An Interview With Watts S. Humphrey

With more than 50 years in software and countless CROSSTALK *articles, Watts S. Humphrey needs no introduction—but we will anyway.*

*After World War II and academic work at the University of Chicago, Humphrey led an engineering group at Sylvania Electronic Products. Humphrey then joined IBM in 1959, where he worked on everything from fixing the OS/360 to leading projects as Director of Programming and Vice President of Technical Development. After retiring from IBM in 1986, he joined the SEI, where he established the Software Process Program, led development of the Software Capability Maturity Model, and introduced the Software Process Assessment and Software Capability Evaluation methods. Humphrey also led the development of the Personal Software Process*SM *(PSP*SM*) and the TSP. At a White House ceremony in 2005, President George W. Bush awarded Humphrey the National Medal of Technology. Known as the "father of software quality," he is also the author of 12 books—with another one, "Leadership, Teamwork, and Trust: Building a Competitive Software Capability," on the way.*

CROSSTALK *talked with Humphrey, who delved into his past and present work, as well as discussed the future of CMMI®, PSP, TSP—and the software industry.*

Q: What were the personal experiences and values that influenced you while creating CMM, PSP, and TSP technologies?

Watts: Three experiences had a major impact on me.

Let me start with one of my first management jobs. I got hired by Sylvania in Boston to manage a fairly large circuit-design group that was building a great big cryptographic system. I had this group of young engineers all designing circuits, but I had been trained as a physicist and didn't know the first thing about circuit design.

Rather than fake it, I just spent my time asking them what they were doing and had them educate me. It was a different kind of management style than what people are used to. Usually, managers have done development work themselves, know how it ought to be done, and try to tell everybody how to do things. I couldn't do that. It was an education for me and highly motivating for the engineers. They loved it, and began to manage themselves. The exciting thing for me to discover was the fact that it's the only way you can manage really large groups. You discover when you get groups of hundreds or thousands of people—which I later did—that you can't manage what they are doing, so you need to count on them. That is the style I've used throughout my career. It's influenced everything I've done.

Basically you treat management as a continuous learning process, as a leading process, as a motivating process, and not as a directional process. So you're not telling people what to do—you're having them work it out and explain it to you and justify it. It really makes an extraordinary difference.

The second experience was at IBM where I was a crisis fixer. I found that the problems were never technical; they were always management problems. That's what I have struggled with trying to fix. I didn't know it then, but it's something I would be working on for rest of my life.

Fundamentally, you need to challenge people to prove to you that they are managing themselves: Putting motivation and accountability together turned out to be very effective. By and large I'd say that, with essentially no exceptions, all the crisis projects I led were successfully fixed. One example was an enormous project of 4,000 developers building an operating system for IBM. It was terribly late. We basically stopped everything for about 60 days and had them make plans, and it worked.

The reason I feel that the planning issue was critical comes from my third influence—my MBA education at the University of Chicago. For some strange reason, I decided to major in manufacturing. The manufacturing professor emphasized three things in management: planning, planning, and planning. Basically it's what he focused on throughout the whole course.

What fascinated me was that while hardware engineers have to work with manufacturing, the software engineers don't. The manufacturing people require plans, so the hardware engineers have to understand planning. The software people could manage their own work if they learned how to make plans and manage themselves. The CMM, the TSP, and the PSP all start with planning—it's the first step for everything you do. But software people are never taught how to plan. You can't just tell them to plan—you have to show them. That's a big part of what we do.

Q: As you expanded the PSP process to the TSP, did the industry develop at a slower or faster pace than you envisioned?

SM The Personal Software Process and PSP are service marks of Carnegie Mellon University.

® CMMI is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

Watts: Slower. I am reminded of a story about Fredrick Winslow Taylor[1]. More than a century ago he was working with a machine tool shop in England and invented the idea of lubricating the cutting tool: It was effective, it was very easy to introduce, and it cost practically nothing. He went back 20 years later and checked on machine shops in England and discovered that only one other shop was using it.

Now my point is that extraordinary methods that save an enormous amount of money are often relatively easy to put in place and are unbelievably effective—but they don't get adopted. You have to wonder why that is. I've concluded that there are three reasons.

First, every five years or so for the past 60 years that I've been working, some "magical new software method" comes along, and most of them don't work—except for the person who invented them. This has caused a lot of skepticism. Workable new ideas won't be believed until software engineers see them work for themselves. And they won't try it for themselves until they believe it will work—so you're stuck with a chicken and egg problem.

The second is that introducing new ideas is always difficult. When things are going well, organizations don't think they need to change—and when things are going badly they can't afford to change. So you're stuck and it takes people with great vision to see the strategic need to change even when they don't have to. Most only change when there's some pressure that makes them change.

And the third is that very few managers below senior levels are willing or able to take the initiative to introduce new methods—even when the benefits are obvious and proven.

Q: So what you're saying is they feel powerless.

Watts: Well ... they feel that way, but people really aren't powerless. I've talked to managers who have hundreds of people working for them and say they can't do anything. But I've been a manager with hundreds of people working for me and I've done it. I basically would just do what I felt I had to do and I'd tell my manager, "Here's what we're doing." I've always believed that if it makes sense and you can justify it, just do it and it almost always will work. Tell people in advance and don't ever surprise your boss, but I've always followed that old saying: "It is better to ask for forgiveness than permission."

Q: What are the other engineering areas to which you see the PSP/TSP expanding?

Watts: The PSP and the TSP are fundamental; they're not just about software. If you've read Peter Drucker, he started talking about knowledge work in 1959. Knowledge work is dramatically different from the typical work we do with our hands. It's work where you can watch knowledge workers but you can't tell what they're doing.

Now with typical engineering work—hardware engineering, manufacturing, construction, you name it—you can watch people doing their work and you can tell what they are doing and how well they are doing it. I mention in the article [in this issue], the method called "management by walking around" (MBWA): Managers go out and walk around the shop and see what is going on. As I say in the article, an increasing amount of the true work is done on computers and in people's heads. That's where the value is. And that is knowledge work.

So with a few exceptions, knowledge work is becoming pervasive—and why it's so extremely hard to manage some of these jobs. Managers don't know how to operate if they can't use MBWA—you can't use it with knowledge work. That's why software has been so extraordinarily hard to manage from the very beginning. Drucker's point was that the knowledge workers have to manage themselves, and that's what we are showing them how to do with the TSP.

Today's knowledge workers don't know how to manage themselves because they don't believe that they need to. They think that anything called management is the manager's job. So the knowledge workers don't manage themselves: they don't want to, they don't know how to, and they literally can't do it. As a result, knowledge work is not being managed—which means that projects often fail.

The TSP is designed for knowledge work. It's not just designed for software, it's designed for any kind of knowledge work—and we've used it with systems design, video game development with artists and game designers, as well as with software people and hardware groups. The TSP is universally helpful as a management system for just about any kind of complex work. One of the most impressive things to me was that we trained some Mexican software engineers to be TSP coaches—and they returned to Mexico and started a business. They've now grown that business to nearly 400 people, and they expect to be at 1,000 in a few years. They're even running their corporate office with the TSP.

The places where it can be used are almost limitless—and the benefits in software are so extraordinary that that's where knowledge workers are most likely to see the benefits of self-management. In software, you can cut test time incrementally. We've seen organizations that have been spending a year in test; with the TSP, they now spend a month and a half in test. You don't quite see that in the other areas, but it is equally applicable. The TSP is universally helpful as a management system for just about any kind of complex work.

Q: On a side note, our organization just finished a six-year training program where we trained more than a thousand leaders here and—I'll just be honest—we stole heavily from those concepts to get folks outside of software to use those types of disciplines. The thing that I noticed is people seem to inherently be attracted to the PSP/TSP—I mean, it just makes sense to them.

Watts: Well, people love it: Their morale goes up, they are excited about it, and their jobs are much better. PSP-trained people have to know how to manage, estimate, and track their own work—personally. If they don't know how to plan, they don't have the foundation required for self-management. That's why the training is so extraordinarily important. But managers just want to read a book on "how to be a team." That doesn't work because they don't really understand planning, management, and tracking, or how to control a project.

Q: What is your vision for using the TSP concepts in whole organizations and how would they manage such an effort?

Watts: There are two ways to look at this. Let me first talk about a big company. I recommend that organizations start with a modest-sized area and run a few teams. Typically, they'll start with multiple projects of six to 12 people—which are great for TSP teams—and we have them run those projects and start building skills. Then, broadening TSP use across the business is purely a question of how rapidly the organization wants to go and how rapidly they can build the management skills. The engineering skills can be built very quickly: In one week, we can train an engineering team to use the TSP and be a TSP team. The problem is that it takes quite a while for management to understand what the TSP is all about. It's a change in management style, and changing management style in an organization is much more difficult. The engineers take to it like ducks to water—they just dive in. We typically have engineers refusing to work any other way after they've used it.

Second, there is the case where you've got an entire organization-wide program and the organization may have multiple companies, multiple locations, and it's all one great big job. When the people running these great big projects want to really know project status, the managers must go to the individual software and system design teams—and talk to the developers and ask them exactly where they stand on their schedules and how they're doing. Unfortunately, the teams can't tell them. They're guessing, they don't know. The project or team leaders may poke around and talk to everybody and they get vague stories. Then, when these managers talk to more senior management, they're guessing and defensive. They really can't level with management because they really don't know.

With the TSP, you don't sit down and argue and debate. You say: "Here is exactly where we stand, we've got these problems, and here we are." You discover that when you have the facts, your customers and your managers will work with you to solve problems. We have seen that with the Navy and with several DoD projects we've worked on. All of a sudden, instead of faking it, you know exactly what you're talking about. Nobody is guessing. You've got the data and you sit down and say, "Okay, we've got a problem to solve. How do we do that?" It is a totally different attitude.

In big programs you need to start at the base, get everybody using it, and have that whole attitude—of honesty, of leveling, of data, of facts—where you can really negotiate and deal openly with the management team. Building an environment of trust is absolutely crucial. We do not have that today in large programs because the facts aren't there to engender trust.

Q: PSP theory talks of experiencing a 100 defect per thousand lines of code (KLOC) defect density as typical for PSP software engineers (as well as the TSP teams they belong to) when developing software. With software teams using more powerful, context-sensitive editors for developing software, TSP teams with which we are familiar are seeing defect densities of anywhere from 50-70 defects per KLOC. Assuming this new "reality" exists beyond the teams we study, does this evolution in software development impact the way that PSP should be taught?

Watts: Well, the 100 defects per KLOC defect level is what we see when developers first start learning PSP. At the end of training, they're typically at 50-70 defects per KLOC or less. I've seen some down in the 20s. The numbers come down: People are using more disciplined methods and are aware of the defects they inject because they are measuring and tracking them. Just measuring and understanding the mistakes you make generates feedback real quickly. The numbers really do come down sharply when people understand their mistakes.

The second issue concerns the development environment. I've written some programs with .NET, with stuff like that. But those tools do not eliminate your errors and do not address the key problems with logic errors. All of these tools are designed to generate a working program from whatever the developer puts in, so it could very cleverly produce a working program from a highly erroneous design. Software people tend to think that when some tool fixes their trivial defects that all the defects are fixed. People have to be conscious of what they're doing and even more so when working with very sophisticated tools like .NET, where you can put in all these very complex functions that most people don't even understand. These languages are so complicated that very few people ever really understand them completely.

Q: Is it safe to say they are getting a false sense of security, as far as these defects go, by using some of these tools?

Watts: Yes, that is true. Powerful tools can lead to powerful mistakes. The whole process of designing tools and languages is aimed at richer and richer capabilities. Not a whole lot of attention is paid to understanding why developers make errors and/or how to design languages and tools that minimize human error. I've hoped that the academic community would look at this, but unfortunately no one has done it yet. It is time they started. We really do need that kind of help.

Q: You have stated that your personal goal since the mid-80s has been to transform the world of software engineering. In what ways have you succeeded in this, and what unreached goals do you hope to meet in the future?

Watts: When I retired from IBM, I made the outrageous statement that I was going to "change the world of software." You can never really do that sort of thing by yourself, but it was really motivating. What I found fascinating was that people got excited about it. They joined in. I got a whole movement going and it was marvelous. When you get people working with you, you can get a lot done. That has been exciting—and we've had some remarkable successes as well as some real disappointments.

Let me talk about the successes first. With the OS/360[2], after I took over that project [at IBM], we put together a new plan and we put in place the management systems I've been talking about. This was at a much earlier level—we didn't understand it all then. But we [then] didn't miss a single delivery date for two and half years. There are not many people who have done that with big software systems. We put out the first 19 releases of OS/360 on schedule. Take a look at Microsoft or anybody else today: They never deliver on schedule, but we did, and it made an enormous difference.

Okay, so that is the success. We now have a basic understanding of this stuff. We know how and why software costs, schedules, and quality have been out of control—and we know what to do to fix them. I'm not saying that we have solved every problem, but when we work with organizations, we can help them build the capabilities they need to consistently deliver quality products on schedule. And it works: We have seen it with hundreds of teams across many businesses; we've seen it work with small two, three, four person projects, up to great big multi-company programs.

What is so disappointing is the acceptance of these ideas. The defense industry hasn't really looked at this at all. One of the main reasons and one of the big objectives I had when I started this whole thing was to address this national need: We have these enormous programs and our whole defense industry and military preparedness is dependent on these projects getting completed—and on time.

During my very first SEI project, I was working with the electronic systems command. That's where we started the CMM, the predecessor of CMMI. Every project was failing. They were all behind schedule—on average 60 to 70 percent late, and costs were at least twice what was planned. There was a recent report in the New York Times[3] saying that two-thirds of the largest DoD weapons systems ran over their budgets and the combined extra cost was $296 billion dollars—and they were on average two years behind schedule. This is a tragedy. We know how to do better—we are just throwing money down the drain.

I had a major executive on a defense contract ask me, "You mean to tell me you want me to spend profit dollars to cut revenue?" These companies today are being paid to do crappy work and then fix it later. And with the current system, you and I as taxpayers just keep paying for it. The more junior-level managers can't fix it, and it isn't that they don't want to; my guess is that they would love to, but can't. They are measured on revenue and profit and they literally cannot reduce it. They can't spend profit dollars to train their people to do quality work and have their revenue go down. The whole structure does not allow them to do it.

"[The TSP] lets teams know precisely where they stand. They can give data to their managers, they can tell them exactly what is going on, and they can identify any problems."

The DoD is constantly struggling. They are trying to change procurement regulations and trying to change managers and get smarter people, but there will be exactly the same problems until they start to deal with the fundamental management system that's currently being used. And that is what the TSP does: It manages knowledge work. It's not dealing with the work as something you can walk around and watch, because you can't. Knowledge work is invisible to the managers and if we continue to operate in the same way, none of the band-aids the DoD is trying will ever fix it. I've seen it for 50 years. So it is not going to change in the next five, 10, or 20 years until leadership begins to realize that we have to try something different.

And the TSP is different. It lets teams know precisely where they stand. They can give data to their managers, they can tell them exactly what is going on, and they can identify any problems. Most crises in big programs are obvious: They are identified years ahead by somebody way down in the trenches. And usually those people don't feel that they own the project. They assume somebody else will handle the problems, so they just go on with their jobs—and the crisis blows up.

What is exciting about TSP teams is they actually do risk analysis and track problems. They take ownership and assign individual team members to track and manage them. We have standard roles for TSP teams—a customer interface manager, a test manager, a design manager, and others—and each of these roles is assigned to a team member. So you now have ownership at a team level; you have team members who will bring issues upstairs when they need to. Instead of hiding their problems and going on with their work, they're actually addressing issues proactively and getting management's help when they need to—and it works.

We need that attitude throughout these enormous programs in the DoD. It must start down at the root level. If you have it there, it will build all the way up to the executive level. When the managers know what they are talking about, you begin to get cooperation between the defense contractors and the DoD. And the DoD can now deal honestly with Congress. Right now, Congress doesn't trust the DoD because they can't get the facts and everything is a surprise.

The other disappointment is the academic community. With few exceptions, computer science and software engineering programs have shown no interest in the TSP and PSP. Until they start teaching this stuff, their graduates won't understand it and industry will have to re-educate their people—and that's expensive. The way people are working today, they're basically beating their heads against the wall, testing until midnight, in at all hours. Nobody likes it, it's a painful job, there are failures all the time—and it has become a very unattractive career.

The U.S. Census Bureau did a study some time ago—and unfortunately I don't have the reference—forecasting that within 10 years, 50 percent of the people doing software work would leave the field. These are enormously talented and skilled people who have had 10 years of experience that are just going off and doing other things. They make some money and then they leave. They can't take it. It's because they don't know how to manage themselves; they don't know how to work in this environment. The academic community really has to get on board: understand it and teach it. They ought to lead this charge.

Q: Do you see issues between the model community (CMMI) and the process community (TSP) and, if so, what are your thoughts on how to overcome such differences?

Watts: Frankly, in the past, we have had differences, but they were principally due to misunderstandings. The groups had not been working that closely together. We were basically on our own paths. Our challenge was to figure out how to make this stuff work. Now we've worked through these differences and see that the two approaches work together extremely well. The CMMI people now see that it works.

Fundamentally, one of the big problems CMMI has now is performance—performance for high-maturity organizations. The CMMI community is beginning to see that what we've got *does* work. And so we are beginning to work together; CMMI and the TSP are very complementary. We are now working as a coordinated group to figure out how we can better help people improve their organization's performance and accelerate process improvement.

Q: What is your opinion of the direction that CMMI appears to be headed for high maturity? Specifically, do you believe that we will see benefits from the kinds of Process Performance Baselines and Models for which lead assessors are now looking?

Watts: Let me talk about the two kinds of processes: procedural and operational.

CMMI is an excellent example of a procedural process. Fundamentally, it sets organizational standards and sets baseline procedures across a business.

The CMMI framework is exactly what we did at IBM: We defined standard milestones where the teams had to go through six project review steps before they could go out the door. They had to do this before they could get funded, before they could announce a product—that sort of stuff. We had steps that all the projects had to go through: guidelines for quality assurance, testing techniques, and inspection procedures. We established a review procedure and all the involved groups participated—the maintenance people, the marketing people, the support groups, and so forth. They all had to sign off. While this was a lot of bureaucracy, it forced the organization to do things that TSP teams do naturally. But it worked real fast. Since we were in a crisis, we needed that.

Before CMMI, we didn't have that sort of thing—everybody was off doing their own stuff, nobody had a standard framework, none of that. CMMI is extremely helpful in stabilizing an organization and getting a level of statistical control: It is repeatable; you can more or less get stuff to work in a predictable way. And so that is how CMMI—in Levels 2, 3, 4, and ultimately 5—stabilizes an organization and begins to build the kind of foundation you need for real improvement. So CMMI is what I call a procedural process.

Now you get to an operational process where you are talking about what the development teams do when they develop software. How do they do it? How do they manage quality and cost and schedule? What data do they gather? What measures do they use? We found that until you provide specific guidance to the developers, they won't do it.

Think of it this way: When you tell a developer, "I want you to make a plan," they don't have the vaguest idea of how to do it and they don't even know what one looks like. That's what I saw at IBM. Everybody was coding and testing. Everybody knew that they ought to have plans and requirements, and they knew that they ought to have all this other stuff in place—but they didn't know how to do it. I put a thousand managers through a course on how to plan. If we hadn't done that, the managers wouldn't have been able to make plans. We put that in place and it worked. It was extraordinary.

So the whole idea here of the procedural process is to build the base capability and then begin to move toward an operational process where people really do what they have to do to generate the data, manage the quality, and build the performance of the organization. So that's the distinction between the procedural process and operational process. With a procedural process, you usually need a bureaucracy to enforce it, but with an operational process—as long as the teams are properly coached—they can be trusted to do their work properly and the bureaucracy is unnecessary. So the tradeoff is coaching versus bureaucracy.

When we originally put together the whole maturity model framework, we were doing it for the acquisition community. We knew that we had to give guidance on the question, "What do you look for?" So we focused on artifacts. What is the evidence of an organization's performance? Say we want an organization that is producing plans and that uses configuration management and requirements management. What are the things that you'd have to have if you did that? You could say, "Well okay, if you do planning then you ought to have plans." You can now look around and say, "Do you have plans?", "Do you have review meetings?", and "Do you have review meeting minutes?" If you have configuration management then you ought to have configuration management audits, reviews, and updates so that there are actual artifacts produced as a natural consequence using the process. It shouldn't be expensive to produce. If you're actually using that process those are things you ought to naturally have; you look and make sure they're there.

When we put together the original maturity models, this is what we did. While the acquisition people didn't really understand the details, they could tell that somebody had a development plan, it was for this project, it was signed off, and it had what appeared to be the right stuff in it. It was fairly easy to do. The original intent was that these artifacts were the natural consequence of the process being used, so there shouldn't be a lot of cost involved in preparing for such a review.

Now notice what happened with CMMI: Appraisals became important so organizations were in a great hurry to reach a high maturity level. Increasingly, organizations discovered that it is extremely hard to change what the development teams actually do. It's a heck of a lot quicker to have task groups generate documents that meet the needs of the appraisal. So you've got groups that put together configuration plans, development plans, and all of this stuff. And it's not developed by the developers—but there is nothing in CMMI says that it's wrong to do this—so you've got all of these artifacts. Now you have these independent groups bureaucratically producing stuff that has no relationship to the work that is being done. And so you don't improve organization performance at all. Unfortunately CMMI, as currently built, doesn't protect against that. And so that's what we need to focus on: How do we work together so the CMMI and TSP folks really focus on what it takes to have a high-performance organization?

This is why the performance idea is so critically important. If you really are talking about data and measurement, you have to think about performance in a different way. You need to show that you not only have the artifacts but you are getting the performance the artifacts should produce. So that's the thing that we are talking about. Up to this point, when we've put people together, it has cut the cost and time for process improvement. It accelerates the movement from one level to another and produces dramatic performance improvements.

Q: Where do you see things going in the future? Do you see the DoD taking more steps to utilize the TSP?

Watts: In terms of where we are going, the future is exciting. I've found that even enormous programs can be managed. Can you imagine how our economy would work and how the DoD would function if people could actually put together plans for these massive programs and then delivered them on schedule and for their planned costs?

People don't understand when I say, "Delivering on cost and within schedule," that this will be a fundamental problem for the DoD. It doesn't mean that the teams can deliver on whatever schedule the politicians or generals demand, it means that the development teams themselves—when they know how to manage their own operations and put together their own plans—can go to the generals. Then the generals can go to the politicians and say, "Here is what it is really going to take." Instead of saying, "We're going to do it in 18 months," you may do it in 30 months, but people will actually deliver on schedule and they will meet their cost goals with quality products.

We are seeing that time and time again. We are seeing it with big teams and we've even seen it with multi-company teams where you have people working together across several companies. There was one case with two competing companies[4]—under a DoD contract—where they actually did deliver on schedule and the product really did work. We heard the customers say, "This is extraordinary. We're not going to work any other way." So we know it can be done.

So the customers will like it, the politicians will like it, the generals will like it, the users will like it, and we'll get a hell of a lot faster stuff out there to the fighting forces. It's a very exciting future. I hope I'm there to see it.◆

Notes

1. See <http://en.wikipedia.org/wiki/Fredrick_Winslow_Taylor>.
2. IBM's OS/360, officially known as the IBM System/360 Operating System, was developed for IBM's then-new System/360 mainframe computers. The multiple virtual storage version of OS/360 was the first large-scale general purpose operating system and it was one of the first to make direct access storage devices a prerequisite for their operation.
3. See <www.nytimes.com/2009/03/31/business/31defense.html>.
4. This project is detailed in the May/June 2009 CrossTalk article, "A Distributed Multi-Company Software Project," co-written by Humphrey with Dr. William R. Nichols, Anita D. Carleton, and James W. Over. See <www.stsc.hill.af.mil/crosstalk/2009/05/0905NicholsCarletonHumphreyOver.pdf>.