Why Is Process Improvement So Hard?[1]

Karl E. Wiegers

Process Impact
716-377-5110
www.processimpact.com

I have never met anyone who could truthfully say, "I am building software today as well as software can ever be built;" I certainly cannot. 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. It seems straightforward, and the literature has a number of success stories of companies that substantially improved their software development and project management capabilities. However, many software organizations that tackle process improvement do not manage to achieve significant and lasting improvements in the way they conduct their projects.

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 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 software process improvement. This article examines five major reasons why it is difficult to make process improvement work and provides some suggestions of how you can avoid these pitfalls.

Not Enough Time

Insane schedules leave insufficient time to do the essential project work, let alone to investigate and implement better ways to work. No software groups are sitting around with plenty of spare time to devote to exploring what's wrong with their current development processes and what they should be doing differently. Customers and senior managers are demanding more software, faster, with higher quality, and incidentally we have to downsize a bit and we don't have the budget to pay for overtime, so sorry. One consequence is that software organizations may deliver release 1.0 on time, but then they have to ship release 1.01 almost immediately thereafter to fix the most egregious bugs.

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 is true far less frequently than it is claimed to be true. I use some well-known commercial software products that may have met their ship dates (I don't know for sure), but I would be embarrassed to admit I was on the development team. I would have preferred the developers spent more energy on improving quality, an outcome of many process improvement efforts.

[1] This paper was originally published in the on-line edition of Software Development, February 1999. It is reprinted (with modifications) with permission from Software Development magazine.

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 fail to appreciate the potential 3- to 10-fold return on investment in quality (according to Capers Jones). 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 I, the customer, waste working around bugs in the software, and my annoyance toward the company that sold it to me.

Manufacturing industries realize their equipment must be taken off line periodically for retooling, 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 can't really shut down the software development machine, so you have to implement the upgrades in parallel with your production work.

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

You have 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 effort to ongoing process improvement: 10 percent if you're aggressive about improvement, 5 percent if you want to make some significant headway, and 2 percent if you want to pretend you're making progress. 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 do not spend the time to improve how you perform technical and management work starting tomorrow, don't expect your next project to go any better than your current one.

Lack of Knowledge

A second obstacle to widespread process improvement is that many software practitioners don't seem to be familiar with industry best practices. I do informal surveys of audiences at conferences and training seminars, asking how many attendees perform one practice or another and how many people are familiar with certain of the topics I'm discussing. This unscientific data suggest that the average software developer doesn't spend much time reading the literature to find out about the best known ways of doing software development. Programmers may buy books on Java and COM, but don't look for anything about process, testing, or quality on their bookshelves.

The industry awareness of process improvement frameworks such as the Capability Maturity Model for Software (CMM) has grown in recent years, but effective and sensible application still is not that common. Too few developers and managers are actively applying the experience embodied in collected bodies of software knowledge such as the IEEE Software Engineering Standards Collection (www.computer.org). Many recognized industry best practices simply are not in widespread use in the software development world. These best practices include formal inspections, risk management, change control, metrics-based project estimation and tracking, and requirements management.

To overcome this barrier to successful software process improvement, read the literature! Learn about industry best practices from sources such as Steve McConnell’s *Rapid Development: Taming Wild Software Schedules* (Microsoft Press, 1996), and an article by Norm Brown titled “Industrial-Strength Management Strategies” (*IEEE Software*, July 1996). Leverage the sharing of local best practices among the people in your group or department, so all have a chance to learn from those individuals who have strengths in particular areas. Facilitate a periodic Lunch and Learn session (lubricated with pepperoni, sugar, and caffeine) in which team members take turns selecting an article or book chapter to read, discuss, and seek to apply to your project. As a manager, reward those team members who go out of their way to learn about and apply practices that can address the group’s technical and management challenges. The information is out there; take the time to find it and use it.

Wrong Motivations

Some organizations launch process improvement initiatives for the wrong reasons. Maybe an external entity, such as a contractor, 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 told me recently that his organization was trying to reach CMM Level 2 (of five), but when I asked why, he had no answer. 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 software process improvement should be to make some of the current pain you experience on your projects go away. Team members are rarely 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. I know of many process improvement initiatives that met with immediate practitioner resistance when couched in terms of “doing the CMM thing,” without a clear explanation of the reasons why improvement was needed and the benefits the team hoped 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 most often to frustrating rework of completed effort? Are you struggling most with meeting timetables, 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 review (postmortem) also provides a structured way to reflect on what has gone well so far (keep doing it!) and what has not (do something different!).

Dogmatic Approaches

A fourth barrier to effective process improvement is the checklist mentality, a rigid and dogmatic implementation of the CMM 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 better collective results. You can have fabulous procedures and processes, but if no one follows them, if they collect dust on the shelf, if people still work as they always have, then you have not succeeded with process improvement.

Process change is culture change, so 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 piece of manufacturing apparatus that runs twice as fast. 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 my article “Molding the CMM to Your Organization,” *Software Development*, May 1998). For example, many development organizations have problems with their requirements engineering and testing practices, but these issues are not addressed at all by the key process areas at CMM Level 2. If it hurts, fix it, no matter what the checklist says.

New processes must be flexible and realistic. I know of one company that mandated that all software work products would be formally inspected, which is a great idea but not a realistic expectation for most organizations. Their waiver process will be running overtime and practitioners may play games to appear as if they are 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.

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?” 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 long-term benefits to Persons B and C in the downstream development and maintenance activities, even if it costs Person A more time today.

Insufficient Commitment

A final reason that software process improvement fails is that organizations claim the best of intentions but lack a true commitment to process improvement. They start with 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 pilot no new processes.

It is easy to achieve a return on your software process improvement investment of zero. You do this by investing in a process assessment, training the team, writing process improvement action plans, and developing new procedures, but never actually changing the way people in the organization 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.

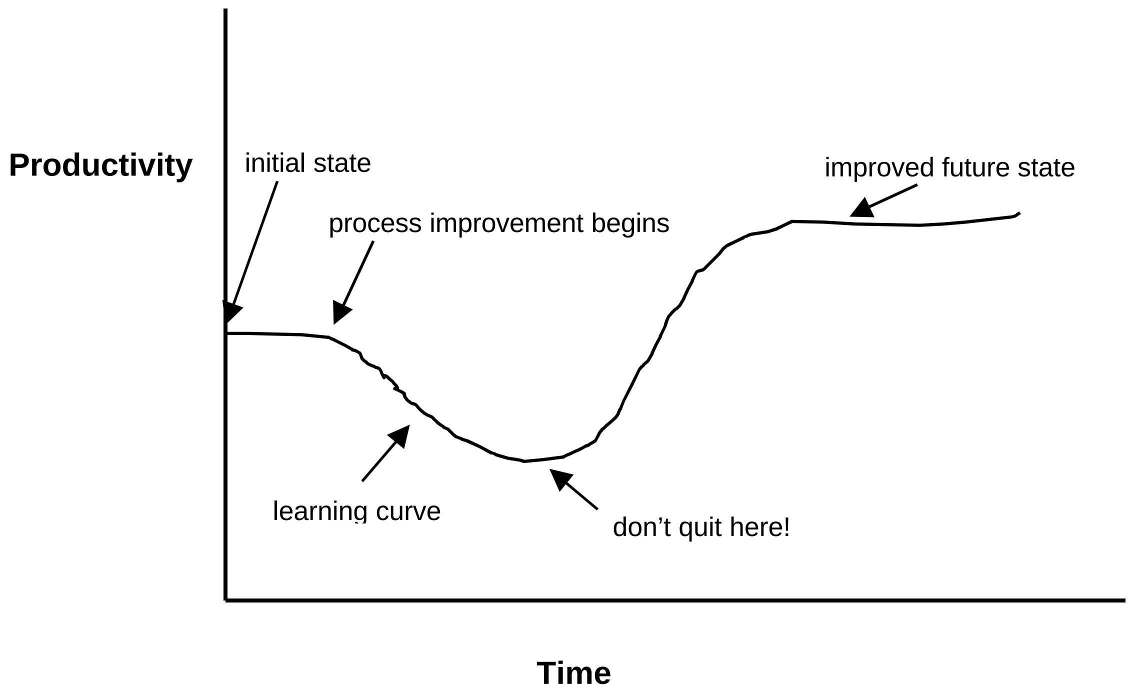
Sometimes, initial enthusiasm and commitment wane when quick results are not immediately forthcoming. Some improved practices, such as the use of static code analysis tools, can yield better results immediately. Other practices, such as metrics-based estimation, may take awhile to demonstrate their payoff. Respect the reality of the learning curve, or the J-curve of technology adoption (Figure 1). The investment you make in process improvement will have an impact on your current productivity, because the time you spend developing better ways to work tomorrow is not available for today's assignment. 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 (including Motorola, Hewlett-Packard, Raytheon, Boeing, and others that have not published their experiences) have enjoyed from sustained software process improvement initiatives.

What Should You Do?

Here are some tips for avoiding these process improvement pitfalls:

- Recognize that SPI is 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.
- 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 any of the established approaches and use it as a guide, not a

Figure 1. The process improvement learning curve.



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. Allocate resources, consider risks, build schedules, track progress against the plans, measure results, and celebrate your successes.

You can make substantive improvements in the way you build software; you have to.