Reducing Software Project Productivity Risk

by Richard Bechtold
Abridge Technology

Software project results continue to be highly subject to the skills of the individuals involved in the development life cycle. Although improved processes and tools can help increase overall productivity, it remains true that the project team is one of the greatest variables that impact project productivity. To significantly reduce software project productivity risk, it is important to simultaneously evaluate the processes, tools, and skills that characterize the project and its personnel. This article presents three approaches to reducing software project productivity risk by minimizing process rules, by maximizing simple tools, and by de-emphasizing the importance of technical skills in favor of basic abilities.

Rules, Tools, and People

Software-intensive systems development still persists as an extremely people-intensive effort that is highly subject to productivity variances. Software programmer productivity differences have been reported as high as 100-to-1 [1], and more recently as 22-to-1 [2]. Even worse, on a software project with 10 people, you can expect to have up to three developers who are "net negative producing programmers" [3]. These are people whose high rate of defect insertion more than negates their rate of code production. In effect, overall productivity on the project accelerates when such people do not show up for work. Clearly, enhancing programming productivity is essential for ensuring software project success. However, since an average of 84 percent of all software projects fail [4], it is also clear that we are not adequately addressing the problem of reducing software project productivity risk.

Three key components to reducing software project productivity risk are rules, tools, and people. Initially, projects often strive to make individuals more productive. This can be a mistake. Ultimately, what you need to achieve is to make the project, as a whole, more productive. This is accomplished first by thinking of the project personnel as a cohesive team, and second by providing this team with productivity-enhancing tools and rules.

The notion of productivity-enhancing tools is self-evident. However, what are productivity-enhancing rules? Simply stated, these rules are the processes in use on your project. Their purpose is to help ensure the project achieves operational or business objectives. Ironically, in striving to improve the processes, a misunderstanding of the differences between rules and tools can easily lead to bloated processes that reduce overall project productivity. If this is accompanied by the far too common tendency to staff a software project with the wrong types of people, then low productivity and high project failure rates are hardly a surprise.

This paper discusses how to improve overall software project productivity by avoiding bloated process descriptions, by leveraging simple tools, and by recognizing the team-member skills that best ensure the overall success of your project.

Leveraging Rules

Most process quality frameworks, such as the Software Capability Maturity Model (CMM®) [5] and the ISO 9000 set of standards [6], require the use of defined processes. The premise is that well-defined and institutionalized processes are a key component in ensuring the performance of efficient, effective, and repeatable activities. These high-quality activities, in turn, help ensure the production of high-quality products.

However, numerous projects and organizations have suffered through the agony of attempting to implement processes that were voluminous and hard to interpret. In particular, when a project manager is handed a 100-plus page process description, how does he or she distinguish between actual process requirements and the associated guidance, suggestions, or supporting material? Where is the project manager allowed to exercise discretion and judgement? As organizations encounter these problems, and strive to answer these questions, they often resort to attempting to develop process-tailoring guidelines. Occasionally, this effort can result in a set of process descriptions that are far more complicated, and correspondingly more difficult to use.

For example, if you have a notion of four types of projects (e.g., database-intensive, object-intensive, web-intensive, hardware-intensive) and four sizes of projects (e.g., small, medium, large, very large) you now have up to 16 different process variations to define and maintain. Add another dimension, such as customer type (e.g., Department of Defense, other government, industry, mass market), and you may be looking at up to 64 process variations.

Potentially the greatest factor contributing to these problems, and to the corresponding reduction in project productivity, is the difficulty process users have in discerning between those parts of the process descriptions that must be followed (the rules) and those that simply help with understanding and following the rules (the tools). A simple solution to this problem is to stop thinking about process in the abstract, and instead distill it to the component parts of either rules or tools. This will contribute to far smaller process descriptions that document only the essential elements, or rules, that must be followed. All other material is relocated to the tool side of the process.

As with standard process descriptions, there will typically be several layers within a process rule description. For example, at the highest level there may be a rule similar to the following:

- Projects will manage software configurations.

Subordinate to that rule, you might have the following five rules:

- Projects will plan configuration management activities.
- Configuration items will be uniquely identified.

- Changes to configuration items will be controlled.
- Configurations will be audited.
- Configuration status will be reported.

Each of these rules may have its own set of subordinate rules, and so on. Taking this approach can dramatically reduce the size of a process description by reducing it to its essential requirements. This results in a process representation that clearly communicates what has to occur on the project.

This does not eliminate the need for tailoring the rules to specific project characteristics and circumstances. However, there is far less material to tailor, and consequently the tailoring is far easier. For example, tailoring might simply consist of waiver criteria that define when a project is exempt from a particular rule. Implicitly, being waived from a higher-level rule results in a waiver from all its subordinate rules.

Additionally, the principle of moving all extraneous material out of the process descriptions provides the opportunity to apply a litmus test that anyone can use when developing, reviewing, evaluating, implementing, or discussing the contents of process descriptions. The test is to ask whether or not the item under examination is absolutely a requirement for the project. If not, then it does not belong with the process rules. It belongs with the tools.

Leveraging Tools

When we think of software project tools that enhance overall project productivity, we usually think in terms of automated tools that assist with project planning, project tracking, configuration management, action-item tracking, requirements management, systems design, regression test support, quality tracking, etc. However, as implied by the preceding discussion on process rules, anything that is not a rule but helps us interpret or comply with the rules can be considered a process tool. A few common, simple examples of such nonautomated tools include:

- Checklists
- Guidelines
- Templates
- Outlines
- Examples
- Training material

By putting as much process-related information and content into the above and other types of simple process tools, you reallocate your process information more closely to the ratio project personnel generally prefer. Most people prefer as few rules as possible. Conversely, they also prefer having a large assortment of tools available such as templates, examples, and checklists.

Another advantage to minimizing rules, and maximizing these types of simple tools, is the flexibility you have to regularly and rapidly revise, upgrade, augment, or otherwise improve the information, material, and workflow sequences associated with the set of tools that supports the project. Typically, if you want to change the process rules, your proposed changes will require a sequence of reviews, revisions, and approvals before they are authorized for use on the project. Conversely, in most organizations authority to change the miscellaneous material used to support the project is delegated to lower levels, and might even be within the authority of individual projects.

In addition to the simple, nonautomated tools presented above, you can certainly increase productivity through the careful use of automated tools. However, be cautious that the tools you deploy on the project result in an increase in the ability of the project personnel to perform their work. For example, you will usually want to avoid any automated tools that have long, steep, learning curves. Likewise, avoid tools that encourage excessively complicated approaches or solutions, since it is highly likely you will end up with precisely that. Conversely, tools that are easy to learn and use, and that encourage simple solutions to complex problems, can help significantly improve overall project productivity.

It is very important to clearly communicate to all project personnel that the overall objective of tool usage is primarily to improve project productivity. When a tool seems to interfere with productivity, its value must be carefully re-evaluated. Certain tools, such as a checklist that helps developers determine if their code is compliant with software coding standards, may initially seem to reduce productivity. It will take additional time for programmers to complete the checklist as they review their code for compliance to the standard. However, this is a good example of how something that might not seem productive for an individual in the short-term can be productive for the project over the long term. In this case, the checklist can result in higher levels of standards compliance, which can lead to increased readability, maintainability, and reusability of the software components. Each of these benefits can directly and significantly contribute to an overall increase in project productivity.

Although rules and tools are vitally important to software project success, their importance is dwarfed by the need to have the right type of team, consisting of people with the right types of skills. No combination of tools, nor cleverly crafted rules, can save a project that is chronically staffed with the wrong people.

Leveraging People

As described in the introduction, two software developers with similar education, skills, and salary can have a productivity difference of 22-to-1, or even greater. Put differently, Mary can get the job done in two weeks, but John will take nearly a year to accomplish the same result. Rules and tools cannot eliminate this extreme productivity variance. Further reduction of the risks associated with software project productivity clearly requires identifying and staffing the project with a highly productive team. But what traits allow a software team to be productive while pushing toward a common goal? Most organizations talk about finding the best people. But what do we look for when looking for the best?

Although it can be difficult to detail who the best people are, it is very easy to describe who the best people are not. Pick up the Sunday classified ads from your newspaper. Read the job descriptions for the information- and technology-related jobs. There you have it. A perfect prescription for how to identify the worst people. Using the Washington Post Web site [7], most technical, software project related job postings are looking for things like those shown in Figure 1.

Desperate ads look for six months of tool-specific experience. Choosy ads prefer two years. Occasionally, someone wants applicants to have a college degree. It is rare that an ad requires someone to be flexible, multitalented, or a team player. This is in spite of the fact that these three skills can be crucial to the success of a software project.

The following list provides examples of three levels of skills that can apply to any given software development project. The various skill areas are divided into the following levels: essential, important, and useful.[1]

Essential

- Able to learn quickly
- Willing to learn quickly
- Team player
- Generalist, or multiarea specialist
- Strong on fundamentals
- Very flexible
- Able to teach efficiently/effectively

Important

- Insightful
- Adept with abstractions
- Organized
- Disciplined
- Self-motivated
- Good negotiator
- Good problem-solver

Useful

- Strong understanding of the solution engineering techniques
- Strong understanding of the solution engineering tools
- Moderate familiarity with the underlying environment
- Moderate understanding of the problem domain (including under standing the customer/market)
- Familiarity with primary applicable standards
- Good estimation skills
- Good reporting skills
- Good leadership skills

Figure 1. *Breakdown of IT-related job postings*

Keyword	# of Listings
C++	3946
Java	1539
Access	2193
Oracle	2083
Visual Basic	892

The relative importance of various skills from the preceding list runs almost completely at odds with the typical approach most people use for staffing a software project. Typically, the selection criteria for staffing a project is based upon someone having a few years of experience in a particular tool. The above approach demotes tool-experience to the least important skill category. Instead, team skills, fundamental skills, interpersonal skills, and the ability to learn and change rapidly are given much higher value.

The rationale for reducing the importance of tool-specific technical skills and increasing the importance of fundamental abilities and character traits is that if software projects are characterized by anything, most are characterized by rapid changes in requirements, technologies, competing products, and opportunities. Given this environment, software development projects need to be staffed by people who are extremely agile and quick to learn.

Another key factor for determining how to assign relative importance to various skills is the general ease with which the lack of that skill can be fixed by the software project manager. For example, consider the useful skills. Typically, if a new developer does not know your customer, that is fairly easy to fix. Likewise, if the developer does not understand one or more of the tools you use as part of building the product, this, too, can usually be fixed rather easily. This is especially true if, as discussed earlier, the tools you use on the project are easy to learn, understand, apply, and yield simple and elegant solutions.

Similarly, consider the essential skills. It is very hard to teach someone to be a team player. They either naturally are, or naturally are not. Likewise, it is nearly impossible to change someone from being an inflexible person to a flexible person. Additionally, teaching someone everything they need to know to be strong on computer science and software engineering fundamentals is similar to trying to teach them the equivalent of a software technology-intensive bachelor's degree. While not impossible, this type of comprehensive and fundamental education is likely best left to academic institutions.

Skills at using the latest language, tools, and libraries are certainly important, but these sometimes are the most volatile and short-lived skills needed on a project. The rapid and sometimes extreme rate of change in tool capability, requirements, and opportunities may require you to reconsider the tools you are using on your current project, or cause you to prefer other tools for your next project. Having a flexible team with the ability to rapidly learn new tools seems preferable to trying to build a new team.

From a different perspective, the years of experience you seek from a job candidate may not equate to years of applicable experience for your project. For example, you may be better off with someone who has four months of direct experience on your project—even as a trainee—than someone who has a year of experience acquired somewhere else. Even the person with a year's experience might require a month or two to come up to speed in your environment. From this perspective, it can be better to quickly find and hire someone who only lacks your specific tool experience, than to let the position go unfilled even for a month.

In the contracting environment, much of the focus on specific tool experience is the result of standard, if dated, government procurement habits. Most procurements force a bidder to, for example, provide someone with two years of Common Business-Oriented Language experience. These are typically "must comply" requirements and leave the bidder with virtually no choice in identifying and providing the best overall team for a particular contract.

Without question, software development teams need to have at least some people who are adept at the various tools the project will use. No project can afford to be entirely a learning experience. Nevertheless, if most team members exhibit the essential and important skills described in this paper, and at least some team members exhibit the useful skills, then you have an excellent pool of available skills regardless of the circumstances the project eventually encounters.

Summary

It is hoped that this article has illustrated the potential value of minimizing the number of rules your defined process employs, and maximizing the number of

simple tools that support process rules.

But when it comes to further reducing software project productivity risk for a specific project by deliberately seeking personnel with certain skills, assessing the relative value of those skills remains a challenging task. The varying contribution that different skills make toward reducing project productivity risk truly does depend on your project's context, constraints and circumstances.

Nevertheless, here is a test scenario to consider:

You have been tasked to manage a complex software development project that spans several years, has uncertain requirements, and might involve uncertain technologies. A similar project recently finished—two years late—and it was built using power–wizzy–gui–whatever 1.5.

You must select a software development team from the two teams that are available.

One team has considerable experience using power–wizzy–gui–whatever 1.5, but it must comply with a 300-page process description. The team has few routine tools available; it is inflexible, untrainable, disorganized, undisciplined, and cannot function as a team.

The other team clearly understands and applies its 30-page outline of process rules, and it has a comprehensive repository of efficient and effective process tools. However, at the moment the team is weak on the details regarding the use of power–wizzy–gui–whatever 1.5. This team is very quick to learn, reliably operates as a cohesive team, is adept with abstractions, is well-grounded in fundamentals of software architectures and data management, is highly flexible, self-motivated, and contains clever problem solvers.

You decide.

References

1. Shneiderman, Ben. *Software Psychology: Human Factors in Computer and Information Systems.* Little, Brown. Boston. 1980.
2. Curtis, Bill. Managing the Real Leverage in Software Productivity and Quality. *American Programmer,* Vol. 3, July-August 1990.
3. Schulmeyer, Gordon. The Net Negative Producing Programmer. *American Programmer,* Vol. 5, June 1992.
4. *Chaos.* The Standish Group. Dennis, Mass., 1995. Also at http://www.standishgroup.com/chaos.html
5. Paulk, M. et. al., Capability Maturity Model for Software, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa. 1993.
6. Radice, R., *ISO 9001 Interpreted for Software Organizations,* Paradoxicon Publishing, 1995.
7. Washington Post classified ad website. 10/01/99, www.washingtonpost.com/wp-adv/classifieds/careerpost/front.htm

Note

1. If you have a preference for mathematics, you can assign values to each of the listed skill areas as follows: Essential–8 points, Important–4 points, and Useful–2 points. These values sum to 100 points.

About the Author

Dr. Richard Bechtold is a principal consultant who supports industry and government in the analysis, design, development and deployment of improved software management, engineering, acquisition, and risk-reduction processes. He assists clients in process analysis, modeling, definition, training, and deployment. Bechtold also assists government acquisition agencies in the systematic evaluation of contractor software process capability. He has more than two decades of experience in the software industry.

He is an adjunct professor for George Mason University, where he teaches software project management and software process improvement to masters and doctorate students. He has written more than two dozen publications relating to software project management, software process improvement, risk management, acquisitions, logistics, and related topics. His latest book, *Essentials of Software Project Management* was published in August 1999.

Bechtold received his doctorate degree from George Mason University.

Dr. Richard Bechtold
Abridge Technology
42786 Oatyer Cr.
Ashburn, Va. 20148-5000
Voice: 703-729-6085
Fax: 703.729.3953
E-mail: rbechtold@rbechtold.com
Internet: www.rbechtold.com

Call for Articles

If your experience or research has produced information that could be useful to others, CrossTalk will get the word out. We welcome articles on all software-related topics, but are especially interested in several high-interest areas. Drawing from reader survey data, we will highlight your most requested article topics as themes for future issues. In future issues of CrossTalk, we will place a special, yet nonexclusive, focus on:

Network Security
October 2000
Submission deadline: June 1

Software Acquisition
November 2000
Submission deadline: July 1

Project Management
December 2000
Submission deadline: August 1

Modeling and Simulation
January 2001
Submission deadline: September 1

Configuration Management
February 2001
Submission deadline: October 2

We accept article submissions on all software-related topics at any time; issues will not focus exclusively on the featured theme.

Please follow the Guidelines for CrossTalk Authors, available on the Internet at www.stsc.hill.af.mil.

Ogden ALC/TISE
ATTN: Heather Winward
7278 Fourth Street
Hill AFB, Utah 84056-5205

You may e-mail articles to features@stsc1.hill.af.mil.
or call 801-775-5555 DSN 775-5555.