in isolation. Reviews don't waste your time—defects do.

Adopt good check-in/check-out configuration management discipline, keeping source code as well as other key project documents under version control. When you check in a module, record the reason for the changes you made when the tool prompts you; don't just hit the space bar to save a few seconds of typing. Document and automate the build procedures for your system so anyone on the project can correctly build the product. Get in the habit of frequent, even daily, builds of the growing product, using automated tests to reveal defects quickly.

"Discipline" is the key to effective configuration management. I keep records of all the changes I make in my web site, www.processimpact.com. Sometimes I'm tempted to skip the recordkeeping, thinking that I'm just making a little change in a file or two. But I take the few mo-

[2] A comprehensive eLearning course titled "Software Inspections and Peer Reviews" is available from http://www.processimpact.com/elearning.shtml.

ments to backup the existing files, install the new ones, test them, and update the records. This discipline helps me deal with web site problems because I know exactly when I installed which new files.

Maintenance

Maintaining ill-structured, undocumented, and fragile legacy systems isn't fun. The mystery is why so many developers continue to create low-quality software for their colleagues—or themselves—to maintain in the coming years. Developers, managers, and customers often balk at taking the time to document software, yet they complain bitterly about how hard it is to modify existing applications and the distraction that maintenance poses to new development work.

Take a stand against future maintenance nightmares. The time you spend documenting your designs and commenting your code will be amply repaid when you or your successors perform system surgery in the future. Clear documentation that matches the code enhances your ability to modify the program, whereas documentation that conflicts with the code is useless or even misleading. An old army maxim says that when the map and the terrain disagree, you should always believe the terrain. If the code and its comments do not agree, the maintainer is forced to believe the code—the terrain—even if it's incorrect.

You might find yourself in a maintenance black hole, lacking essential system information. Without reliable documentation, the maintainer must reverse engineer the code every time he needs to make a change. While it's rarely cost-effective to fully document an existing system, avoid digging the black hole any deeper. Document what you learn through reverse engineering so you or a coworker need not rediscover that knowledge in the future. Record what you deduce about a legacy system's interfaces to the rest of the world. The incremental cost of capturing the knowledge is small compared to the cost of discovering it.

As you add new functionality, model it using your design conventions and show where it connects to the existing application. This is helpful even if you don't have complete models of the entire system. Create test cases to verify the new code and modify existing test cases as necessary. Introduce changes at the highest level of abstraction that each change affects. For example, you might document a functionality change in the requirements specification, then cascade the change into affected design elements, code, test cases and other artifacts. Create a portion of a requirements traceability matrix to capture these logical links. Simply diving into the code to implement a new feature creates disconnects between the code, the requirements, and the design. It can result in a brittle system that's less amenable to future change.

Try to leave maintained code in better shape than you found it, but avoid the temptation to rewrite a module just because you don't like its style or structure. I fell into this trap once and wasted several hours before I caught myself. However, if you find yourself making many corrections in a defect-riddled module, consider rebuilding it to reduce future maintenance effort.

A valuable process improvement is to keep records of defect reports and enhancement requests through a standard change-control process. You can start such a process on your own and then share it with your teammates to establish a consistent way to manage changes on your project. A simple spreadsheet might suffice to start recording defects, but for the maximum benefit, use one of the many available defect-tracking tools to support the process. Commercial configuration management tool suites usually include a defect-tracking component. Remember, though: a tool is not a substitute for a process.

Track the time you spend dealing with each change or fix to understand the true cost of quality shortfalls or overlooked requirements on your project. While some change is legitimate and unavoidable, excessive change requests and problem reports can indicate inadequate requirements development, poor programming, or testing deficiencies. Ferret out the causes of wasteful project rework to spot you improvement opportunities for future attention.

I once suffered a spate of "bad fixes" when performing maintenance hastily. Either I failed to fix the defect properly or I introduced new defects in the process (or both). Track the percentage of your maintenance fixes that aren't done correctly and understand why. My bad fix causes included:

- Failing to think enough about the problem
- Addressing a symptom rather than the underlying disease
- Inadequate regression testing
- Just plain sloppy programming when rushing to get something done.

Use those insights to improve your checklists that identify the steps to perform when implementing a change. Set a goal to reduce your bad fix rate by, say, 50 percent in the next three months and track your progress toward that goal.

A true commitment to process improvement means you never stop learning and growing. You're always working on a few selected areas that you know would give you better results. It takes time to internalize new ways of working, to effectively apply unfamiliar techniques and accept them as better than your traditional approaches. Once you've adopted improved practices, never let your boss, your customers, or your colleagues talk you into doing an inferior job.

Practice Activities

1. Complete Worksheet 3-1.

Worksheet 3-1: Your Personal Process Pulse

Use this worksheet to assess your own process discipline. If you're a software developer, rate how frequently you perform each of the activities listed; they might not all apply to you. Think of this as a micro-process assessment, which you could expand to cover other practices that are essential to the success of the individual software engineers and development teams in your organization.

Question	Rarely or Never	Some-times	Usually	Always
1. Do you carefully review the requirements specification before implementing it?				
2. Do you avoid adding cool new functionality no one asked for?				
3. Do you create explicit design models or documents?				
4. Do you hold design and code reviews?				
5. Do you track how you spend your project time?				
6. Do you record your task estimates?				
7. Do you compare actual task outcomes to the estimates and learn from the difference?				
8. Do you follow a consistent coding style standard?				
9. Do you use Lint or other code analysis tools?				
10. Do you use run-time code analyzers to find memory leaks and other problems?				
11. Do you plan your unit tests systematically?				
12. Do you write down your unit tests?				
13. Do you track defect reports systematically?				
14. Do you improve your development process based on the defects you discovered?				
15. Do you have documented and automated build procedures?				
TOTAL POINTS:				

Scoring: Give yourself 0 points for each Rarely or Never answer, 1 point for each Sometimes, 2 points for each Usually, and 3 for each Always.

35-45 points:	Serve as a software engineering mentor for others in your team.
25-35 points:	Adopt the discipline to consistently apply the practices you now do part of the time.
15-25 points:	Pick out two practices you don't currently perform, learn about them, and try them – next week.
0-15 points:	There are probably many points of pain in your software development daily life. Start addressing the items on which you scored the lowest, beginning next Monday.

Chapter 4. Process Improvement Traps to Avoid

I once knew a man who worked in the software engineering process group for a large telephone company. Cal's group had been working diligently on a CMM-based software process improvement program. One day the new senior manager decided to abandon the CMM effort and pursue ISO 9001 registration instead. The members of the SEPG were demoralized. They'd invested a great deal of effort in their CMM activities, only to discard most of what they'd accomplished and start over in a different direction. Lack of management commitment and continuity has decapitated a lot of SPI programs.

Software process improvement efforts can be derailed in many ways, leaving the members of the organization jaded, frustrated, and more committed than ever to the ways of the past. This chapter describes ten common traps that can undermine an SPI program (Wiegers 1996b). Other chapters pointed out where some of these traps can bite you. Learning about these process improvement killers, their symptoms, and some solutions will help you prevent them from bringing your initiative to its knees. Keep in mind, though—none of these solutions will work if you're dealing with unreasonable people. That's not a software problem.

Trap #1: Lack of Management Commitment

Symptoms: While individual groups can improve the way they do their work through grassroots efforts, sustainable changes across an organization require management commitment at all levels. Senior managers might claim to support process improvements (how can they say otherwise?), but they might not really be willing to make short-term sacrifices to free up the resources required for the long-term investment. Larger organizations often lack alignment between senior management and one or more layers of mid-managers, who feel immediate pressure to ship products.[3] If you're leading the SPI effort, you might obtain senior management commitment but get pushback from those mid-level managers. In this case you'll be forced to spend time and energy debating the importance of software process improvement with people who should only have to be educated, not sold.

Such mixed signals from management make it hard for team leaders and software developers to take the effort seriously. Watch out for lip service and buzzwords masquerading as commitments. Project managers who assign their least capable people—or none at all—to SPI are providing a clue that the lack of management commitment is about to foil your effort.

Solutions: Managers at all levels need to send consistent signals about process improvement to their constituencies. Executives must be educated about the costs, benefits, and risks so they'll have a common vision and understanding of this domain. Commitments need to be aligned along the

[3] Here's a question those mid-managers might ask: "What's currently preventing us from shipping products as fast as we'd like?" The answer might well indicate that some process improvement would be a big help.

organizational hierarchy so managers aren't working at cross purposes and a reluctant manager can't sabotage the effort through inaction. In one corporate division with a highly successful SPI program, the senior manager had a portion of his salary at risk, dependent on SPI success. Lower-level managers had smaller portions of their compensation at risk. Money gets people's attention.

I often hear people say, "I can't make this change without management support." Management "support" often translates into permission or acceptance, not leadership. I draw a big distinction between management *support* for something and management *commitment* to that same activity. Figure 4-1 illustrates some strong signs of management commitment to SPI. Make sure that commitments from management translate into resources devoted to SPI, realistically defined expectations of the SEPG and the engineering staff, and accountability for the results.

Management commitment to also affects the morale and dedication of SPI leaders. When management objectives change with the wind and the staff devoted to facilitating process improvement is downsized, those affected might be embittered at having months or years of their technical careers sidetracked for nothing. Once burned, they'll hesitate to step forward the next time the organization seeks volunteers to lead a change initiative.

1. Providing the resources and time to develop, implement, and sustain an effective SPI program.
2. Setting policies, expectations, and goals for the SPI program.
3. Establishing policies, expectations, and goals for the processes that are adopted.
4. Expecting the new processes to be used even when projects are under time pressure.
5. Ensuring that project schedules include time for process improvement work.
6. Making training available to the team members.
7. Holding team members accountable for applying new processes in good faith.
8. Publicly rewarding the early adopters of new processes to reinforce desired behaviors.
9. Running interference with managers and customers who challenge the need for changing how the organization works.
10. Asking for status reports on the SPI program, what it costs, and its benefits.

Figure 4–1: Some signs of management commitment to process improvement

Trap #2: Unrealistic Management Expectations

Symptoms: Excessive enthusiasm by ambitious managers also poses risks. If the goals, target dates, and results expected by managers aren't realistic, the process improvement program is set up for ultimate failure. Managers, particularly those with little software experience, might not appreciate the effort and time involved in a large-scale SPI effort. These managers could be confused about how process improvement frameworks like the CMM or CMMI relate to other software engineering approaches, such as the IBM Rational Unified Process or an agile development methodology. They might focus on issues of pressing importance to them that are not realistic outcomes of the process improvement effort. A senior manager once told me that he hoped to deal with current staff shortages by driving the organization to reach CMM Level 2, which typically leads to higher software productivity and quality. However, since it can take a year or more to reach Level 2, this isn't a solution to near-term staffing problems.

Management needs to understand that they can't mandate or buy the behavioral changes and organizational infrastructure that are essential parts of process improvement. Catchy slogans like "Level 5 by '05" or "Six Sigma by '06" are not constructive. In an unhealthy competitive environment, process improvement can become a contest: Department A sets an objective of achieving CMM Level 3 by the end of 2006, so the head of Department B says they can do it by the *middle* of 2006. This kind of competition is rarely inspiring or motivating.

Solutions: Educate your managers to help them to understand the realities of what a serious SPI initiative will cost and what benefits they might expect. Collect data from the software literature on other companies' results and the investments those companies made over a specified time period. Every organization is different. It's risky to promise an eight-fold return on investment just because some other company actually achieved that. Use data from the literature or from other areas of your own company to help your managers develop realistic expectations and set reasonable, even ambitious, goals.

Trap #3: Time-Stingy Project Managers

Symptoms: When a senior manager says he's committed to improving the organization's software processes, most project managers will agree, whether they mean it or not. However, successful SPI requires project managers to adjust their schedules so team members can devote some time to improvement activities. A project manager who claims to believe in SPI but who treats it as a burden added on top of the project work is sending conflicting signals.

Even if team members are allowed to work on improvement tasks, these tasks often get low priority. "Real work" can easily squeeze process activities out of a busy engineer's schedule. Project managers might respond to today's schedule pressure by curtailing the effort that should go into process upgrades.

Solutions: You need consistent, active commitment at *all* management stages. A bottleneck anywhere in the hierarchy can bring the process program to a screeching halt. One way to achieve consistency is through a corporate policy of interlocking management commitments. Senior managers publicly state their goals and priorities, including SPI. Lower-level managers align their goals and priorities with those of the higher-ups.

Senior management must make it clear that they'll evaluate project managers on the effectiveness of their process improvement activities, as well as on project success. Project plans and commitments need to account for the staff that are being devoted to developing the new processes. In small organizations with shallow management hierarchies, the first-line manager is the most critical factor in the success of any SPI effort. If this person doesn't make process improvement a visible priority, it just isn't going to happen.

One way to keep a program viable is to treat all SPI activities as mini-projects, to give them the visibility and legitimacy they need for success. Write an action plan for each mini-project, as discussed in Chapter 2. This plan identifies resources, states timelines, itemizes deliverables, clarifies accountability, and defines measures to evaluate new processes. Don't try to solve every process problem at once. Concentrate on the two or three top priority items, as determined through a process assessment, then tackle the next few, and so on down the line.

Project managers can't just assign their least effective people to the software process improvement efforts, either. We all want our best people working on technical projects. However, if good people and respected leaders involved, the working group outputs will carry less weight with the rest of the team.

Trap #4: Stalling on Action Plan Implementation

Symptoms: Action plans might be written after a process assessment, but little progress is made on them because management doesn't make them a clear priority, assign people to work on them, or otherwise take them seriously. Managers might never mention the action plans after they're written. This signals team members that achieving improved processes really isn't important. The lack of progress on improvement plans is frustrating to those who actually want to see progress made. It also devalues the investment made in the process assessment.

Solutions: As with Trap #3, a good way to turn action plans into actions is to treat improvement activities as mini-projects. Focusing on just a few improvement areas at a time avoids overwhelming the project team. You need to measure progress against the plans, and to measure the impact of each action plan on the business results achieved. For example, a plan to improve unit testing effectiveness might include interim goals to acquire test automation tools and train developers in their use. It's easy to track progress on these interim goals. The desired business outcome of such an action plan should be a specific quantitative reduction, over some period of time, in the defects that slip through your unit tests.

If your project managers never make much progress against their action plans, a management oversight function might encourage them to take the plans more seriously. In one organization, all project managers had to report the status of their action plans every three months to a multilevel management steering committee. No one wanted to be embarrassed by reporting meager action-plan progress at his quarterly review.

From one perspective, such periodic reporting reflects appropriate management accountability for the commitments people have made to improve their software processes. From another, though, this is a "big stick" strategy for enforcing SPI. This is best avoided unless teams simply aren't implementing the action plans. Your culture will determine the best techniques for driving action plans to completion. The management oversight approach did achieve the desired results in that one organization.

Trap #5: Achieving a Maturity Level Becomes the Primary Goal

Symptoms: Organizations that adopt the CMM framework for process improvement (Paulk 1995) risk viewing maturity level attainment as the process improvement goal. In reality, this is just one mechanism for achieving the organization's real business goals. The focus is on a race to the next maturity level, even though some energy should be devoted instead to other areas that can contribute to the organization's quality, productivity, people, and management issues.

Sometimes, a company is in such a rush to reach the next maturity level that it doesn't take time for the recently implemented process changes to become well established habits. The organization can actually regress to the previous maturity level, rather than continuing to climb the maturity ladder. This demoralizes practitioners who are eager to enhance their software engineering culture.

Solutions: In addition to aiming at the next CMM level, align your SPI effort with corporate business and technical objectives. Mesh the process work with any other improvement initiatives that are underway, such as ISO 9001 registration, or with an established software development framework already in use. Recognize that advancing to the next CMM maturity level can take a year or more. It's not feasible to leap from an initial ad hoc development process to a super-sophisticated engineering environment in one fell swoop. Your goal is not to be able to chant, "We're Level 5! We're Level 5!" Your goal is to develop improved software processes and more capable developers so your company prospers by delivering better products faster than before.

Use measurements to track progress toward the business goals, as well as measuring the progress of the process improvement program. Goals can include reducing project cycle times and product defect levels. One way to track SPI progress is to perform low-cost interim assessments to check the status of your project teams in various CMM key process areas, such as requirements management and project planning. See Chapter 7 for suggestions about how to perform these assessments. Over time, you should observe steady progress toward satisfying both CMM goals and your company's software success factors. This is the outcome of a well-planned and well-executed program.

Trap #6: Participants Are Inadequately Trained

Symptoms: I once taught a software seminar at a company that was undergoing an ISO 9001 registration audit that very day. I asked the 20 students in the class how many of them were affected by the ISO program. They all raised their hands. Then I asked how many of them had been trained on ISO 9001. Not a single person had received training. How are they supposed to know what ISO 9001 means to them and how they're supposed to behave and work in a way that's consistent with the ISO 9001 expectations?

A process improvement initiative is at risk if the developers, managers, and process leaders lack adequate skills and training. Each person involved must understand the general principles of software process improvement, the CMM and other pertinent SPI methodologies, change leadership, software measurement, and related areas.

Inadequate knowledge can lead to false starts, well-intentioned but misdirected efforts, and a lack of apparent progress. This undermines the improvement effort. Without training, the organization's members won't have a common vocabulary and a shared understanding of the need for change. They won't know how to properly interpret the improvement model being followed. For example, "software quality assurance" means different things to different people. Training helps to give everyone a common understanding.

Solutions: Many process improvement consultants and training companies offer courses to support established process improvement frameworks. Alternatively, you can either develop such training yourself or license courseware from a commercial provider. Different participants in the process improvement activities will need different kinds of training. If you're using a CMM-based approach, the SEPG members should have two to three days of training on the CMM and how to apply it practically. However, four hours of training about process improvement using the CMM will be enough for most participants. If you become serious about SPI, consider acquiring training in other key aspects of process improvement: managing change, setting up a software engineering process group, establishing a metrics program, performing process assessments, and action planning.

Trap #7: Expecting Defined Procedures to Make People Interchangeable

Symptoms: Some managers might expect that having repeatable processes means that every project can achieve the same results with any randomly assembled set of team members. They might think that the existence of a defined process in the organization makes all software engineers equally effective. They might even believe that working on software process improvement means they can neglect technical training to enhance the skills of their software developers and managers.

Solutions: Individual programmers have been shown to have a 10-to-1, 20-to-1, or even wider range of quality and productivity performance (DeMarco and Lister 1999). Processes alone can never overcome such a large range of individual capability. You can partially close the gap by expecting people to follow effective defined processes, rather than using whatever methods they are used to. This will enable people at the lower end of the capability scale to achieve consistently better results than they might get otherwise.

You'll always get the best results by combining the most talented, capable people with the best known ways of working. Never underestimate the importance of attracting, nurturing, and rewarding the best software engineers and managers you can find. Aim for software success by creating an environment in which all team members share a commitment to quality and are enabled—through superior processes, adequate training, appropriate tools, and effective team interactions—to reach their peak performance.

Trap #8: Failing to Scale Processes to Project Size

Symptoms: A small organization can lose the spirit of the CMM (or any other process model) while attempting to apply the model to the letter. They can generate excessive documentation and formality that actually impedes project work. This undermines the credibility of SPI, as team members bypass the official procedures in an attempt to get their work done. People don't want to perform tasks they perceive as adding little value to their project.

Solutions: To reach a specific CMM maturity level, you must demonstrate that your organization is satisfying all of the goals of each key process area defined for that maturity level and for lower levels. Your processes should be no more complicated or elaborate than is needed to satisfy these goals. Nothing in the CMM says that each procedure must be lengthy or documented in excessive detail. Strive for a practical balance between documenting procedures enough to enable repeatable project successes and getting project work done with the minimum process overhead.

This nondogmatic view doesn't mean that smaller organizations and projects can't benefit from the discipline provided by the CMM. It simply means the CMM practices should be scaled rationally to the size of the project. A 20-hour project should not demand eight hours of project planning just to conform to a CMM-compliant "documented procedure." I once knew a project manager whose organization was intent on following the dictates of the CMM to the letter (as they interpreted it). The SEPG sent a consultant to help develop a project management plan for the project team. This consultant spent *four months* creating a project management plan for a six-month project. That kind of disproportionate effort is what gives "process" a bad name in some organizations. Your working groups should provide scaleable processes, procedures, and templates that can work for the various sizes and types of projects your group undertakes.

Trap #9: Process Improvement Becomes a Game

Symptoms: Yet another way process improvement can falter is when the participants pay only lip service to the real objective of improving how they work. It creates the illusion of change while actually sticking with business as usual. The focus is on making sure a process audit is passed, not on changing the organization's culture for the better. Software process improvement looks like the current flavor of the month. The practitioners just wait for this latest fad to pass so they can get back to working in their old familiar ways.

Solutions: Focus on meeting organizational and company objectives with the help of improved software processes. Do not simply try to conform to the expectations of some established process framework. It's not enough to create documented procedures just to satisfy the letter of the framework. You must also satisfy its spirit by actually following those procedures in your daily work.

The CMM talks about *institutionalizing* process improvements, making new practices routine across the organization. Practitioners must also *internalize* improved procedures, becoming comfortable enough with the new processes that they wouldn't consider going back to the old way. It takes time to assimilate new ways of working, but it happens. My friend Nick works for a large company that has experienced much process improvement success. There are several CMM Level 5 organizations in this company. Nick once asked a member of one of those Level 5 organizations about her process. She replied, "We don't really follow a process. This is just how we do our jobs." The people in the Level 5 organization internalized the new ways of working such they were now second nature.

As a process change leader, identify the behaviors you would expect to see throughout the organization in each improvement area if superior processes are both internalized and institutionalized. As a manager, your team members need to understand that you're serious about continually improving the way they build software; the old methods are gone for good. Improvement isn't a one-shot game we play so someone can fill out his checklist properly.

I know of one organization with a successful SPI program that defined some of the changed behaviors expected of project managers and mid-managers. These definitions helped the managers to understand their role in the change initiatives. They could also see how their behaviors and expectations would influence the behaviors of their team members. Figure 4-2 identifies some management behaviors with respect to requirements development.

1. Take responsibility for defining product vision, project scope, and other business requirements.
2. Expect user, functional, and nonfunctional requirements to be documented.
3. Ask to see the documented requirements.
4. Participate in requirements reviews as appropriate.
5. Commit to the position that requirements must be approved and baselined before implementation of those requirements begins (except for prototyping).
6. Do not commit to the customer's or marketing's cost and schedule requests until the project team has negotiated and committed to them.
7. Expect that the requirements change control process will be followed routinely.
8. Ask to see evidence that project plans and commitments are based on requirements.

Figure 4–2: Desirable management behaviors for requirements development

Trap #10: Process Assessments are Ineffective

Symptoms: If process assessments (often led by the SEPG) don't have adequate participation by the software development staff, they turn into fault-finding audits. This can lead to a lack of commitment, buy-in, and ownership of the assessment findings by the project team. Assessment methods that depend solely on the responses to a CMM-based questionnaire can overlook important problem areas that should be addressed. Outside "experts" who purport to identify your group's process weaknesses without practitioner input won't have credibility with your technical staff.

Solutions: Process change is a cultural change. The ease of this cultural change depends partly on how involved the team is with the assessment and action planning activities. Include a free-form discussion with a representative practitioner group as part of the assessment. This discussion can identify problem areas that weren't covered by an assessment questionnaire but which are worth addressing. For example, CMM Level 2 doesn't address software testing, but if poor testing practices are hurting a Level 1 project, you should do something about that soon. Topics might come up in a project team discussion that aren't part of the CMM at all, including management or organizational issue, a lack of money for buying tools, and so forth.

Use your SEPG to actively facilitate the change efforts of your project teams, not just to audit their current process status and report a long, depressing list of findings. Identify process liaisons or champions in the project teams to work with the SEPG during the assessment. Those process liaisons can help drive effective changes into the project team's behaviors and technical practices after the assessment. The project team must understand that the SEPG is doing software process improvement *with* the members of the project team, not *for* them or *to* them.

As you chart a course to improve your software process capability, beware of the many minefields lurking below your organization's surface. Your chances of success greatly increase if you watch for the symptoms that identify these traps as a threat to your software process improvement program and deal with them right away. Process improvement is succeeding at many companies. Make yours one of them, by controlling these risks—and others—as well as you can.

Practice Activities

1. Complete Worksheet 4-1.

Worksheet 4-1: Your Process Improvement Traps

Think about which of these traps might pose a threat to your software process improvement program. Look for symptoms that indicate that you're caught in a trap or that it could do some damage in the future. Then, think about what you can do to avoid getting caught in that trap or to extricate yourself from it if you're already snared.

Trap	Are You Caught Yet? Is It a Looming Threat?	Ways to Escape or Avoid the Trap
#1		
#2		
#3		
#4		
#5		
#6		
#7		
#8		
#9		
#10		

Chapter 5. Creating a Process Assets Library

Many years ago I managed a small software group that was enthused about increasing its process capabilities. We developed a small set of procedures that helped us consistently do a good job on various aspects of our projects. I offered a copy of our procedures to another software manager, suggesting that much of what we had developed would be useful to his team. His response was, "Thanks, but I don't have time to read it." My feeling was, "You don't have time NOT *to read it." The time he'd spend assimilating our experience and adapting it to his own team's work would be amply repaid through greater efficiency. Another software manager also declined our offer of a copy of the procedures. "My team would rather write their own," he said. But they never did. By rejecting our offer, they missed a free jumpstart on improving they way they work.*

Software developers hate paperwork. As organizations improve their software development and management processes, though, they typically find they would benefit from some better project and process documentation. More structured development life cycles might present practitioners with new expectations about activities to perform and deliverables to create. To enable successful process improvement, you need to develop a supporting infrastructure of *process assets* and a place to store and share them—a process assets library or PAL.

Process assets include policies, process descriptions, procedures, standards, guidelines, forms, templates, checklists, and other work aids. These items help team members understand the expectations embedded in the new approaches. They also provide a mechanism for sharing throughout the organization the best known ways to perform key activities such as developing requirements, estimating project effort and duration, and managing changes.

The contents and structure of a "good" document are not intuitively obvious. A collection of document templates and actual examples will help each project to create better documents more efficiently. By adopting some common templates, similar work products developed on multiple projects will have similar structure and contents. Each new project doesn't have to invent a requirements specification or configuration management plan from scratch, perhaps omitting important sections along the way. When I see that some section of the template is empty, it reminds me to ask questions I might otherwise overlook. Many people learn from examples. Good examples of key deliverables collected from previous projects give developers a model to follow for their own project's documents. These examples can also save time through reuse of boilerplate text, common definitions, and the like.

This chapter describes a project I led at a large company to create a repository of useful software engineering project and process documents (Wiegers 1998b). This process assets library is being used by a wide variety of software projects, thereby leveraging the investment made in developing it. I describe the contents and structure of the PAL, as well as the way we organized and ran the project that created it.

Motivation and Philosophy

Many departments in this company had already launched process improvement activities, mostly using the CMM for Software as a guide. Several had accrued their own collections of example work products and templates. A large central collection of document examples had been assembled several years earlier but it wasn't used much. It existed only in hardcopy form, which made it hard to share the contents across the company. It was difficult to search the repository for specific items. The contents varied in quality, since no filtering or revision was done prior to including submitted items. Also, the contents addressed the needs of only a certain subset of the company's development projects.

From a practical point of view, you should invent new processes and procedures only as a last resort. Instead, look for existing ones to adopt or adapt to meet your needs. We wished to leverage the procedures already created by organizations across the company to make our process improvement activities more efficient. There's also a cultural issue at work here. Unlike the people in the story at the beginning of this chapter, practitioners must be receptive to items from outside sources. No team has time to invent every process artifact on its own. Plenty of public domain or commercially-available starting points already exist. One source is the IEEE Software Engineering Standards Collection (http://www.computer.org/cspress/CATALOG/st01121.htm). Many of these standards include recommended templates for important project documents, such as a software requirements specification, quality assurance plan, or project management plan.

It's harder to find suitable publicly available work product examples. Most companies view their project deliverables as proprietary, although the Internet does contain some examples from government projects. Even if you find a decent example, it probably won't conform to whatever template you've adopted for that type of deliverable. Your goal is to provide sample documents that your practitioners can use to see how they might complete the various sections of the template for their own project. You'll probably need to start with local examples from previous projects and replace them with better examples as teams generate them over time.

To take best advantage of the available resources, we decided to build a corporate-wide, intranet-based collection of software engineering "good practices." This central resource would reduce the effort needed by individual departments to create development procedures because they'd would have starting points. The collection also would house many good examples of plans and other work products created on previous corporate projects. Following a pilot project to explore the concept and evaluate approaches, I was asked to lead the project to implement the repository.

Best Practices or Good Practices?

People have been developing software for more than 50 years. During that time, a number of software engineering and management practices have emerged as being strong contributors to project success. These are known as *best practices*. It behooves all project managers and software practitioners to become familiar with the established set of best practices in their domain.

Best practices are collected by industry gurus and consultants who study many projects, looking for patterns that correlate with success and failure. One such group was called the Airlie Council. In the mid-1990s, the Airlie Council identified 16 best practices for software engineering and software management (Brown 1996; Brown 1999). Other best-practice collections come from authors who synthesize insights from the vast body of software engineering literature. An excellent example is Steve McConnell's fine book *Rapid Development* (McConnell 1996). McConnell describes 35 best practices, their potential benefits, their effects on project schedule and risk, how to apply each practice, and the risks associated with each one.

Some people don't care for the term "best practices," protesting that it implies a global or absolute standard of excellence. Certainly, practices are context-dependent. Few techniques can be stated universally to be superior in every situation. Also, "best practices" begs the question of who determines what is best and what the reference points are.

We elected to use the term "good practices" instead of "best practices." We fully expected to improve the initial collection of documents as projects gained experience and created ever-better examples. Thus, the definition of "best" is a moving target; how do you know when you're there? Since "the best" can be the enemy of "the good," we focused on building a collection of useful—but not necessarily perfect—items that would still be better than the resources presently available.

Our Approach

We assembled a small part-time team of interested participants to carry out this project. Participants included members of a software engineering process group that supported a large product development division, a webmaster, and others who had a strong interest in quality and process. Our initial strategy was to:

- Identify the appropriate contents of the collection.
- Devise a scheme for organizing and labeling the many documents we expected to encounter.
- Assemble small volunteer teams of subject matter experts (SMEs) in various software practice domains to evaluate, revise, and create suitable documents.
- Develop a suitable Web architecture for making the many documents accessible.
- Write a detailed and realistic project plan.
- Follow the plan to create and populate the repository, adjusting the plan as needed.

A fundamental project philosophy was that "less is more," in two ways. First, we wanted to include just a few documents of each type. If you have a dozen sample requirements specification templates from which to choose, you're more likely to create your own than to study all 12 and pick the one most suitable for your needs. Therefore, we included just a few templates and examples of each major document type. These represented approaches that were suitable for different kinds of projects, such as small information systems and large embedded systems products.

The second "less is more" notion led us to trim actual project documents down to a manageable size that simply illustrated the structure, contents, and style. If you're looking for an example to mimic, you don't need to wade through all of someone else's project-specific details to get the idea. We applied considerable editorial license to modify candidate document examples to meet our goals of size, detail, and layout.

We began by assembling a shopping list of the kinds of items we thought would be valuable to include. The list combined items referred to in the CMM, activities and deliverables that were defined by the product development life cycle followed by many projects, and documents that support solid software engineering approaches. The project team also identified candidate items we might harvest from the existing local collections throughout the company.

This initial shopping list identified well over 200 different process asset types. Some of these clearly were more important than others, so we prioritized the list into three levels. Amazingly, about one-third of the items ended up in each priority level, which made it much easier to focus on doing the important things first. Lacking infinite resources and time, we realized that we'd never acquire some of the lowest priority items. The initial version of the Web site included 90% of the high priority documents, 44% of the mediums, and only 20% of the lows. This demonstrates that we did a good job of investing our effort where it could yield the greatest benefit.

The project team also applied sound requirements engineering practices to this project (Wiegers 2003). We began by brainstorming a list of use cases, each describing one objective a prospective user of the PAL might be trying to achieve. I've found the use case approach to be valuable for other Web development projects, as they help the project team focus on the things a user would be able to do at the site. Some of these use cases were:

- Find a template for a particular deliverable that's appropriate for a certain type of project.
- Find a CMM-compliant procedure for a specific activity.
- Download forms or other work aids for a specific activity, such as a code inspection.
- See what training classes are available for a specific software practice area.

We concluded that 20 of our 22 use cases were in scope for this project. We addressed four of these by providing hyperlinks from the Good Practices Web site to other sites, such as the company's on-line training catalog. The remaining two use cases were out of scope and we didn't address them. Yes, we could envision that someone might want to perform those tasks with this resource, but they were beyond the charter and funding we had available for the initial release.

We also modeled the data architecture of the Web site using an entity-relationship diagram and data dictionary. We modeled the user interface architecture using a dialog map, which shows the various proposed Web pages and the navigation pathways among them (Wiegers 2003). Figure 5-1 shows a portion of this dialog map. Each box represents a web page and the arrows represent built-in navigation pathways between pages. This modeling allowed us to simplify the Web site and improve its usability before we implemented it. Finally, we developed a series of evolutionary Web site prototypes, which our teams of subject matter experts evaluated. The final prototype revision became the initial production release. These solid requirements and design practices helped us develop a usable and robust resource that was well received by our user community of more than one thousand software developers and managers.

The Good Practices project team delegated much of the evaluation, improvement, and even creation of artifacts to the SME teams. SMEs came from the development, management, process improvement and quality engineering ranks. Without these active volunteers, we'd probably still be working on the project.

Good Practices Contents

To make it easy for users to locate what they needed, we developed three searchable catalogs that listed all of the PAL contents. We organized one catalog by CMM key process area, as many departments in the company were using the CMM to guide their process improvement activities. Another catalog was based on the activities and deliverables defined for each phase of the product development life cycle being followed by dozens of projects in various organizations. We identified available procedures and checklists for the activities, and available templates and examples for the deliverables.

We organized the third catalog by the software engineering practice areas shown in Table 5-1. Every document in the collection had a unique identifier made up of the practice area to which it

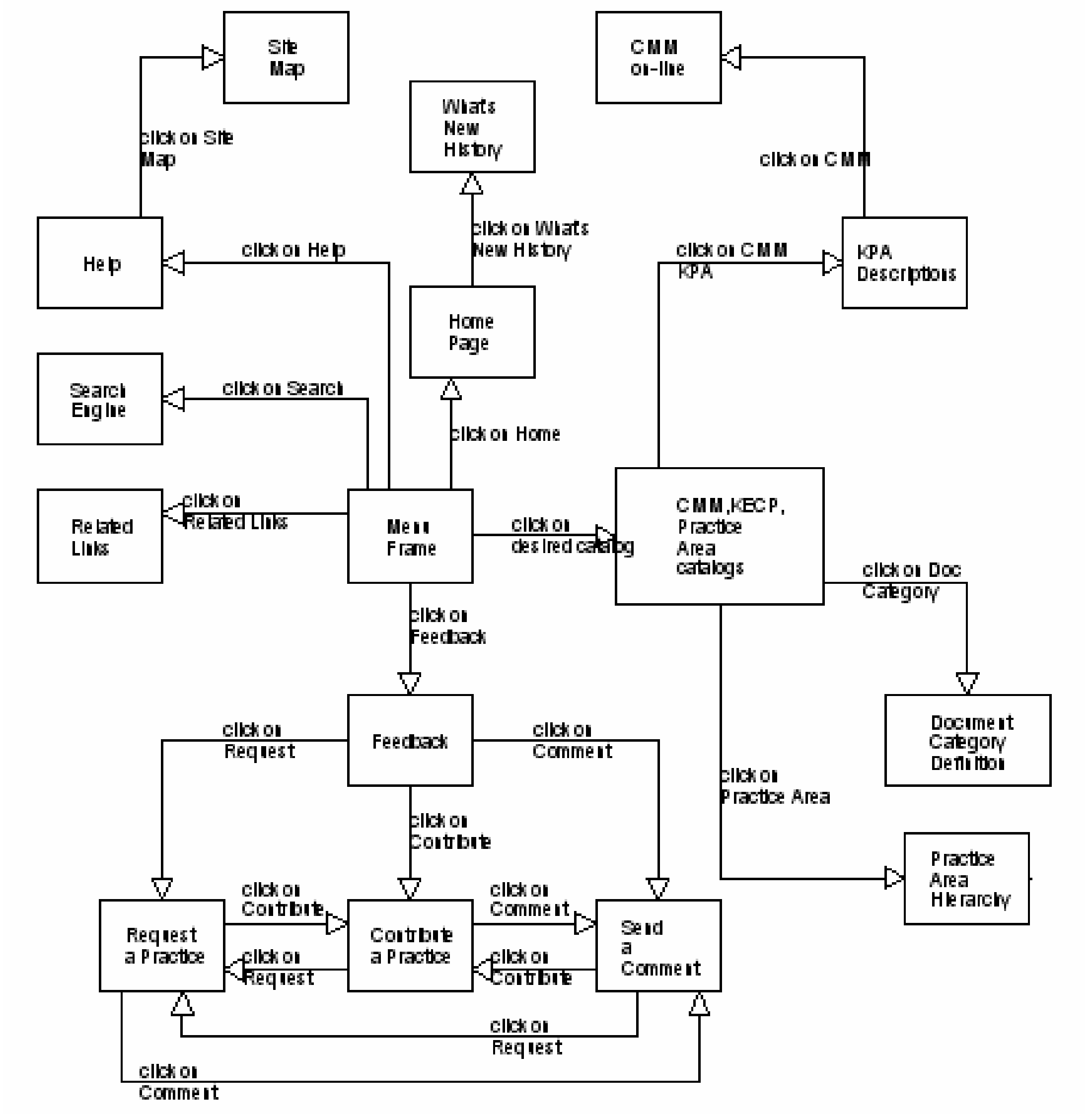


Figure 5-1: Portion of dialog map for Good Practices Web site

belonged, a document type code (e.g., work product example, procedure, template), and a sequence number. For example, DES.ARCH.EX.2 was the second example we had of an architecture design specification.

By scouring the company, we found good examples for most document categories. Many of our best examples came from projects or organizations that already had a well-established software process improvement program. To plug some holes in the collection, we selected items from a commercial source or adapted them from the IEEE software engineering standards.

The publication process for preparing documents for installation on the intranet was quite elaborate. We create a master copy of each document in Microsoft Word. Documents intended to be read-only were converted to Adobe Portable Document Format (PDF). Templates intended to be downloaded and adapted were published on the Web site in both Word and Adobe FrameMaker format, which many projects were using. We wrote a detailed procedure for the many steps involved in

Table 5-1: Software Engineering and Management Practice Areas

Practice Area	Subpractice Areas
Design	Architecture Design Detailed Design User Interface Design
Implementation	Coding Integration
Maintenance	(none)
Project Management	Estimation Project Planning Project Tracking Risk Management
Requirements Engineering	Requirements Development Requirements Management
Software Configuration Management	Configuration Auditing Configuration Control Configuration Identification Configuration Status Accounting
Software Quality Assurance	Auditing Metrics Peer Reviews
Software Subcontract Management	Accepting Subcontracted Materials Proposals and Contracting Tracking Subcontract Activities
Testing	Beta Testing Integration Testing System Testing

the publication process. These steps might include OCR scanning of hardcopy documents, adding descriptions, editing for size or confidentiality, and reformatting.

Tracking the status of this project was not trivial. We had to monitor the evaluation and publication status of about 300 separate items, using a series of linked Microsoft Excel worksheets. For those candidate documents that we accepted, we tracked the following milestone dates:

- Accepted for inclusion
- Permission to include it received from the owner
- Converted to Word
- All edits completed
- Converted to its ultimate delivery format
- Delivered to the webmaster
- Installed
- Tested

We also tracked the number of documents that we delivered each month in each practice area category and at each priority level. Additionally, we tracked the monthly time that all project participants (including subject matter experts) spent on the project and the time spent on the many document conversion operations of various kinds. These metrics provided a complete and accurate picture of the project's cost, achievements, and status at all times.

Web Site Architecture

As illustrated in Figure 5-2, each software engineering practice area from Table 5-1 has a separate Practice Area Web Page. This describes the practice area in general and provides information about, or links to, available training classes, tools, pertinent books, local subject matter experts, and other useful resources. Each Practice Area Page lists the document categories available for that practice area. For example, the Requirements Management practice area includes document categories for software requirements specification, interface specification, requirements management procedure, requirements traceability matrix, and others. Each document category page provides links to specific process documents. Each document has a short description to help the user select items that are most pertinent to his needs and project type.

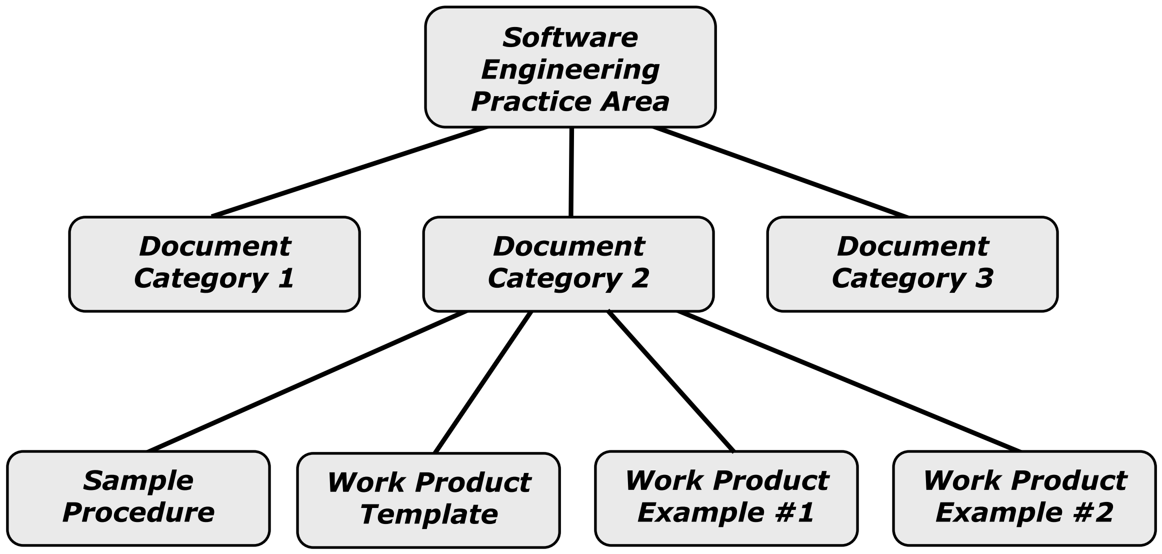


Figure 5-2: Practice area Web page hierarchy

Lessons Learned

At the conclusion of the project, the team recorded some lessons learned from the experience to assist future such projects. Many lessons came out of things we did right; we learned others because we missed the best approach the first time around. These lessons fell into three categories:

Doing It Right

- ◆ Develop use cases to focus scope.
- ◆ Develop multiple prototypes.
- ◆ Provide prototype evaluation scripts to the users.
- ◆ Document procedures for publishing documents on the Web.

Controlling the Project

- ◆ Plan the project carefully.
- ◆ Build a shopping list of documents needed.
- ◆ Set—and respect—three levels of work priority.
- ◆ Track document and project status rigorously.
- ◆ Review the risk list periodically.
- ◆ Walk through the project work breakdown structure as a team to spot missing tasks.
- ◆ Record action items and decisions made at meetings.

Stability Over Time

- ◆ Gain commitment from webmasters who own pages to which we link to inform us if those links change.
- ◆ Test all external links periodically as part of the project maintenance plan.
- ◆ Review and improve contents periodically to keep them current and useful.

Process improvement projects that rely on the volunteer labor of participating co-workers face a major risk: this commitment might not float to the top of a busy developer's priority list. If a SME team was not getting the job done, we had no choice but to replace certain members with people who would contribute effectively. We also decreed that a decision-making quorum for any meeting was whoever showed up, which kept the project rolling along.

This project met most of its major milestones and the initial release was delivered on schedule and well under budget. During the following six months, hundreds of developers, quality professionals, and process improvement leaders downloaded more than 3,500 documents, suggesting that software teams found this to be a valuable resource.

It's difficult to quantify the return on investment for such a project. However, this consolidated process assets library has clearly made both the process improvement activities and the development projects at this company more efficient. An on-line "Good Practices" collection is a vital part of any process improvement initiative.

Practice Activities

1. Complete Worksheet 5-1.

Worksheet 5-1: Your Process Assets Library

List items you might wish to add to your own process assets library because they'd be useful to your project teams. Prioritize them to indicate which would be most valuable. Note the items for which you think you'll be able to find local examples to refine and share with the rest of the organization.

Subpractice Areas	Candidate Artifacts and Priority	Possible Sources
Architecture Design		
Detailed Design		
User Interface Design		
Coding		
Integration		
Maintenance		
Estimation		
Project Planning		
Project Tracking		

Subpractice Areas	Candidate Artifacts and Priority	Possible Sources
Risk Management		
Requirements Development		
Requirements Management		
Configuration Auditing		
Configuration Control		
Configuration Identification		
Configuration Status Accounting		
Quality Assurance Auditing		
Metrics		
Peer Reviews		

Subpractice Areas	Candidate Artifacts and Priority	Possible Sources
Accepting Subcontracted Materials		
Proposals and Contracting		
Tracking Subcontract Activities		
Beta Testing		
Integration Testing		
System Testing		
Unit Testing		

Chapter 6. Molding the CMM to Your Organization

I once had a debate with another member of my SEPG. We disagreed on how we should apply the CMM to our improvement effort. Should we follow the CMM right down the line, or should we be more flexible? "The CMM is a model," she insisted. "We're supposed to apply it as it's written." My reply was, "The CMM is a model. We should adapt it to suit the nature of our projects, organization, and objectives." Which philosophy do you think is correct?

Religious wars wage continuously in the software industry. One such conflict is over the merits of the various Capability Maturity Models, such as the CMM for Software and its successor, the more practical CMM Integration. Inside the CMM fortress are hundreds of organizations that have found the CMM to be a useful tool for helping them improve the quality, timeliness, and predictability of their software products. Besiegers arrayed outside the walls loudly protest that their organization wasted tons of money fruitlessly attempting to do the CMM thing, that the CMM emphasizes dogmatic perfection, or that the CMM doesn't apply to them for any number of reasons.

As in many wars, both sides have a point. Using the CMM as a framework for software process improvement has clearly helped many companies. Others have struggled with the CMM, trying to follow its guidance but achieving less than stellar results. The wise process improvement practitioner will keep eyes and ears open to what those around him have tried, to avoid rediscovering painful lessons. In this chapter I present a case study of how a large software division in a very large company adapted the CMM to best meet its needs (Wiegers 1998a).

Why Not Just "Do the CMM"?

The CMM for Software is published in the form of a 440-page book (Paulk 1995). Simply declaring to your software community that you're going to follow the CMM generates immediate resistance: "I can't do all that." "I don't understand it." "That doesn't apply to me." "Get out of my office."

Concise improvement objectives like "Level 3 in 3 [years]" are appealing but they might not be right for your organization. A better approach is to respect the wisdom embedded in the CMM, but adapt it to your organizational culture, needs, and realities. Study the key process areas of the CMM, especially those at Levels 2 and 3, and think about the activities and practices that will help *your* organization achieve its business objectives. Focus on obtaining "Level 2 results"—predictability, stability, and visibility—more than on satisfying every detail of the CMM's expectations for Level 2.

I was a member of the SEPG that helped guide a large division's process improvement activities. This division had about 500 software engineers working on dozens of projects to create both commercial applications and embedded software for complex equipment products. Our SEPG helped managers set strategic goals that aligned with business objectives. We also provided an infrastructure of training, resources, expertise, and CMM knowledge.

We tried to steer clear of the dogmatic quest for the next maturity level, although at times this was attractive to managers who favored simple responses to complex problems. Instead, we focused on adapting the CMM's practices to address the very real software-related problems facing this organization. We found that adapting the CMM in this way made model-based process improvement more palatable than did a more rigid, dogmatic application of the CMM.

Process Assessment

Any process improvement effort should begin with some kind of assessment to establish a baseline of current practices and problem areas. One option is to perform a full organizational appraisal based on the CMM. An alternative that works well in small groups is a simple, facilitated brainstorming session (see Chapter 2). We chose an intermediate approach: We developed a flexible process mini-assessment method based on the CMM that we could apply to a wide variety of projects, which is described in Chapter 7.

Over the span of a year, our SEPG conducted about 20 of these mini-assessments on diverse projects. Some project teams were happy to see us bring more structure to their improvement activities. Other teams clearly resented our presence. This is par for the process improvement course. The mini-assessments typically addressed the six Level 2 key process areas (KPAs), plus occasionally the Peer Reviews and Intergroup Coordination KPAs from Level 3.

The deliverables from each mini-assessment included both relative strengths of the project and a list of findings that pointed out process weaknesses. We augmented each finding statement with a list of current and potential consequences of that process shortcoming. There's no point in worrying about findings derived by comparison against a reference model if the team isn't experiencing—or likely to experience—problems as a result of that finding. We also proposed several realistically achievable recommendations for addressing each finding.

We collected our findings from all these projects in a database and then looked for patterns. Certain issues recurred across multiple projects. Often, we could cut-and-paste from the findings database to generate a portion of the findings on another mini-assessment. It seemed reasonable to focus our limited improvement energy on these common problem areas, using the CMM as a guide to suggest sensible remedies for those problems.

Molding the CMM

Once we had the common process weaknesses from across the organization in hand, we scrutinized the Level 2 KPAs and picked out individual practices that we felt were worth pursuing. The selected practices had to satisfy three criteria:

1. If properly implemented, they would make the findings—and their associated consequences—go away.

2. They must be realistically achievable.

3. They could be interpreted sensibly in the context of our organization: what level of managers were appropriate for a particular CMM-specified review, for example.

Each CMM key process area describes several key technical or managerial practices ("Activities Performed") that will generally satisfy the goals prescribed for that KPA. However, the CMM also recognizes that other key practices are necessary to enable an organization to consistently and routinely achieve those goals. These practices are found in components of each KPA called Commitment to Perform, Ability to Perform, Measurement and Analysis, and Verifying Implementation. We examined all of these CMM components for each KPA and chose the practices that were both pertinent and realistic for our division's.

CMM maturity levels are achieved by demonstrating that specific sets of KPA goals are being satisfied. It's not essential to perform all of the key practices defined for each KPA. The practices we selected generally aligned with satisfying those KPA goals, but that alignment was secondary to the ability of the practices to address the division's known process shortcomings.

We documented exactly what an effective application of each selected KPA practice would mean in our organization. These individual "evaluation criteria" included a statement of the observable reality if the practice has been successfully applied, and also whether this observable was an action or a deliverable of some kind. Table 6-1 illustrates some of the evaluation criteria we developed for the Requirements Management KPA. The evaluation criteria set out the expectations of what should be different if our organization succeeded in following the CMM practices we adapted.

Next, we determined what would constitute suitable evidence that each selected practice was being applied as intended. This verification evidence should be simple, realistic, and objective. I've found that responses to questionnaires about current process performance are often overly optimistic. People are tempted to give themselves generous partial credit, "rounding up" in their favor. The verification methods did not address *who* would be doing the evaluation or the exact procedure to use, thereby leaving latitude for doing whatever makes the most sense on each project.

A fundamental philosophy behind our approach was that we didn't worry about whether our organization would achieve Level 2 even if every project satisfied all the evaluation criteria. That wasn't our paramount concern. What was extremely important was to make sure that the practices we selected really would rectify some known shortcomings in the way this division currently ran its software projects. Being "substantially Level 2" was fine with us.

Table 6-1: Examples of Evaluation Criteria for Requirements Management

Item	Activity or Deliverable	How Verified
A documented procedure for developing a software requirements specification has been adopted and followed.	D	See procedure; compare SRS to procedure
Software requirements are documented, with a preliminary version at Gate 1 and a baselined version at Gate 2.	D	See two versions of SRS at Gates 1 and 2
The baselined software requirements document is under version control and change control.	D	See that SRS versions are identified; change procedure is in place
Team has been trained in requirements development and management, through course 6307 or equivalent.	A	Review training records

Tracking Progress

Tracking progress in software process improvement is not trivial. The ultimate indication of success is that the team is working in new ways that yield superior results. Results-oriented measurements around quality and productivity are lagging indicators of process improvement success.

An alternative is to use activity—not results—measurements, with the faith that if we do the right things, eventually we should get better results. However, we need finer-grained tracking than doing a full-scale process assessment every couple of years to see if we've reached the next CMM maturity level. Our evaluation criteria permitted just such tracking. Each KPA has an average of 21 individual evaluation criteria. We could set goals in terms of the percentage of evaluation criteria that are satisfied for each KPA, and we could track progress toward those goals.

We judge each evaluation criterion in a binary way: either satisfied or not. However, there is still opportunity for evaluator judgment as to what constitutes satisfaction. For example, certain evaluation criteria involve sending the whole project team to a specific training course. For a team of 20 people, how many have to be trained to satisfy this criterion? All 20? Just 10? How about 13, or 18? We expect the evaluators to be reasonable when making such judgments. Formulaic approaches to process improvement cannot replace common sense.

What We Learned

The notion of adapting the CMM to fit the needs and realities of our organization was not uniformly accepted. As the story at the beginning of this chapter illustrated, we encountered resistance from one member of our SEPG. She felt we were attempting to reinvent the CMM, wasting time and confusing practitioners in the process. My opinion, though, is that the CMM is simply a model or framework that recommends ways to address the kinds of problems that affect many software organizations. Practitioners are expected to apply this model in a sensible, relevant, and practical way, which means adjusting it to fit local situations and objectives.

We also received resistance from some project teams. Their initial reaction was that our approach was still too CMM-ish, and there were far too many items in the evaluation criteria for them to take seriously and address. "We'll go through your list and cut it down to the size that makes sense for us," one team said. "Fine," we replied. So they took a close look. As I predicted, they kept every single one of the evaluation criteria our SEPG had proposed!

This outcome reinforces my contention that you should apply most of the practices in the CMM because they represent sound approaches to software development, not simply because they're in the CMM. Even if the CMM was never conceived, these practices describe things most projects ought to be doing. It's easy to feel overwhelmed when confronted with the entire CMM. When examined item by item, though, you won't find much that a reasonable developer or manager will pooh-pooh as silly or useless. If some practices don't fit your world, don't do them!

As with most things in life, neither extreme position is sensible. Don't throw out the CMM simply because some parts that don't apply to you and other important elements are missing. But neither should you apply it dogmatically without considering how best to make it work for your organization.

I'm confident that this approach represents a sensible way to apply the CMM to a large and diverse organization. By first getting a handle on the real problems and issues facing the projects, we can thoughtfully judge how selected elements of the CMM can help us better achieve our technical and business objectives. Focus on what the experience embodied in the CMM can do for you, rather than on racing for the next maturity level.

Practice Activities

1. If you're using the SW-CMM or the CMMI as a guide for your process improvement program, think about how to pragmatically modify or adapt it to your organization. Identify the key process areas that pertain to your projects. Determine which key practices will be of greatest value, the problems they'd address, and the business or technical benefits they would yield. Define your own evaluation criteria, the evidence that would indicate whether the projects were applying the practice in the intended fashion.

Chapter 7. A Modular Mini-Assessment Method

In 1995 I participated in my first full-scale, CMM-based software process assessment of a corporate division with about 60 information technology staff. This was a major activity, involving seven of the company's process improvement specialists plus an outside consultant for ten days. The assessment was educational but exhausting. I have since done many assessments of one or two project teams at a time by myself. I find that I learn most of the important issues within a single day of interviews, discussions, and document review. After that, additional input just provides corroboration and further details about what I've already learned. You don't need a huge, formal process appraisal to spot your improvement opportunities.

Organizations often launch software process improvement initiatives with a comprehensive process appraisal, such as the CMM-Based Appraisal for Internal Process Improvement or CBA-IPI (Dunaway and Masters 1996). However, such appraisals are expensive and time consuming, so many companies find it difficult to perform them frequently.

Several organizations have created small-scale assessment techniques to take the process pulse of a software organization between—or instead of—full appraisals. Motorola developed a progress assessment instrument that defined criteria an organization can use to evaluate its performance for the approach, deployment, and results achieved in a given CMM key process area (Daskalantonakis 1994). The Interim Profile technique, developed jointly by the SEI and Pacific Bell, relies primarily on the CMM maturity questionnaire[1] to develop a process profile indicating KPA satisfaction (Whitney, et al. 1994). The Norad System Support Facility uses a "spot check" approach, in which each activity described for a KPA is evaluated as to how effectively it is being conducted (Wakulczyk 1995).

Many software development organizations at Eastman Kodak Company used using small-scale assessments to stimulate SPI at the project level, to track progress toward higher maturity levels, and to assess an organization's readiness for a full-scale appraisal. This chapter describes a flexible, modular mini-assessment method that enables construction of a customized activity sequence to meet the needs of an individual project (Wiegers and Sturzenberger 2000).[2] I've included several tips for readers who are interested in implementing their own mini-assessment methods.

A process assessment itself does not lead to any benefits—it leads to knowledge. An assessment is part of the investment an organization makes in software process improvement. If you don't make beneficial process changes after the assessment, you wasted this investment. A lack of action also makes the participants skeptical about the organization's true commitment to SPI.

[1] A maturity questionnaire asks questions to learn whether, or how frequently, specific process activities are performed on the project.

[2] Other contributors to the mini-assessment method were Linda Butler, Jeff Duell, Ron King, Jeff Perdue, Dave Rice, and Marsha Shopes.

Background

Over time, different Kodak departments had developed three distinct mini-assessment approaches. While their objectives were similar, they differed in the maturity questionnaire used, the steps involved, and the time commitment by both assessors and project team members. One method was essentially a two-day miniature version of a full appraisal. Another variant required about 12 hours of contact time spread over several sessions, including an eight-hour session to hold a practitioner discussion and generate findings by consensus. In a third, very compressed approach, one to three project representatives completed a maturity questionnaire by consensus in a facilitated session. The assessors then generated findings by identifying performance gaps in all CMM key practices in the KPAs covered.

Members of Kodak's SPI community wished to develop a common method that used a standard set of tools and procedures, yet accommodated the current methods where possible. Our objective was to construct a mini-assessment architecture that we could tailor to each project's improvement objectives, life-cycle status, team size, and time constraints. Standard procedures and tools would make it easier to bring new assessors up to speed. They would also facilitate collaboration among assessors from different departments. The method had to yield consistent and reliable results. We successfully applied the modular mini-assessment method, or MMA, described here on many projects in several organizations.

Tip 1: Determine the objectives of your SPI initiative and select assessment techniques that are aligned with tracking your progress toward those objectives.

Method Description

Figure 7-1 illustrates the overall process flow for the MMA. We can combine these steps in various ways to reduce the number of separate events. The mini-assessment doesn't include follow-up action planning and action-plan tracking activities. Members of the SEPG that support the assessed project's organization often facilitate these essential steps, but they are ultimately the project's responsibility.

The principal data-gathering methods used in the MMA are responses to a process maturity questionnaire and an optional project participant discussion. All team members who participate in the MMA must have one to four hours of CMM orientation. No managers above the project manager level are involved in data gathering. We use a confidentiality agreement to make it clear that all data and findings are private to the project team and that we won't attribute any data to individuals. The MMA is designed to use two assessors (lead and backup), although a single assessor can perform several steps effectively, which reduces costs.

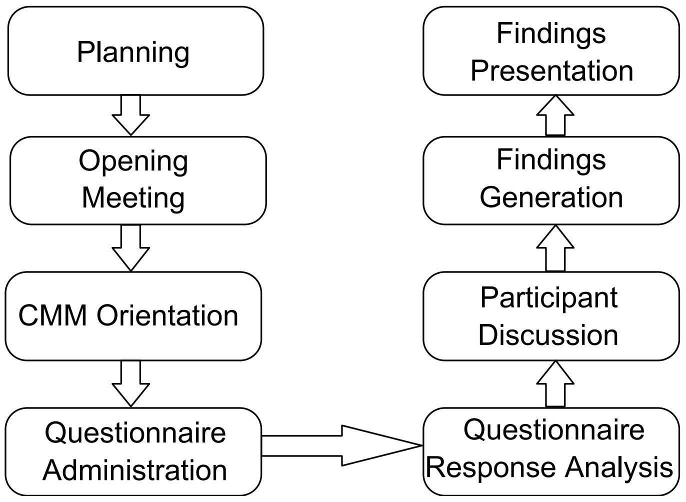


Figure 7-1: The modular mini-assessment process flow

Tip #2: Decide which data collection methods will afford the appropriate balance between objective and subjective input on both the process activities the project team performed and the results the team achieved.

Although the MMA is based on the CMM, it was not specifically designed to comply with the CMM-based appraisal framework (Masters and Bothwell 1995). Consequently, a mini-assessment cannot yield an official maturity level rating. We used the method primarily to initiate and sustain SPI activities. Identifying appropriate improvement opportunities was more important to us than were maturity level ratings.

The MMA's flexibility comes from the multiple options available for most assessment steps. The selections that are made affect the number of meetings held and their durations. The time required for a mini-assessment ranges from two to 16 contact hours per participant, depending on the options chosen. Table 7-1 shows the options available for each MMA component.

Planning

After a project decides to have a mini-assessment, the SEPG manager assigns two SEPG members to be the assessors. These assessors meet with the project manager (and perhaps the project's quality assurance manager) to plan the activities. The assessors describe the mini-assessment process and collect information about the project and the team members. The project manager conveys his expectations for the mini-assessment experience, and the assessors state their expectations of the project participants.

Table 7-1: Mini-Assessment Component Options

Component	Options
Opening Meeting	1. Separate event 1. Brief lead-in to questionnaire session
CMM Orientation	1. 15-minute refresher 2. 1-hour briefing on SPI and CMM 3. 4-hour course
Questionnaire Administration	1. Questionnaire used: A. Practices, subpractices, some institutionalization B. All CMM key practices C. Institutionalization factors only. 2. Questionnaire administration mode: A. Each participant completes a questionnaire B. One set of consensus responses
Participant Discussion	1. No discussion 2. Discussion on selected KPAs 3. Discussion on any process-related issues
Findings Generation	1. Assessors generate off-line 2. Participants generate with assessor facilitation
Findings Presentation	1. Assessors present findings to project team 2. Project team presents findings to their management

The assessors stress that the mini-assessment is only the first step on the path to improved process capability. The real work consists of planning and implementing actions to address shortcomings identified in the project's current processes. If the project manager balks at following through on action plan implementation, we question whether it's worth performing a mini-assessment at this time.

Tip #3: Periodically during the mini-assessment, explain how the assessment results relate to the follow-up steps that will need to be taken. This will help build momentum for the post-assessment SPI activities.

Next, the assessors present typical mini-assessment objectives, and the project manager rates each of these as being of high or low priority; "medium" is not an option. The pattern of high-priority objectives helps the planning group select appropriate assessment activities to meet the objectives. Possible objectives are to:

- Identify process strengths and improvement opportunities
- Educate the team on SPI and the CMM key process areas
- Serve as a catalyst for improvement
- Prepare for a formal CMM assessment
- Obtain team buy-in to the importance of software process improvement
- Identify software engineering best practices being used

The planners select the specific components that will make up this mini-assessment, choosing from the options in Table 7-1. This includes selecting the KPAs that the questionnaire selected will cover. The project manager also decides whether the entire project team or just a representative slice will participate in the assessment. The planning group then sets a preliminary schedule. The project manager identifies a team representative to serve as the process liaison between the assessors and the project team and to assist with logistics. The deliverable from planning is an agreement that summarizes the objectives, participants, and events for this mini-assessment.

Tip #4: Decide which subset of the project team should participate to provide accurate data, gain ownership of the mini-assessment outcomes, and keep costs low.

Opening Meeting

This kickoff event provides the first opportunity for the project team to hear what the mini-assessment is all about. The opening meeting can be held as a separate event (typically as part of a regularly scheduled project team meeting) or as a brief lead-in to administering the questionnaire. When it is done as a separate event, the assessors typically spend more time describing software process improvement and the organization's SPI strategy. The opening meeting provides an excellent opportunity for the project manager and higher level managers to state their commitment to the mini-assessment and their expectations for the subsequent process improvement activities.

CMM Orientation

Three choices are available for presenting some background on the CMM and SPI to the project team members. The assessors normally present a short briefing (10 to 15 minutes) prior to administering the questionnaire. This refresher is sufficient for teams that have undergone a previous mini-assessment. Projects having less CMM exposure can opt for a briefing of about one hour that gives an overview of software process improvement and the CMM. We strongly recommend that participants who are unfamiliar with the CMM take a half-day in-house course on software process improvement using the CMM prior to beginning the MMA.

Questionnaire Administration

A maturity questionnaire is an important data-gathering instrument for our mini-assessments (Zubrow, et al. 1994). The project manager can choose which questionnaire to administer in an MMA, as well as choosing who should complete it. The assessors facilitate the questionnaire administration session, using standard slides to describe the intent of each KPA before the participants answer the questions for that KPA. We adapted the basic questionnaire used from one co-developed by Kodak and the Institute for Software Process Improvement (ISPI). It addresses many key practices and subpractices of the Activities Performed common feature of each KPA, along with some institutionalizing practices from the Commitment to Perform, Ability to Perform, Measurement and Analysis, and Verifying Implementation common features. We also had available a second questionnaire that addresses only institutionalization factors and a third, composite questionnaire that encompasses all key practices of the CMM. The assessors can use any of these questionnaires in a mini-assessment. In practice, we nearly always used the first one.

All the questionnaires have possible responses that indicate how frequently each practice is performed (Always, Usually, Sometimes, Rarely, Never, Don't Know, Not Applicable). We also encourage participants to write comments on the questionnaires. Figure 7-2 illustrates a portion of the questions dealing with the Software Project Planning KPA.

The second questionnaire option is to decide whether to collect an individual questionnaire from each participant or to generate a single set of consensus responses. The consensus approach is valuable for stimulating discussion among participants and clarifying their understanding, but it's impractical if more than a few participants are involved. Individual responses (anonymous, of course) provide a broader cross-section of input from project team members. The pattern of responses can also highlight discrepancies in how the team members view the practices being used on the project. For example, on one mini-assessment of a small team with just six people, we got six different responses to one question about software quality assurance: one each for Always, Usually, Sometimes, Rarely, and Never, plus one person who honestly responded with Don't Know. This distribution of responses indicates a need for better communication within the team about how it's carrying out the project.

PP-1 Does the project have a documented software project management plan (including more than schedules or budgets)?

❑ always ❑ usually ❑ sometimes ❑ rarely ❑ never ❑ don't know ❑ doesn't apply

PP-2 Is a software life cycle with predefined stages identified or defined for the project?

❑ always ❑ usually ❑ sometimes ❑ rarely ❑ never ❑ don't know ❑ doesn't apply

PP-3 Are estimates for the size of software work products (the software itself and its documentation) derived and recorded?

❑ always ❑ usually ❑ sometimes ❑ rarely ❑ never ❑ don't know ❑ doesn't apply

PP-4 Are the estimates for the project's effort, cost, schedule, and computer resources related to the size estimates of the software work products?

❑ always ❑ usually ❑ sometimes ❑ rarely ❑ never ❑ don't know ❑ doesn't apply

PP-5 Is historical data used to support estimation?

❑ always ❑ usually ❑ sometimes ❑ rarely ❑ never ❑ don't know ❑ doesn't apply

PP-6 Are the software project management plan and estimates agreed to by the individuals and groups that will work on the project?

❑ always ❑ usually ❑ sometimes ❑ rarely ❑ never ❑ don't know ❑ doesn't apply

Figure 7-2: Sample questions from a maturity questionnaire

Tip #5: Don't expect participants new to SPI to understand the CMM well enough to accurately complete a maturity questionnaire on their own. Administer the questionnaire in a group to help all participants have a common understanding of the questions.

Questionnaire Response Analysis

The assessors analyze the questionnaire responses using a spreadsheet tool that was originally co-developed by Kodak and ISPI. The outputs from this tool are: individual question response distributions; profiles of question ratings for each KPA (see Figure 7-3); and an overall project KPA profile that indicates a satisfaction percentage for each KPA (see Figure 7-4). To compute individual question ratings, we weight the responses from individual questionnaires. An Always response receives a 1.0 weight, Usually gets 0.75, Sometimes gets 0.5, Rarely gets 0.2, and a response of Never has a zero weight. We average these weighted scores for all of the questions for a given KPA (all questions weighted equally) to compute an overall percentage satisfaction rating for that KPA. Note that this is an approximate quantification of the practices being performed, not a direct evaluation of whether the KPA goals are being achieved, which is the primary objective of a CMM appraisal.

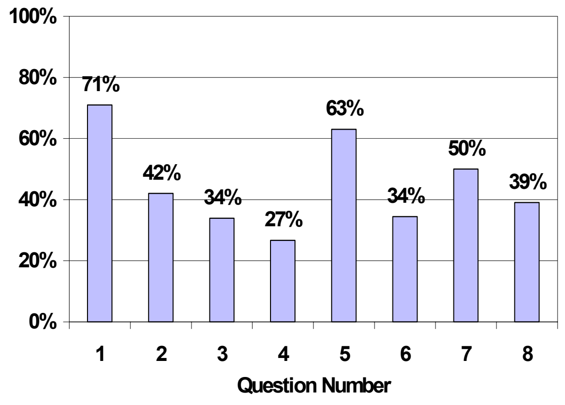


Figure 7-4: A sample response profile for one KPA

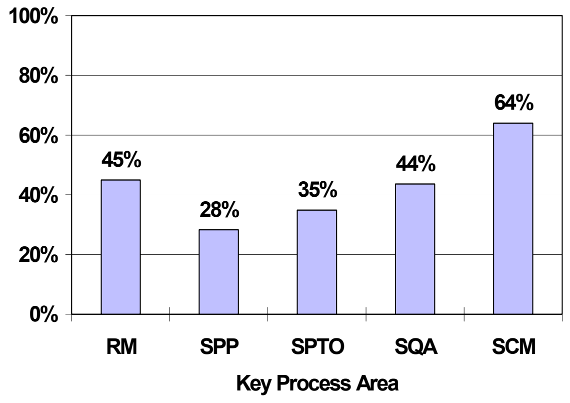


Figure 7-3: A sample KPA profile for one project's mini-assessment

Tip #6: Unless achieving a specific CMM maturity level is your overriding goal, downplay the significance of questionnaire scores and a maturity level rating. Focus instead on the improvement opportunities that the questionnaire response patterns reveal.

The assessors study the questionnaire response profiles and participant comments to compile observations about the project's practice of each KPA. If a participant discussion is scheduled as part of this assessment, these observations constitute preliminary findings that the assessors present at the beginning of the discussion. However, if no discussion is planned, the assessors craft the observations into relative process strengths and finding statements.

Participant Discussion

In this supplemental data-gathering activity, the assessors facilitate a discussion with the project team members, optionally including the team leader. Though optional, we strongly encourage holding a participant discussion if the project team can afford the time, and nearly all teams did hold one. The discussion elicits additional information that provides a more complete picture of the project's software practices. The scope of the discussion is set during the planning session. It can be limited to CMM topics, or it can cover any process-related issues that are important to the project team.

An important contribution of the discussion is to validate the questionnaire results. Assessments that rely solely on a questionnaire for input are less reliable. The participants might have misinterpreted questions or provided a distribution of responses that is difficult to interpret. Inaccurate assessment findings reduce the assessors' credibility with the project team.

To open the discussion, the lead assessor presents the observations gathered from analyzing the questionnaire responses. Then the project team selects up to three KPAs for further discussion. The process issues raised around these KPAs will be used to generate mini-assessment findings.

Tip #7: Although the questionnaire results and the assessors' observations provide an important basis for the participant discussion, listen closely to the "points of pain" expressed during the discussion and use these as additional input to the findings.

Findings Generation

During planning, the project manager can choose to have the assessors generate the findings off-line or to have the project team itself draft the findings with assessor facilitation. In the first case, the assessors develop finding statements after the discussion (or, if no discussion was held, after questionnaire response analysis), using all available data gathered during the assessment. Alternatively, the participant discussion is extended by at least one hour, during which the assessors help the participants craft finding statements from the notes gathered.

In either case, the primary deliverables are up to three finding statements per explored KPA. Each finding identifies a relative process weakness, states actual or typical consequences of the weakness, and recommends ways to address the weakness. Many projects face similar problems. To accelerate findings generation, we compiled a database of all finding statements from completed

Tip #8: Keep the number of findings small to keep the project team from being overwhelmed and to provide just a few improvement areas on which to focus.

Tip #9: Perform a reality check on each recommendation you offer by evaluating whether it really would help address the corresponding finding, and whether the project team could actually implement it if they decided to do so.

mini-assessments. This database also provides a valuable summary of the issues and concerns project teams are facing, and we have used it for strategic planning.

Findings Presentation

The last step of a mini-assessment is to present the findings summary to the appropriate audience. If the assessors generated the findings, they will present them to the project team. If the team generated the findings, they may choose to present the final findings slides to their own management. The scope of visibility of the findings is entirely up to the project team.

During this presentation, we conclude the mini-assessment and again emphasize the next steps that the project team needs to take. The project manager should announce the intentions for action planning sessions. Projects that complete a mini-assessment but take no further action, and projects that are unable to translate action plans into actions, create an impression that management isn't serious about process improvement.

Supporting Infrastructure

We developed an extensive supporting infrastructure to make the MMA repeatable, reliable, and efficient. These elements are not available publicly, but I recommend you develop similar components for your own use. The work aids could include procedural guidance, presentation slide modules, forms, checklists, and a database of information from the assessments you conduct.

Procedural Guidance

The MMA procedural guidance document contains detailed procedures for planning and conducting a mini-assessment, entry and exit criteria and checklists for each step, and the roles and activities for the lead and backup assessors. Various tables guide the selection of appropriate component options to satisfy each project's mini-assessment objectives. We update this document as we gain experience and find ways to improve the method.

Slide Modules

We developed more than a dozen Microsoft PowerPoint slide modules that assessors can use during the presentation events. These include three CMM training modules of different durations, slides describing the MMA process, a one-slide confidentiality statement, slides illustrating how questionnaire responses are analyzed, and templates for preparing observations and findings. The ability to quickly assemble the slides to be used for each MMA activity from standard packets saves considerable time and reinvention for each mini-assessment. It also increases the repeatability of the MMA process. Assessors can quickly tailor the slide modules for a specific project's assessment.

Forms, Checklists, and Templates

We also developed several tools to streamline mini-assessment execution. To collect background information prior to the planning meeting, we give the project manager a standard project profile questionnaire and a mini-assessment readiness survey. We created forms to plan a mini-assessment, record the time each assessor spends on each mini-assessment stage, collect summary data, and obtain post-assessment feedback from project members. An overall process checklist helps make sure the assessors don't overlook any tasks, and that nothing is forgotten on the way to a meeting. Our templates also help us write mini-assessment agreements and summary reports. As with the slide modules, these electronic aids save us time and enable a repeatable MMA process.

Tip #10: Avoid over-customizing your generic materials for specific events. You can lose the cost savings from reuse if you spend too much time on tailoring.

Mini-Assessment Metrics Database

One large Kodak department created a database to store information about their mini-assessments. The data stored includes:

- The date, duration, and number of participants in each MMA meeting
- The questionnaire used, KPAs covered, and questionnaire results
- The KPAs that were selected for process improvement activities
- The assessor time spent on each phase of the mini-assessment

The accumulated data regarding the average time spent on each step helps us plan our schedules and commitments more reliably, as well as letting us calculate the cost of each mini-assessment in total staff time. The data in this database also lets us track an organization's SPI progress, by aggregating results from multiple projects and seeing which KPAs are being pursued by various projects.

Experience Report

For our first 24 mini-assessments, the average project team size was 12, with a range of six to 20 participants. The flexibility of the MMA method results in a wide range of both assessor and project participant effort, depending on the component options selected. On average, the participants spent four hours in project activities and the assessors expended a total of 48 labor-hours of effort. The assessor effort decreased as all our assessors became fully trained and experienced. The cost of a typical mini-assessment is approximately $6,000, 5-10% of the cost of a formal CMM appraisal.

Project managers chose many different combinations of assessment components to create custom approaches that best suit their needs and schedules. This demonstrates the value of the modular approach. Nearly all the available component options have been selected by at least one project. Project managers appreciate the respect this flexible method shows for their realities and pressures.

Project team members also feel generally positively toward the mini-assessment experience: 79% of those who returned feedback forms (approximately 30% response rate) indicated that the amount of project time spent on the MMA activities was about right. Moreover, 52% felt the mini-assessment would be very or somewhat helpful to their SPI efforts, 37% said it was too soon to tell, and only 11% indicated that the mini-assessment would be detrimental or not helpful.

The strategy of using mini-assessments to launch and sustain SPI activities yields multiple benefits. A mini-assessment can be timed appropriately for each project team, whereas a full organizational assessment can be disruptive to projects with looming deadlines. Each project can deal specifically with its own process issues. The whole project team usually is involved, rather than just a subset of a large organization as in a typical CBA IPI. This helps engage all project team members and leaders in the process improvement activities.

By working closely with project teams through mini-assessments, SEPG members acquire a better understanding of the common challenges the projects face. One large organization used the patterns of observed process shortcomings to direct the evolution of its SPI strategy over time. The SEPG members gain credibility with the software developers through face-to-face—but non-confrontational—interactions during a mini-assessment. The SEPG has opportunities to leverage improved processes from one project to another. This minimizes the amount of invention each project must do on its path to improved processes.

This approach to process assessment is subject to the same risks that confront any SPI activity (see Chapter 4). The most common point of failure we have observed is a lack of follow-through into action planning and action plan implementation following the mini-assessment. The timing of the mini-assessment is also important. If performed too late in the project life cycle, it can be difficult for the project to adjust its plans and schedule to permit time for action planning. Also, later mini-assessments might provide fewer benefits for the current project, although they position the team for greater success on their next release or project.

The MMA method does have some shortcomings. The absence of focused interviews and document reviews reduces the rigor of the mini-assessment, making it a less reliable predictor of the likely outcome of a full CMM-based appraisal. In addition, senior managers are not as directly engaged in mini-assessments as they would be in a full organizational assessment. Nonetheless, the modular mini-assessment method became an effective component of a large-scale software process improvement initiative at Kodak.

Practice Activities

1. If you're using the CMM or CMMI to guide your SPI activities, which of the options in Table 7-1 could you use in a mini-assessment approach that would be effective for your projects?
2. If you aren't using the CMM or CMMI, can you modify the options in Table 7-1 to come up with a mini-assessment approach that would work for your purposes?

Appendix: Process Improvement One-Liners

The experienced software process improvement leader accumulates a list of short, pithy observations about this challenging domain. Here are some that I've picked up over the years (Wiegers 1996a, Wiegers 2003).

- **Process improvement is a journey, not a destination.** The destination is improved business results. Better software engineering and management processes are a means to that end.
- **Take chewable bites.** If you bite into too large a process change, the team will likely choke on it.
- **Take a lot of satisfaction from small victories.** You won't have many big victories. Enjoy the small ones, and spread the word about the successes to help motivate other teams and persuade the skeptics that there is indeed a payoff.
- **Gentle pressure, relentlessly applied.** The process improvement leaders and committed managers steer the team toward a better future by keeping the change initiative visible and continually chipping away at it.
- **Focus, focus, focus.** A busy software team can only work on three, or two, or perhaps just one improvement initiative at a time. But never work on fewer than one.
- **Look for allies.** Every team has its early adopters who will try out new templates and procedures and give the improvement leaders feedback. Cultivate them. Thank them. Reward them.
- **Action plans that don't turn into actions are not useful.** It's easy to do a process assessment and to write an action plan. It's hard to get people to work in new ways that hold the promise of better results, yet that's the only useful outcome of process improvement.
- **Process improvement should be evolutionary, continuous, and cyclical.** Introduce change incrementally and sustain the altered behavior so it becomes routine practice on subsequent projects.
- **Process changes should be goal-oriented, data-driven, and nondogmatic.** Change for its own sake doesn't attract many disciples. Think about how you're going to measure the results of the process changes. Focus on curing the pain, not on following a script for a particular process model.
- **Strive for perfection, settle for excellence.** You'll never get perfect processes, products, or projects. Rather than trying to perfect your new procedures on the first try, plan to revise them based on field experience instead of theory.

References

Abran, Alain, and James W. Moore, eds. 2004. *Guide to the Software Engineering Body of Knowledge, 2004 Version*. Los Alamitos, CA: IEEE Computer Society Press.

Beck, Kent. 2002. *Test Driven Development: By Example*. Boston, Mass.: Addison-Wesley.

Booch, Grady, James Rumbaugh, and Ivar Jacobson. 1999. *The Unified Modeling Language User Guide*. Reading, Mass.: Addison-Wesley.

Brown, Norm. 1996. "Industrial-Strength Management Strategies." *IEEE Software*, 13(4): 94-103.

Brown, Norm. 1999. "High-Leverage Best Practices: What Hot Companies Are Doing to Stay Ahead." *Cutter IT Journal*, 12(9): 4-9.

Caputo, Kim. 1998. *CMM Implementation Guide: Choreographing Software Process Improvement*. Boston, Mass.: Addison-Wesley.

Chrissis, Mary Beth, Mike Konrad, and Sandy Shrum. 2003. *CMMI: Guidelines for Process Integration and Product Improvement*. Boston, Mass.: Addison-Wesley.

Daskalantonakis, Michael. 1994. "Achieving Higher SEI Levels." *IEEE Software,* 11(4): 17–24.

DeMarco, Tom, and Timothy Lister. *Peopleware: Productive Projects and Teams, 2nd Edition.* New York: Dorset House Publishing, 1999.

Diaz, Michael, and Joseph Sligo. 1997. "How Software Process Improvement Helped." *IEEE Software*, 14(5): 75-81.

Dunaway, Donna K., and Steve Masters. 1996. "CMM-Based Appraisal for Internal Process Improvement (CBA IPI): Method Description." Technical Report CMU/SEI-96-TR-007, Carnegie Mellon University/Software Engineering Institute, Pittsburgh, Pa.

Fowler, Priscilla, and Stan Rifkin. 1990. "Software Engineering Process Group Guide." Technical Report CMU/SEI-90-TR-024, Carnegie Mellon University/Software Engineering Institute, Pittsburgh, Pa.

Haley, Thomas J. 1996. "Software Process Improvement at Raytheon." *IEEE Software*, 13(6): 33-41.

Humphrey, Watts S. 1995. *A Discipline for Software Engineering*. Boston, Mass.: Addison-Wesley.

Humphrey, Watts S. 1996. *Introduction to the Personal Software Process*. Boston, Mass.: Addison-Wesley.

Jones, Capers. 1994. *Assessment and Control of Software Risks.* Englewood Cliffs, N.J.: PTR Prentice-Hall.

Kerth, Norman L. 2001.*Project Retrospectives: A Handbook for Team Reviews*. New York: Dorset House Publishing.

Masters, Steve, and Carol Bothwell. 1995. "CMM Appraisal Framework, version 1.0." Technical Report CMU/SEI-95-TR-001, Carnegie Mellon University/Software Engineering Institute, Pittsburgh, Pa.

McConnell, Steve. 1996. *Rapid Development: Taming Wild Software Schedules*. Redmond, Wash.: Microsoft Press.

Paulk, Mark, et al. 1995. *The Capability Maturity Model: Guidelines for Improving the Software Process.* Reading, Mass.: Addison-Wesley.

Potter, Neil S., and Mary E. Sakry. 2002. *Making Process Improvement Work: A Concise Action Guide for Software Managers and Practitioners*. Boston, Mass.: Addison-Wesley.

Wakulczyk, Marek. 1995. "NSSF Spot Check: A Metric Toward CMM Level 2." *CrossTalk*, 8(7): 23–24.

Whitney, Rose, et al. 1994. "Interim Profile: Development and Trial of a Method to Rapidly Measure Software Engineering Maturity Status." Technical Report CMU/SEI-94-TR-4, Carnegie Mellon University/Software Engineering Institute, Pittsburgh, Pa.

Wiegers, Karl E. 1996a. *Creating a Software Engineering Culture*. New York: Dorset House Publishing.

Wiegers, Karl. 1996b. "Software Process Improvement: 10 Traps to Avoid." *Software Development*, 4(5): 51-58.

Wiegers, Karl. 1998a. "Molding the CMM to Your Organization." *Software Development,* 6(5): 49-53.

Wiegers, Karl. 1998b. "Improve Your Process With Online 'Good Practices'." *Software Development,* 6(12): 45-50.

Wiegers, Karl. 1999a. "Why is Process Improvement So Hard?" *Software Development,* Web Edition (February).

Wiegers, Karl. 1999b "Process Improvement that Works." *Software Developmen*t, 7(10): 24-30.

Wiegers, Karl. 2000. "Personal Process Improvement." *Software Development*, 8(5): 65-69.

Wiegers, Karl E. 2002. *Peer Reviews in Software: A Practical Guide*. Boston, Mass.: Addison-Wesley.

Wiegers, Karl E. 2003. *Software Requirements, 2nd Edition*. Redmond, Wash.: Microsoft Press.

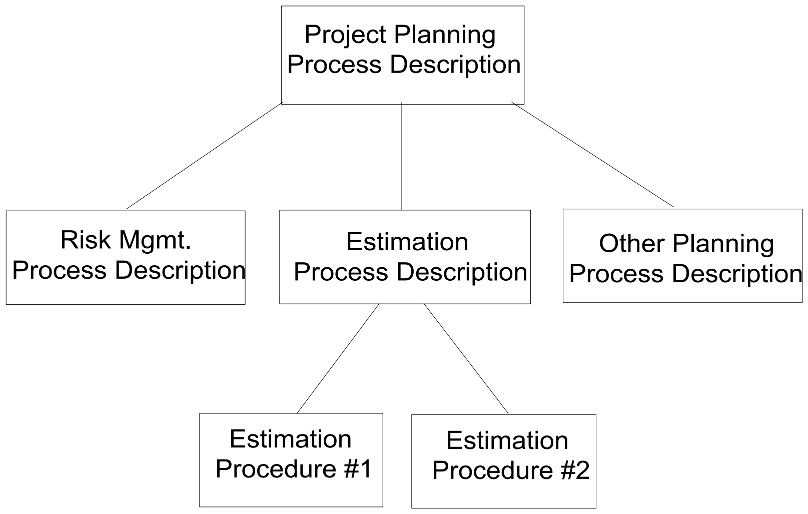
Wiegers, Karl and Doris Sturzenberger. 2000. "A Modular Software Process Mini-Assessment Method." *IEEE Software,* 16(1): 62-69.

Zubrow, David, et al. 1994. "Maturity Questionnaire." Technical Report CMU/SEI-94-SR-07, Carnegie Mellon University/Software Engineering Institute, Pittsburgh, Pa.

Glossary

Assessment (or Appraisal)	An independent and objective evaluation of a project's or organization's current software engineering and project management practices to assess their effectiveness, identify findings, and make recommendations for improvement. Assessments are generally conducted to evaluate performance against an established process reference model, such as the CMM.
Capability Maturity Model	A process improvement framework that describes five levels of increasing process capability. Developed by the Software Engineering Institute. There are several CMMs; "CMM" alone refers to the CMM for Software.
Capability Maturity Model Integration	An integrated, second-generation process improvement framework that combines elements of the original Software and Systems Engineering CMMs. The CMMI comes in both staged (5-level) and continuous representations.
Checklist	A list that enumerates activities, deliverables, or other items to be noted or verified.
CMM	*see* Capability Maturity Model
CMMI	*see* Capability Maturity Model Integration
Finding	An observation from a process assessment of a relative strength or weakness of a project's or organization's current engineering or management practices.
Form	A document with blanks for inserting information.
Guideline	A document that presents recommended, but not required, ways to perform some activity.
Key Process Area	"A cluster of related activities that, when performed collectively, achieve a set of goals considered to be important for establishing process capability." (Paulk 1995)
Management Steering Committee	A team of managers at various levels who set objectives and expectations for an organization's software process improvement efforts, provide funding, allocate staff, and monitor progress.
Plan	A document or collection of documents that outlines how an objective will be accomplished.
Policy	A guiding principle that an organization or project adopts to influence and determine decisions. A policy sets an expectation of behaviors and actions that are to be found in the organization.
Procedure	A written description of a course of action to be taken to perform a given activity. A procedure tells *HOW* some activity is to be accomplished. It typically includes step-by-step descriptions of the sequence of tasks to be performed. A good procedure includes entry criteria, task descriptions, verification steps, and exit criteria.
Process Assets	Documents and other items that assist a project team with the effective performance of its processes. Process assets include policies, standards, process descriptions, procedures, guidelines, document templates, forms, spreadsheet tools, and examples of actual project work products.

Process Assets Library (PAL)	A repository that contains an organization's collected process assets and makes these accessible by the members of the organization.
Process Description	A "process" is a sequence of activities performed for a given purpose. A "process description" is a documented definition of those activities. A process description states *WHAT* is being done. A process description might refer to multiple procedures for the step-by-step task details. See the figure below. A good process description completely and precisely specifies the characteristics of the process. It should include entry criteria, tasks descriptions, verification steps, and exit criteria.


Project Planning Process Description
Risk Mgmt. Process Description
Estimation Process Description
Other Planning Process Description
Estimation Procedure #1
Estimation Procedure #2

Process Owner	A manager or senior process engineer who is responsible for leading the improvement activities for a specific process area. Responsibilities include establishing management policy, chartering working groups, updating the process as experience is gained, and monitoring the effective application and benefits of the process.
Software Engineering Process Group (SEPG)	A group of process improvement leaders, software practitioners, and managers chartered by senior management to coordinate and facilitate an organization's software process improvement activities.
Standard	Mandatory requirements employed and enforced to prescribe a uniform approach to performing an activity or creating a product. A standard often describes the required contents of a document. The standard does not in itself dictate that the activity will be performed or the work product will be created; policies do that. However, if an activity is performed or a work product is created and a pertinent standard exists, the standard will dictate how the activity will be performed or the form and content of the work product.
Template	A pattern to be used as a guide for producing a complete document or other item.

About Process Impact

Process Impact is a software process consulting and education company in Portland, Oregon. Karl Wiegers, Ph.D., the Principal Consultant, has many years of experience with software development and management. His interests include project management, requirements engineering, peer reviews, process improvement, and metrics. Karl is the author of the books *Software Requirements, 2nd Edition*, *More About Software Requirements*, *Peer Reviews in Software,* and *Creating a Software Engineering Culture*. Process Impact provides a variety of training and consulting services. Visit http://www.processimpact.com for many magazine articles, templates, spreadsheet tools, other process assets, and tasty recipes (on the Biography page).

Training Seminars

Process Impact's presentations include the following full seminars, in addition to many shorter presentations. Seminars can be customized to meet your needs. You may license these seminars for presentation by your own staff.

"Project Management Best Practices" (1 day)
"Introduction to Software Risk Management" (half-day)
"In Search of Excellent Requirements" (1-3 days)
"Exploring User Requirements with Use Cases" (1 day)
"Writing Quality Requirements Workshop" (1-day)
"Software Inspections and Peer Reviews" (1 day)
"Creating a Software Engineering Culture" (1 day)
"Software Subcontract Management Process and Guidance" (1.5 days)

eLearning Seminars

Several of our most popular seminars are available on CD in a rich, self-paced eLearning format for both individual and site licensing. These courses integrate animated slides with narration by Karl Wiegers in a presentation that plays in your Web browser. They include slide notes, student handouts, quizzes, practice sessions, magazine articles, templates, and other process assets.

"In Search of Excellent Requirements" (10 hours of audio)
"Exploring User Requirements with Use Cases" (6 hours of audio)
"Project Management Best Practices" (7.5 hours of audio)
"Software Inspections and Peer Reviews" (6 hours of audio)
"Process Impact Virtual Conference" (6.5 hours of audio)
"Software Requirements: An Executive Overview" (90 minutes of audio)

Consulting

Process Impact provides both on-site and off-site consulting services to software organizations of all types. Let us help you tune up your software processes, write better requirements specifications, and manage projects more successfully.

Contact Information

Process Impact
11491 SE 119th Drive
Clackamas, OR 97015-8778
503-698-9620 (voice)   503-698-9680 (fax)
sales@processimpact.com
http://www.processimpact.co