Microsoft's IT Organization Uses PSP/TSP to Achieve Engineering Excellence

Carol A. Grojean
Microsoft Corporation

Projects today are beset with problems from the very beginning. Many of these problems come from outside the team, be it over-ambitious deadlines from management, last-minute scope changes from the customer, or not enough resources. But not all problems can be blamed on these issues; many of the projects' problems come from within: failure to plan, failure to track actual progress against the original plan, failure to include changes in the plan, poor estimation, and failure to understand when effort deviates from the plan. The Personal Software Process^SM teaches engineers that estimating and planning are a big part of their job, and that what they predict drives all other efforts. At the Team Software Process^SM level, we bring the team together to define their processes and make a detailed plan. The process then requires the team to enforce their commitment as well as their individual behavior to follow the process to track time, use data, and get high quality.

The success of organizations that produce software-intensive systems depends on well-managed software development processes. Implementing disciplined software methods, however, is often challenging. Organizations seem to know what they *want* their teams to be doing, but they struggle with *how* to do it [1].

Unfortunately, when consistently challenged with a schedule, the defect elimination process continually gets pushed later and later into the development cycle and, eventually, over-the-wall to test so the engineers can continue to be *productive* and meet their schedule-driven goals. Many organizations fail to truly understand the impact poor quality has on their ability to meet schedule commitments.

Peter Russo, general manager for Microsoft's information technology (IT) application architecture group comments that:

> There are two fundamental issues in most IT organizations today, one being the ability to accurately predict a project schedule, and the other being the quality of the product once you are finally done – and these are two challenges we have to start addressing today.

Data Analysis and Findings

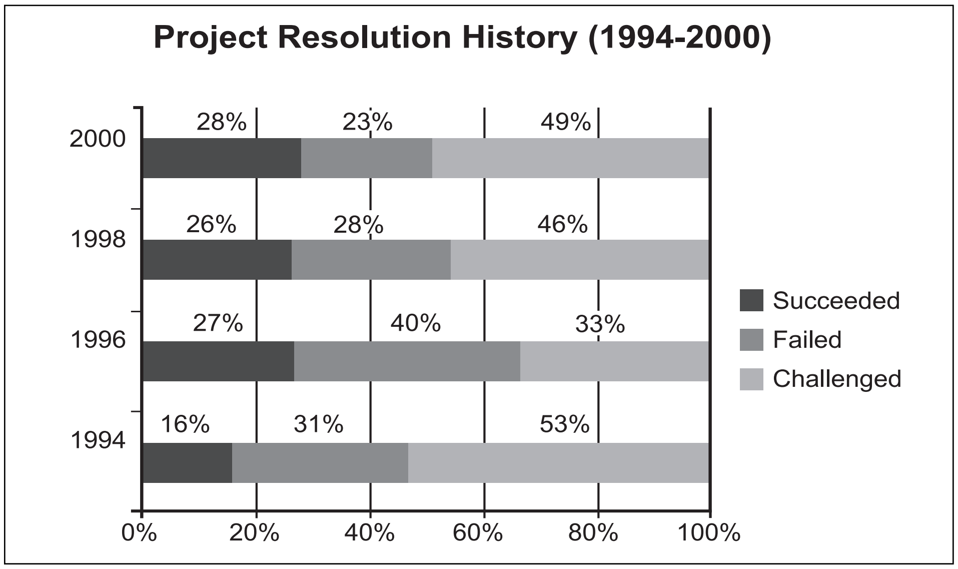
Most software organizations are facing critical business needs for better cost and schedule management, effective quality management, and cycle-time reduction [2].

In 1994, the Standish Group reported in their famous "Chaos Report" [3] that only 16 percent of projects succeed while 31 percent fail and 53 percent are significantly challenged, with the average project running approximately 189 percent over schedule (see Figure 1). In 2000, while things appear to be getting a little better, we still have a ways to go.

Furthermore, the Standish Group cites that while numbers appear better, that is not the entire story: many projects are overly estimated.

> In a number of focus groups, IT executives told us that they first get their best estimate, multiply by two and then add a half! It should not be surprising, therefore, that the majority of these successful projects were already 150 percent over budget before they began! [3]

Figure 1: *Project Resolution History [3]*



Most organizations have a schedule and a fairly detailed plan, and generally their plan goes one of two ways: either they have a development cycle followed by a test cycle where they expect the engineers to just fix bugs during the test cycle, or their plan has the engineers move on to developing the next set of features, setting aside a little time in case any bugs come up. In either case, the amount of time the engineer needs to spend regressing and fixing defects found by test is woefully underestimated every time. Additionally, once a product is released to the customer, management assumes the engineer is free to move on to the next project or cycle of the product. The plans generally do not consider that defects from the first release will take engineers away from their progress on the second release – this is how the vicious cycle begins.

The gap of quality code means that engineers are spending too much time fixing bugs, either from the previous release or the current release, on code already passed along to test. In doing so, they cannot make progress on new work as originally planned. This conflict creates a tension in the cycle where there needs to be a balance between injecting defects and removing them. The current process of engineer-injecting and test-removing is not a natural balance: The system pushes back by surfacing all the defects that testing cannot find to later in the cycle. The only solution is to understand that balance has to exist within the development cycle, which Personal Software Process^SM (PSP^SM)/Team Software Process^SM (TSP^SM) helps us understand.

Life at Microsoft

As it turns out, life in Microsoft's IT organization is not a whole lot different than many other IT departments when it comes to development. Many IT managers face decreased headcount and budgets with increasing support costs, as well as projects with unpredictable schedules and lower quality than they would like to see. As IT manager Todd Baumeister puts it:

> Today's projects are estimated and managed not with data, but on the gut instinct of the developer – and with this I have to go to the customer with conviction to our project schedule and then refute their asks of wanting more in less time, and this isn't a position I want to be in … I want a predictable schedule that will demonstrate our ability to plan for a date and hit that date and when we deliver the product it will be of high quality.

Every other IT manager I interviewed had a similar story to tell, and many added that they would like to see their team's morale increase. Another IT manager stated that he would like to basically "deliver a high quality product on a predictable schedule (when we said we would) without any death marches or dead bodies left behind."

Fundamental to all was the need to get developers focusing on features and design and delivering a high-quality product so that test can focus on performance, reliability, security, and ensuring the customer's needs are met, not this finding-fixing-testing loop that exists today.

About two and a half years ago, Microsoft IT Manager Aidan Waine was on a plane from Seattle to Reno reading Watts Humphrey's book "Winning with Software." Waine's development projects were out of control with high bug counts, ever-increasing test cycles and low delivery predictability. Client satisfaction and engineering team morale were both low. This book directly addressed these issues, describing controlled high-quality software delivery through disciplined, repeatable, data-driven engineering processes. Waine bought another 28 copies for his management team and clients. He jokes that no one read the book or took him seriously, so he went one step further and brought Humphrey out to Microsoft. Senior management bought into the experiment – two development teams were trained in the new methods, and two TSP projects launched.

Results so far have ranged from good to amazing. Waine said that the very first project he ran in his newly organized team more than paid for the training. His initial reaction was, "Wow, we're giving them two weeks of training and look what we got back!" The result of their first project, though small, was all they needed.

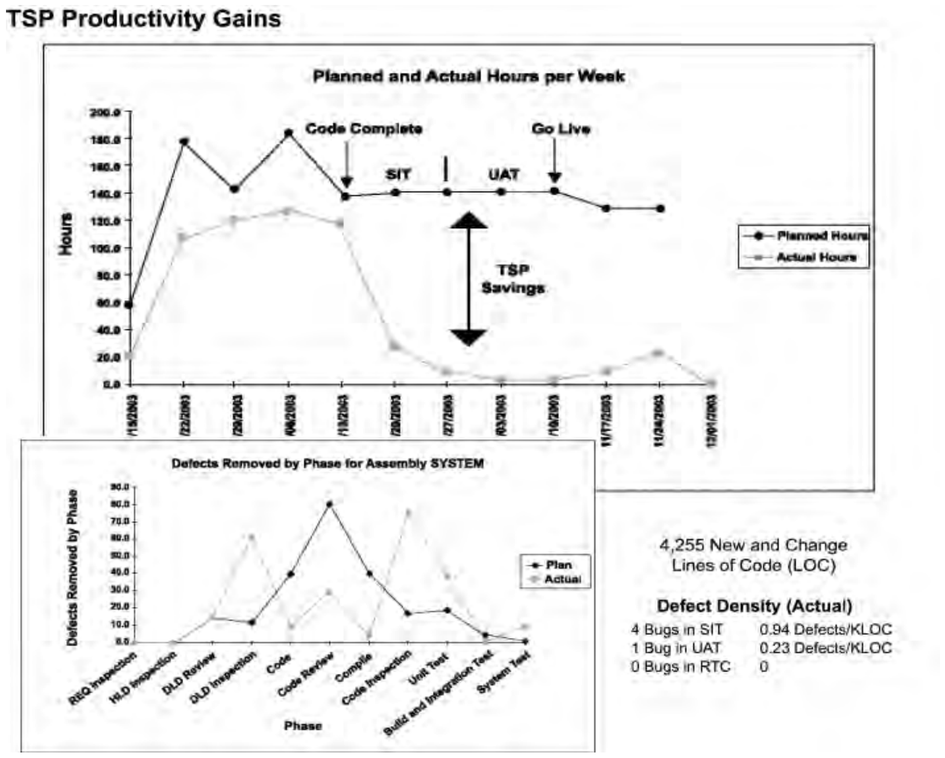


Figure 2: *TSP Productivity Gains*

As you see in Figure 2, for 4,255 new and changed lines of code, Waine's team had only five defects from the time the product went to system integration test (SIT) to user acceptance test (UAT) to released to the customer (RTC). The engineers were bored in test – can you imagine that?

When asked what challenges they faced once they made the decision to pilot or deploy PSP/TSP, all the managers interviewed were senior enough that they understood that they needed to lead by example, and that executive support and involvement is the No. 1 driver in project success. They understood that what they were asking for was a change of behavior from the individuals as well as something that goes against the culture of the organization, not to mention the software industry itself: collecting your data around time, size, and defects.

For the most part, they also mandated the change. This understanding and executive sponsorship sent a common and consistent message that change was expected and as a result, they faced very little political pushback. The biggest administrative challenge they faced was finding the time to get the engineers trained. "People don't come off the shelf PSP trained," one manager exclaimed. However, they understood that this had to happen, and that no time was going to be a good time. Moreover, they found it increasingly hard to find good, available coaches to lead their teams.

Additionally, there was the management challenge of convincing the business to take a four-week hit (for training and launching) with some sort of promissory note that it would be worth it in the end – a story they had been sold before. Fortunately, as more and more projects were finishing up, customers began to hear success stories from their counterparts and asked for the same results on their IT-driven projects.

"I'm excited about this," says Microsoft Chief Information Officer Ron Markezich, referencing a recent Accenture study titled "Value Discovery: A Better Way to IT Investments," a survey of 100 large European IT shops that showed astonishing results [4]. Up to 60 percent to 70 percent of most of an IT shop's budget is spent on sustainer activities such as support and fixing bugs. Markezich said:

> I'm excited because the potential for this to not only reduce our product cycles and increase our quality, but ultimately freeing up much of our sustainer activities enables us to invest more in builder activities that drive more value to the business.

What Do the Engineers Think?

A little over a year ago, I had the opportunity to launch an internal tools team that was literally broken down, having just come off a project that was cancelled after two and a half years of effort. According to Software Engineer Vivek Rao:

> I had just started at Microsoft when our team decided to adopt TSP/PSP. I did not know then, but would learn later, that the team was lagging in their interface with customers and was not listening to their needs. The team did the two-week PSP training of which I was a part. In the beginning, I was very skeptical of the idea, since it seemed like too much process to me. I felt it would stifle innovation. However, during the training I saw the importance of spending time to review my design and code. The satisfaction that I was getting by having zero defects in compile and an essentially defect-free program was invaluable. I also realized that I was a lot more confident about the code than I ever was. There was clearly an idea of design, code, and review both and be done with it, rather than keep fixing bugs later on, forever. I quickly realized the contribution of the PSP process toward quality of the code.

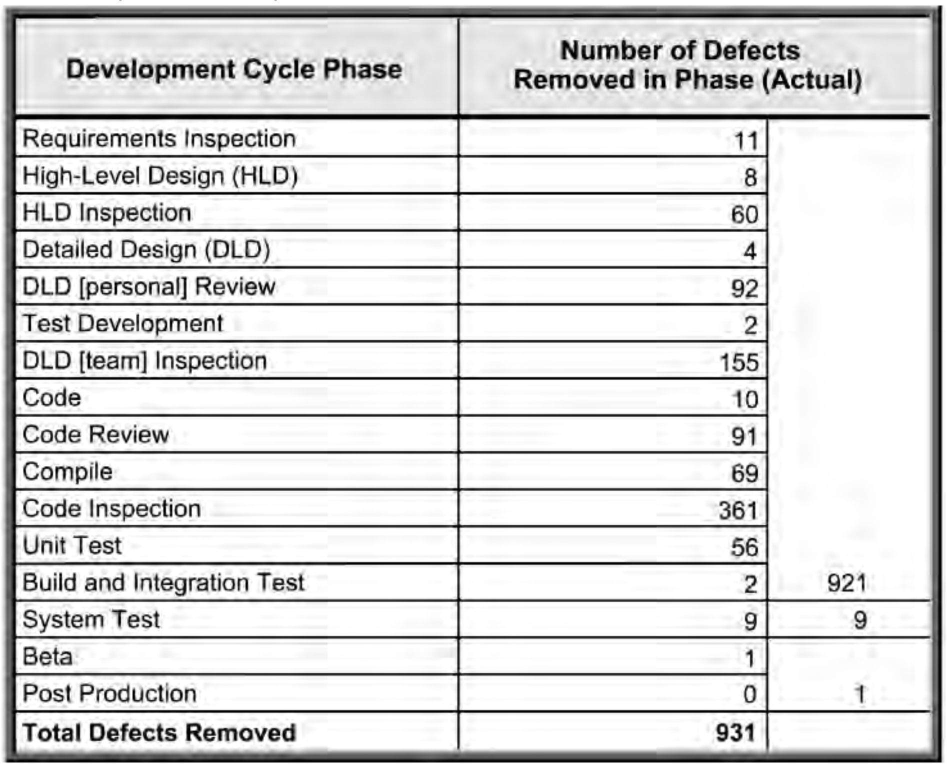
"It is very hard to convince people to change until they have to," said Watts Humphrey, "and the power of the PSP is that they see things change effectively" [5]. With PSP, the engineers honestly begin to see the change and begin to understand that to do good work you have to be disciplined, but change is hard. "People want to fight change and fight the notion of tracking time, size, and defect information," Humphrey said. "Most engineers just want to write code and don't perceive the other phases of the project (planning, design, reviews and inspections, etc.) as value-added, though they generally know they are good things to do."

That is the good part about PSP/TSP: earned value helps measure time all the way through the product development cycles, and people realize the importance of planning before design, design before coding, code reviews and inspections, etc. They also start seeing little changes right away – a code compiles without defects (or many fewer than before) or a test is virtually defect-free, which is something they never believed possible before. When the product gets to test and they are not scrambling around fixing bugs, they quickly become believers. Furthermore, they do not spend a lot of time arguing severity or priority of a bug because they have time to fix everything.

Vivek stresses this and goes on to say:

> The team then went into launching the project, what TSP calls a launch. The amount of energy that was generated during the launch was amazing – it really gelled the team together. The launch made the customer requirements clear and helped the team obtain buy-off on the schedule from the customers. It also helped me see the big picture of the project and my dependencies. During the launch, we also modified some of the processes to fit our needs, and it was clear that TSP could be modified to suit a team's needs. We have made small changes to the process over time to make it more efficient for us, while at the same time ensuring quality.
>
> As the project proceeded, I quickly realized one additional benefit of TSP: as a newcomer to the team, I was not familiar with the code base. I started taking part in many design and code reviews and because of this, I learned a lot about the code from experienced team members. For a new hire, TSP is an excellent means to learn about the design and code of a product in a short period of time. TSP also generated in the team a new sense of ownership and commitment toward the customer needs. Although this is not a direct result of TSP, it has resulted in the team scoring a 10/10 in customer relations.
>
> To summarize, the proper application of PSP leads to an immense sense of achievement and satisfaction, and TSP furthers this to the team level, ultimately resulting in good products.

Table 1: *Defect Removal Profile by Phase*



Development Cycle Phase	Number of Defects Removed in Phase (Actual)	
Requirements Inspection	11	
High-Level Design (HLD)	8	
HLD Inspection	60	
Detailed Design (DLD)	4	
DLD [personal] Review	92	
Test Development	2	
DLD [team] Inspection	155	
Code	10	
Code Review	91	
Compile	69	
Code Inspection	361	
Unit Test	56	
Build and Integration Test	2	921
System Test	9	9
Beta	1	
Post Production	0	1
Total Defects Removed	**931**	

And the results of their first pilot speak volumes:

- Ninety-six percent schedule accuracy – finished two weeks late with three weeks of added features.
- Delivered 1.36 defects/thousand lines of code (KLOC) to system test.
- Huge improvement in partner satisfaction.

Specifically, their defect removal profile on 12,253 new and modified LOC looked like Table 1.

On 12,253 new and modified LOC, the team spent approximately 584.6 hours in review and inspection phases of the project (otherwise known as appraisal cost of quality), and spent 109.1 hours total for compile, system test, and user acceptance test phase (otherwise known as failure cost of quality) for a total of 693.7 hours finding and removing defects. This represented an appraisal to failure ratio of 6.36[1].

Prior to system test, they had removed 921 defects – a yield of 98.6 percent (meaning 98.6 percent of the defects injected into

Phase	Number of Defects Into Phase	Yield	Defects To Be Fixed in Phase	Average Time to Fix (each bug)	Total Fix Time	Actual Number of Defects	Actual Time Spent by Team
System Test	460	50%	230	4 hours	920 hours	9	28 hours
User/Beta Test	230	50%	115	8 hours	920 hours	1	6 hours
Release to Customer	115	50%	58	12 hours	696 hours	0	0 hours
Total					**2,536 Hours**	***versus***	**34 Hours**

Table 2: *Comparison of Typical Results*

the work product prior to system test were removed prior to system test). If the industry average were to be about 50 percent (many feel it is not that good, but I will be conservative) and had this team been average, then approximately 460 of the 921 defects would have slipped to system test or later phases. If system test had a 50 percent yield, then they would have had to remove at least 230 defects in system test. Time wise, most teams I coach plan for defects in system test to take about half a day to find and fix (on average), meaning they would have spent 920 hours finding and fixing defects in system test instead of the 28 hours that they actually spent.

Additionally, another 230 defects would have slipped to user/beta testing where, with another 50 percent yield and a cost of about a day, or eight hours, to find a fix, you have to find and fix an additional 115 defects at the cost of eight hours per defect (on average) to find and fix or a total of another 920 hours. Instead, they had no defects.

That leaves the product going to the customer with 115 defects which, on average, 50 percent of those will be found over the product lifetime (or a total of 696 hours to find and fix). That is a difference of 2,536 hours of finding and fixing bugs in our old way of doing things versus 34 hours they actually spent (a project savings of 2,502 hours). That is the difference of spending three-quarters of a week fixing bugs post-development versus 5.76 weeks for 11 engineers. This is illustrated in Table 2.

What Does the Customer Think?

If your IT organization is anything like ours, then you are probably continually pushing for change – whether it is with tools or the latest methodology or a new definition of your project life cycle. You are constantly striving to figure out how to get more out of your people for less.

I can only imagine what our internal customers thought when we approached them with this new process.

Cyndee Kraiger has been in Microsoft Operations for more than 11 years, and has been the recipient of many IT projects in that time. About 18 months ago, she pushed for a new tool to manage the operations for our volume license deliverables. This tool was to replace an extensive set of spreadsheets that had become so unmanageable that even a small mistake could cost Microsoft hundreds of thousands of dollars. While this project was the second TSP project for the Business Unit IT organization Kraiger was in, it was the first for her and she did not really know what to expect.

Kraiger had been through several projects before as the business owner. She said that this project had a different level of engagement than traditional projects from the very beginning. "More of my time was required but the content of the meetings were of high quality." She said that she was even given examples of what to expect from the project up front. "My team was engaged daily and felt very involved and committed at all times."

Other projects, she said, typically start out with a meeting in the beginning of the project with some sort of expectation being set (time, features, cost) and then another meeting in the end with what was developed. Of course, compromises happened along the way, generally without her knowing it. But with this project, her expectations were managed all the way through and at the end of the project, she was putting the bow on the wrapping, (not her typical experience). There were no surprises. The final result is in Table 3. The project was delivered on schedule. As you see, "I got what I wanted when it was promised and the product was of high quality," said Kraiger.

Summary

It would not be fair to blame all software problems on software developers. When we are consistently challenged with a schedule, the process of eliminating defects continually gets pushed back later and later into the cycle and, eventually, over-the-wall to test so that the engineers can continue to be *productive* and meet their schedule-driven goals; this is classic *shifting the burden* [6].

In this environment, we have our quick fix of giving the code to the test team to be fixed. Then we have the unintended consequence of defects coming back to us later to eventually fix, at the expense of the current project, which then falls behind. However, the pressure to continue to meet schedule milestones continues, and so the behavior continues as a project's schedule begins to atrophy in its ability to meet established schedules, quality statements, and/or features.

It is just too expensive for any organization to try to test quality in; it cannot be done. And without being able to accurately predict our project schedules and resource needs, we just cannot run our organization.

Jon DeVaan, senior vice president of Engineering Strategy at Microsoft frequently references the article "Nobody Ever Gets Credit for Fixing Problems That Never Happened" [7]. The article addresses the reality every manager faces: dedicating additional effort to either work or improvement can increase the performance of any process. The issue at hand is do you go down the destructive work-harder loop, where you feel short-term gains despite long-term consequences, or do you follow the constructive work-smarter loop, where you feel short-term pain for long-term investment in capability?

What strikes DeVaan most about this article [7] is the story about the BP team that reduced butane flare-off to zero in just two weeks, saving $1.5 million per year at a cost of about $5,000 to implement, creating a return on investment of 30,000 percent per year. The article reported that members of the team had known about the problem and how to solve it for eight years. They already had all the engineering know-how they needed and most of the equipment and materials were already on site. What had stopped them from solving the problem long ago? The only barrier was the mental model

Table 3: *Illustration of Final Results (Quality)*

New Lines of Codes	59,616	
Number of defects found in System Test (ST)	42	(0.705 defect density)
Number of defects found in User/Beta Acceptance Test (UAT)	5	(0.084 defect density)
Number of defects found in Production (to date)	9	(0.151 defect density)

Coming Events

April 4-6
Defense Technical Information Center (DTIC) Conference
Alexandria, VA
www.dtic.mil/dtic/annualconf/

April 5-7
Federal Office Systems Exposition (FOSE) 2005
Washington, DC
www.fose.com

April 18-21
2005 Systems and Software Technology Conference
Salt Lake City, UT
www.stc-online.org

May 2-6
Practical Software Quality and Testing (PSQT) 2005
Las Vegas, NV
www.qualityconferences.com

May 14-15
ACM Symposium on Software Visualization
St. Louis, MO
www.softvis.org/softvis05

May 15-21
27th International Conference on Software Engineering (ICSE)
St. Louis, MO
www.icse-conferences.org/2005

May 16-20
STAREAST 2005
Orlando, FL
www.sqe.com/stareast

May 23-26
2005 Combat Identification Systems Conference
Portsmouth, VA
www.usasymposium.com/combatid

(thinking) that there were no resources or time for improvement, that these problems were outside their control, and that they could never make a difference.

DeVaan emphasized that people should have the courage to change:

> Generally, most people know what the problem is and perhaps even how to fix it; the difficult part is just getting people to change. Everyone recognizes the problem and oftentimes it gets expressed over and over again in cynicism. The true insight is getting every level of management to understand that they are part of the problem when they continually reinforce the *work-harder* loop.

DeVaan further expressed that it takes a lot more guts to change the lower in the management chain you are. "At some point there has to be a line drawn where any management above the line is to the point of negligence for letting the behavior continue." He said that we need people to have the courage to take the heat when the drop (in productivity) is down, whether it comes from the board of directors, the chief executive officer, or the line manager.

We all have to get on the *work-smarter* track and recognize that long-term gains in process improvement do not come overnight – just as they were not created overnight.◆

References

1. Humphrey, W.S. "A Discipline for Software Engineering." 2nd ed. Manuscript submitted for publication. 2004.
2. Humphrey, W.S. Winning With Software: How to Transform Your Software Group Into a Competitive Asset. Boston: Addison-Wesley (Pearson Education), 2002.
3. Standish Group International, Inc. The Chaos Report. Standish Group International, Inc. <www.standishgroup.com/sample_research/PDFpages/chaos1994.pdf>.
4. Curtis, G.A., R. Melnicoff, and Tor Mesoy. "Value Discovery: A Better Way to IT Investments." Outlook 2003, No. 3 <www.accenture.com/xdoc/en/ideas/outlook/3_2003/pdf/info_technology.pdf>.
5. Humphrey, W.S. Personal interview. Aug. 2000.
6. Senge, P., et al. The Fifth Discipline: Strategies and Tools for Building a Learning Organization. New York City, NY: Doubleday, 1994.
7. Repenning, N., and J. Sterman. "Nobody Ever Gets Credit for Fixing Problems that Never Happened: Creating and Sustaining Process Improvement." California Management Review 4 (2001): 64-88 <http://search.epnet.com/direct.asp?an=5244741&db=bch&loginpage=Login.asp&site=ehost>.

Note

1. The appraisal-to-failure ratio is a measure of the cost of quality. Specifically, you want to measure the percentage of time you spend in appraisal phases of your cycle (such as design and code reviews and inspections) versus how much time you spend in failure phases (such as compile, system test, and customer test). Appraisal cost of quality (COQ) percentage is calculated as 100*(appraisal time)/(total development time). The failure COQ percentage is calculated by taking 100*(failure time)/(total development time). The total appraisal to failure ratio is then calculated by taking the percent appraisal COQ divided by percent failure COQ (percent appraisal COQ)/(percent failure COQ).

About the Author

Carol A. Grojean is a certified Personal Software Process℠ (PSP℠) instructor and Team Software Process℠ (TSP℠) Launch Coach and has been practicing TSP at Microsoft since May 2002 when she was the team leader of the company's first pilot project. Grojean was in Microsoft's IT organization for eight years before moving in to her current role as a member of the company's Quality Engineering Practices organization, helping to drive engineering best practices throughout the product group, including piloting PSP/TSP. Grojean has a Masters of Business in management information systems and a Masters of Science in project management and is a certified Project Management Professional.

Microsoft Corporation
One Microsoft WY
28/1230
Redmond, WA 98040
Phone: (425) 706-8903
Fax: (425) 706-7329
E-mail: cscott@microsoft.com