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.

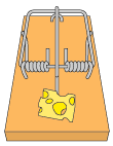
Manufacturing industries realize they need to take their equipment off line periodically for retooling, performing preventive maintenance, and installing upgrades that will improve productivity or quality. The product managers and marketing people accept this planned downtime as a necessity and adjust 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.



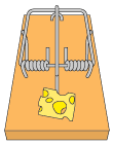
See Trap #3, time-stingy project managers, in Chapter 4.

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® - SE/SW (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-SE/SW, 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.

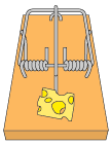
Start with some introspection. What aspects of your current ways of working cause the most late nights at the keyboard? Which situations lead to frustrating rework of completed effort? Are you struggling most with meeting schedules, applying new technologies effectively, developing the right product, 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

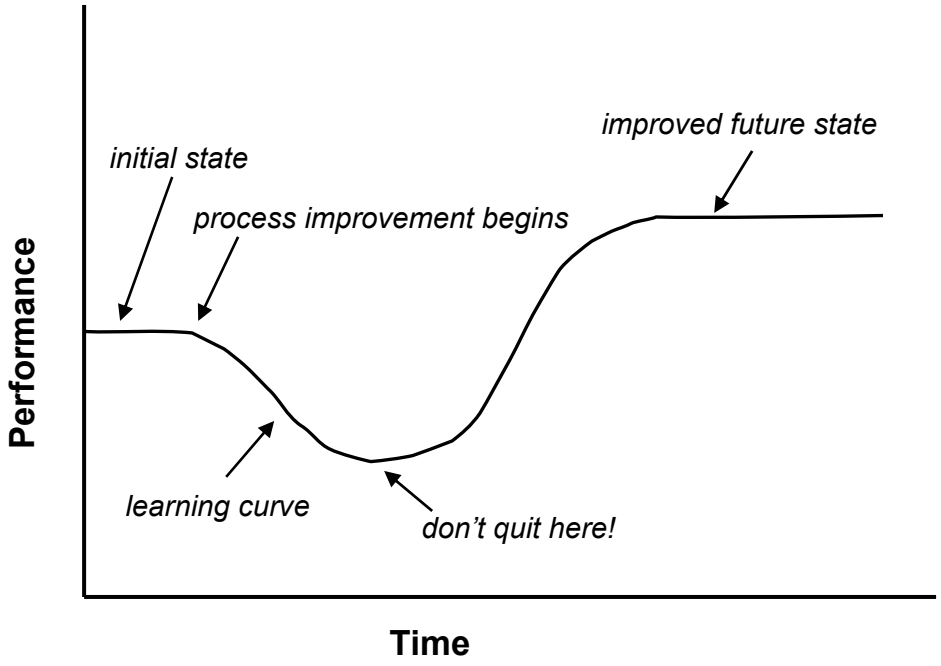


Figure 1-1: The learning curve

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			