The Power of Process

Steve McConnell, Construx Software Builders

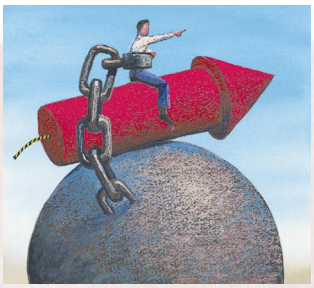
Some people in the software development community think "process" is a four-letter word. They think software processes are rigid, restrictive, and inefficient. They hold that the best way to run a project is to hire the best people you can, give them all the resources they ask for, and turn them loose to do what they do best.

Sure, they say, there will be some amount of unproductive work (also known as "thrashing"). After all, developers will make mistakes. But they will also be able to quickly and efficiently correct these mistakes at a cost that is less overall than the cost of processes.

People who hold this view imagine a work breakdown over the course of a project like the one shown in Figure 1: Projects that run without any attention to process can run extremely efficiently. Adding processes, they argue, is pure overhead that simply takes time away from productive work. In summary, their view of the cost of process is illustrated in Figure 2.

This point of view has intuitive appeal. At the beginning of a project (dark areas in Figure 2), a focus on process certainly does take time away from productive work. If that trend were to continue throughout the project (the light shaded area), it wouldn't make sense to spend much time on process.

Send submissions to Management column, *Computer*, PO Box 3014, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720; fax (714) 821-4010; computer@computer.org.



To get managers to pay attention to process, talk ROI.

COSTS OF INATTENTION TO PROCESS

Software industry experience, however, has found that for medium and large projects the trend shown in Figure 2 does not continue throughout the project. In fact, the opposite is true: Projects that don't pay attention to establishing effective processes early are forced to slap them together late, when slapping them together takes more time and does less good.

Here are some examples of how inattention to process can cost you:

Scope creep. In the middle of the project, team members agree informally to implement a wide variety of changes that are proposed to them directly by their manager or customer. They don't begin controlling changes systematically until late in the project. By that time, the scope of the product has expanded by 25 to 50 percent or more, and the budget and schedule have expanded accordingly.

Daily meetings. Projects that don't set up processes to eliminate defects in early stages fall into seemingly interminable test-debug-reimplement-retest cycles. So many defects are discovered that, by the end of the project, the change control board (or feature team) may meet as often as every day to prioritize defect corrections.

No planning or control. Major defects discovered late in the project cause the software to be redesigned and rewritten during testing. Since no one planned to rewrite the software during testing, the project deviates so far from its plans that it essentially runs without any planning or control.

Releasing known defects. Defect tracking isn't set up until late in the project. Some reported defects go unfixed simply because they are forgotten, and the product is released with known defects that could have been fixed easily.

Integration problems. Components developed by different developers are not integrated until the end of the project. By the time the components are integrated, the interfaces between components have gotten out of synch and much work must be done to bring them back into alignment.

Overwriting sources. Source code revision control isn't established until late in the project, after developers have begun to lose work by accidentally overwriting the master copies of source code files.

Constant re-estimation. Because a project is having schedule trouble, developers are asked to re-estimate remaining work as often as once a week or more, taking time away from their development work.

WHEN A PROJECT THRASHES

When a project has paid too little early attention to process, by the end of a project developers feel they are spending all of their time in meetings or correcting defects, with little or no time left to extend the software.

When developers do not meet deadlines, their survival impulses kick in. They retreat to solo mode, focusing exclusively on their personal deadlines. They withdraw from interactions with managers, customers, testers, technical writers, and the rest of the development team. Project coordination unravels.

Far from the steady level of productive work suggested by Figure 1, my observation is that the medium or large project

conducted without much attention to development processes typically experiences the pattern shown in Figure 3.

In this pattern, projects experience a steady increase in thrashing over the life of the project. By the middle of the project, the team realizes that it is spending a lot of time thrashing and that some process would be beneficial. But by then much of the damage has been done. The project team tries to increase the effectiveness of its process, but its efforts can only hold the level of thrashing steady, at best. In some cases, the late attempt to improve the project's processes actually makes the thrashing worse.

In this scenario, lucky project teams release their products while they are still eking out a small amount of productive work. Unlucky teams can't complete their products before reaching a point at which 100 percent of their time is spent on process and thrashing. If you think that attention to process is needless overhead, consider that a canceled project has an overhead of 100 percent.

PROCESS TO THE RESCUE

Fortunately, attention to process provides an alternative to this dismal scenario. When effective processes are used, the project profile looks like the one shown in Figure 4.

During the first few weeks of the project, a process-oriented team will seem less productive than a process-phobic team: The level of thrashing is low on both projects, and the process-oriented team will be spending a significant amount of its time on processes. By the middle of the project, the team that focused on process early will have reduced the level of thrashing compared to the beginning of the project, and will have streamlined its processes. At that point, the process-phobic team will be just beginning to realize that thrashing is a significant problem and just beginning to institute some processes of its own.

By the end of the project, the process-oriented team will be operating at a high-speed hum, with little thrashing, and performing its processes with little conscious effort. It will tolerate a small amount of thrashing because eliminating the last bit of thrashing would cost more in overhead than would be saved. When all is said and done, the overall effort on the project will be considerably lower than the effort of the process-phobic team.

Organizations that have explicitly focused on improving their development processes have, over several years, cut their time to market by about one-half and have reduced their costs and defects by factors of three to 10. See the "Sources of Process Success Stories" sidebar for a summary of some published findings.

Here's the best news. The average cost of these improvements was only about 2 percent of total development costs—typically about $1,500 per developer per year (J. Herbsleb et al., *Benefits of CMM Based Software Process Improvement: Initial Results*, Tech. Report CMU/SEI-94-TR-13, Software Engineering Institute, 1994).

PROCESS VERSUS CREATIVITY

One of the common objections to putting systematic processes in place is that they will limit programmers' creativity. Programmers do indeed have a need to be creative. Managers and project sponsors also have a need for projects to be predictable, to provide progress visibility, and to meet schedule, budget, and other targets.

It is certainly possible to create an oppressive environment in which programmer creativity and management goals are placed at odds, and many companies have done that, but it is just as possible to set up an environment in which those goals are in harmony and can be achieved simultaneously.

Companies that have focused on process have found that effective processes *support* creativity and morale. In a survey of about 50 companies, only 20 percent of the people in the least process-oriented companies rated their staff morale as "good" or "excellent" (J. Herbsleb et al., "Software Quality and the Capability Maturity Model," *Comm. ACM*, June 1997, pp. 30-40). The responses were consistent across managers, developers responsible for process improvement, and senior technical staff. In organizations that paid more attention to their software processes, the proportion of people who rated their staff morale as "good" or "excellent" jumped to 50 percent. And, in the most process-sophisticated organizations, 60 percent

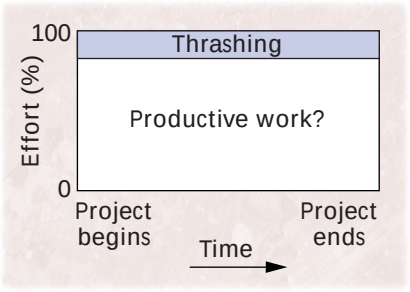


Figure 1. People who don't believe in process envision a work breakdown like this: Developers are essentially productive throughout the project with only a small amount of thrashing.

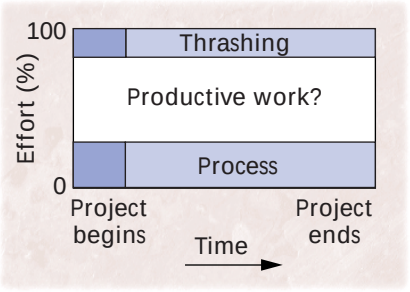


Figure 2. To the people who believe that projects run as illustrated in Figure 1, adding process hinders productivity and adds nothing.

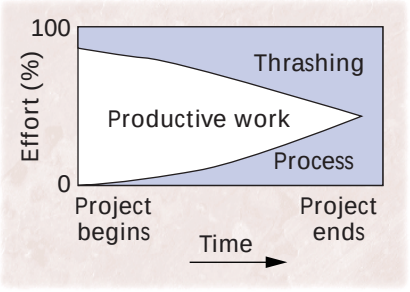


Figure 3. In my opinion, ignoring process actually produces a breakdown like this: Not only do developers thrash more as the project proceeds, eventually the team will have to institute many processes anyway.

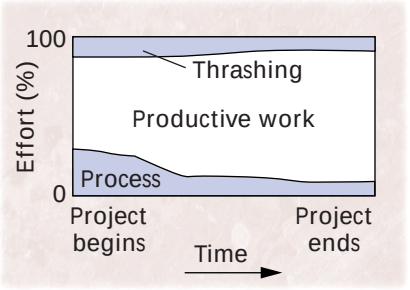


Figure 4. Instead of the scenario illustrated in Figure 3, early attention to process increases productivity as the project proceeds.

of the people rated their morale as "good" or "excellent."

Programmers feel best when they're productive. Programmers dislike weak leadership that provides too little structure to prevent them from working at cross purposes and, in the end, causes them to spend more time fixing defects than creating new software. Good project leadership puts a focus on process that allows programmers to feel incredibly productive. Developers, their project, and their organization all reap the benefits. ❖

Steve McConnell is chief software engineer at Construx Software Builders and editor of IEEE Software's *Best Practices column. This article was adapted from his book,* Software Project Survival Guide *(Microsoft Press, 1998). Contact him at stevemcc@construx.com.*

Sources of Process Success Stories

To learn more about how process can contribute to your development organization, consult these published accounts:

- Over five years, Lockheed cut its development costs by 75 percent, its time to market by 40 percent, and its defects by 90 percent: A.M. Pietrasanta, "A Strategy for Software Process Improvement," *Ninth Ann. Pacific Northwest Software Quality Conf.*, 1991.
- Over six and a half years, Raytheon tripled its productivity and realized an ROI of almost 8 to 1: T. Haley et al., *Raytheon Electronic Systems Experience in Software Process Improvement*, Tech. Report CMU/SEI-95-TR-017, SEI, 1995.
- Bull HN realized an ROI of 4 to 1 in four years, and Schlumberger an ROI of almost 9 to 1 after three and a half years: J. Herbsleb et al., *Benefits of CMM Based Software Process Improvement: Initial Results*, Tech. Report, CMU/SEI-94-TR-13, SEI, 1994.
- NASA's Software Engineering Laboratory cut its average cost per mission by 50 percent and its defect rate by 75 percent over eight years, while dramatically increasing the complexity of software used: V. Basili et al., "SEL's Software Process Improvement Program," *IEEE Software*, Nov. 1995, pp. 83-87.
- Similar results have been reported at Hughes, Loral, Motorola, Xerox, and other companies that have focused on systematically improving their software processes: H. Saiedian and S. Hamilton, "Case Studies of Hughes and Raytheon's CMM Efforts," *Computer*, Jan. 1995, pp. 20-21; W. Myers, "Good Software Practices Pay Off—Or Do They?" *IEEE Software*, Mar. 1992, pp. 96-97; J. Herbsleb et al., "Software Process Improvement: State of the Payoff," *American Programmer*, Sept. 1994, pp. 2-12.