The Myth of the 10x Programmer

featuring Bill Nichols as Interviewed by Eileen Wrubel

Welcome to the SEI Podcast Series, a production of the Carnegie Mellon University Software Engineering Institute. The SEI is a federally funded research and development center sponsored by the U.S. Department of Defense. A transcript of today's podcast is posted on the SEI website at sei.cmu.edu/podcasts.

Eileen Wrubel: Hi. My name is Eileen Wrubel. I am a technical director in the Software Solutions Division here at the Software Engineering Institute. With me today is my colleague, Bill Nichols, who recently published research that challenged the veracity and the relevance of the widely held notion of the 10x programmer. This work has generated a lot of attention. So, Bill, I'm excited to have you here to talk with me about it today.

Bill Nichols: Thank you.

Eileen: Before we dive down into the blog post, and *IEEE* column, can you tell us a little bit about you and the path that you have been on here at the SEI that got you into this work?

Bill: I started out in particle physics. My Ph.D. is actually in physics. That is where I started programming. That is where I developed a more quantitative bent. Some of my friends at the time were also involved in things like sabermetrics and analysis. Later on, when I was working at the laboratory, I spent 15 years developing scientific codes for analyzing nuclear reactors. I became interested in how we could get better performance from our teams and our work. We went into a major reengineering program at the laboratory. I was leading a development team to reengineer a bunch of old codes. And Watts Humphrey and several others from the SEI came and talked to us, and they talked about the PSP [Personal Software Process]. I said, *Wow, this is exactly what I was looking for. This is a way to really apply the quantitative measures to software*.

Well, after working on this project for about five years at the laboratory, I ended up coming to the SEI, so I could see how things are working in the rest of industry, and start teaching others a lot of things I have learned about how to measure software, how to improve team performance, and basically how to get the most out of your software development.

Eileen: For our viewers and listeners who aren't familiar with PSP, that's the Personal Software Process, which is the individual component or counterpart to the TSP or the Team Software Process.

Bill: That's right. The TSP is what we actually used in the development. PSP was really a training mechanism. It was a technique to teach people how to record and use the data that they would actually implement on projects on team projects. How to do things like report your status, how to track your progress, how to measure improvements.

Eileen: That work was the genesis for the column that you wrote in *IEEE Software* and the resulting blog post that combats the myth of the 10x programmer.

Bill: That's right.

Eileen: Why don't we start there. Can you explain a little bit, in case we have viewers or listeners who are not familiar with this idea of the 10x programmer, what is the idea behind that? And why did you decide to take that on?

Bill: There are a couple of different ideas here. I want to be clear on which one I am actually addressing. There is this idealized concept of the x10 programmer that you sometimes hear. What they are really referring to are the super programmer, the Sidney Crosby or the Tom Brady of programmers, who can do anything. What I am looking at is a more literal definition. There have been many studies, many studies that actually measured the performance of programmers by giving them all the same program and measuring how long it took. They found pretty consistently the fastest was at least 10 times faster than the slowest. Now, I had a lot of data I could look at, but there were other things about that result that continued to nag and bother me.

Eileen: What factors did you look at when you started to dig into this and scratch that itch of the things that were bothering you?

Bill: We looked at a lot of different factors that we had in the data. But, what I really wanted to home in on is, *How long did it take people to write the same program*? So, we have this set of 10 exercises that a lot of different people had taken. I looked primarily for each individual program. They each did the same 10. *How long did it take them?* That was the primary measure. Although, there were lots of other measures that we have. We have defects. We have the size of the programs. We have the time they spent doing different activities. I was just looking at the beginning to end time. *How long did it take?*

Eileen: Did anything about this surprise you once you dug under the covers?

Bill: There were some things that were surprising. I mean I wasn't completely shocked, but when I looked at the scale of the difference, I was really surprised. The x10 programmer I said

was highly replicated. You've got it everywhere that you saw. Now, we kind of knew that might be a bit of an exaggeration. From teaching this class, we always know that the same people weren't always the first and the last finishers. There was some variation. So, when I started looking at the data, I did a fairly sophisticated type of analysis that looked at the variation. What struck me was the variation within a programmer's own 10 programs was comparable to the variation between programmers. That is, the *within* difference was as big as *between*. The scale of that that they were so comparable was really very surprising. I shared that with Tim Menzies, who was the editor for the IEEE column on redirections, and He said, *That's a really interesting result. But this is too abstract. It's too hard to understand. No one's going to understand how you did this*.

So [I] decided to make it a little simpler. What I did was, rank every developer on each program from one through 500. We had a sample of about 500 programmers who wrote in C. OK, 1 through 500. I did this for each of the 10 programs. Then I compared the range, and I found, *Oh, the average top and bottom performance of each programmer in this rank spanned 250, 250 out of 500.* That was an enormous range. That is what really surprised me, that this range was so large. The other thing that kind of surprised me was here we had this historic set of experiments. It started with Sackman. It's been replicated countless times. They always showed the same thing: the best and worst had this factor of 10.

I look at my data. I see that for each individual program. But then I look at the span of it, and I do things like, *How do the medians distribute? And, oh, it's not nearly as wide*. So, we had this funny situation where the research actually replicated. It wasn't a replication crisis type of problem. But it was still wrong, because no one had done the repeated measures. No one had actually seen the individual variation. Those huge variations they saw were at least half a result of noise from how long it takes me to write a particular program on a given day.

Eileen: Correct me if I don't get to this correctly, but my takeaway was that a lot of that variation was actually in terms of the individual and their performance on a given day and not so much the 1x individual and the 10x individual. And the environmental factors—I'll use that term loosely—in which they are operating: the teaming environments, the physical environment having quiet and appropriate spaces to work, that those are areas to focus to really achieve productivity gains as opposed to hunting for unicorn software engineers or try to hire rock stars.

Bill: I am actually going to start to address some of those questions in some future blogs. And the quiet space is really important. But the interesting thing was we gave these people quiet space. We saw this variation even without it.

When you start putting in real environments, the day-to-day differences become far more extreme, and that is what we saw with some of the TSP data.

That is what I'll start talking about in the next blog, some of the findings from that. If you are trying to make a productive software team, you're going to be hard-pressed to really up the productivity by finding that super-fast programmer. It doesn't mean that there aren't better programmers than others. We definitely saw differences. It just means that the range of differences between your average programmer and your best aren't that big, and the person who was the slowest on a given day isn't really going to be the slowest on the next day, very likely. So, the number of really super programmers is a lot smaller than you would have thought. You are going to have a hard time finding a team of them.

Most of the variation…It turns out the distribution isn't a bell curve. It's more log normal, long tail. *Why do people get stuck on a program?* It could be they chose a poor design and made something bigger than it had to be. They got into some bug that they just couldn't track. How often do you get stuck on a bug, and the first person who looks over your shoulder says, *Oh, you did this wrong?* One of the things that I would carry forth is not only do you have to create an environment, but you really have to create a team environment, so that you can catch those kinds of problems early. That is why we have some other recommendations. There are a few things that came out of the world that make a lot of sense, but never had a lot of theoretical foundation. I think the small tasks is something that this kind of data supports. *Why do you want a small task*? Well, it's easier to estimate. It's easier to know when it's gone off the rails, and in a team environment, it gives you more opportunities to step in and fix the problem before it becomes a runaway.

Eileen: So those small batch, fast learning-cycle kinds of approaches really seem to be borne out here.

Bill: That would be one of the indicators, yes,

Eileen: One of the things that we always like to emphasize in our podcast, and I know from working together, you certainly heard me talk about this a lot, is transition. So aside from the things that we have talked about with physical environment and collaboration and teaming environment, if I'm listener or viewer managing a team of software developers, what else might I want to think about or take away in terms of using these findings to help my team?

Bill: Well, among the things that I would look at is don't overreact to short-term variations. Variation happens for a lot of different reasons. As a manager, it is very hard to tell why that is. That is one of the reasons that we have always emphasized that the team has to be in charge of that process. The team has to be responsible for figuring out where the problems are because they are the ones closest to the problems. What the manager has to do is be able to see what are things going wrong, so that he or she can ask the right questions and get people to take corrective actions. That is probably the single biggest thing you can do.

I have some other recommendations as far as the tasks. This sort of thing raises questions about how long your sprint should be, for example. If this is your variation, this huge variation, well, it is no wonder people have a hard time estimating.

If you make your sprint too short, a lot of the compensation is, *Well, we are going to make the sprints shorter and shorter and shorter*. What happens when you reduce your sample size? The uncertainty becomes bigger. There is a point at which if you try to correct for these problems by just making your sprint shorter, you are going to make the problem worse.

Eileen: You get to that point of diminishing returns in there if you try to slice it too fine.

Bill: Yes, and because the estimates are so coarse to begin with, you may not even know where that is.

Eileen: Well, that makes sense. There is a colleague of ours who is very fond of saying, *You can get to a point where you are very, very, very precisely wrong by trying to slice it so fine*. What I heard is don't lose sight of the forest for the trees, and push decision-making authority down to where the information lives with the developers. Then, be able to take that broad horizon view and know what the signals are that something is going off the rails, but don't get caught in the small perturbations.

Bill: A lot of the part of the manager is really creating a productive environment. The trick is knowing what you can do that will make the environment more productive and more conducive to bringing your projects, your software development in on time.

Eileen: Well, great. Thank you so much, Bill. I'm looking forward to seeing the next set of blog posts that you talked about. Are you examining any other related areas associated with software development productivity?

Bill: Well, the project I am working out right now is called SCOPE. We are looking at the causal connections. *What are the factors that actually lead to productivity?* We are finding some interesting things. This project actually came out of SCOPE. It came out of a negative result, oddly enough. We were looking for what were the factors that lead to productivity. I was looking at things like, *Well, What about experience? What about total lines of how much code you wrote in a given language? What about degrees obtained*? Nothing mattered. That is kind of what led me into looking at this variation problem. We have lots of other data that we can look at with not only the PSP, but the TSP, and trying to understand what are the causal factors that lead to things like quality, consistency, productivity. It is not just speed. That is not the only factor we are interested in. We are interested in good programs. We are interested in getting them in a predictable pace. It's not just the fastest, but make it predictable. Make them predictable. Make them fast. We will be looking at some of those within SCOPE, and also on side projects. There

are lots of things in that TSP data that I would still love to investigate.be looking at some of those within SCOPE, and also on side projects. There are lots of things in that TSP data that I would still love to investigate.

Eileen: Great, I look forward to your continued work in this area and talking with you more as we uncover more pieces of the productivity puzzle.

Thanks so much for talking with me today, Bill. We will include links in the transcript to all of the references that we have made during this podcast. Thanks.

Thanks for joining us. This episode is available where you download podcasts, including SoundCloud, Stitcher, TuneIn Radio, Google Podcasts, and Apple Podcasts. It is also available on the SEI website at sei.cmu.edu/podcasts and the SEI's YouTube channel. This copyrighted work is made available through the Software Engineering Institute, a federally funded research and development center sponsored by the U.S. Department of Defense. For more information about the SEI and this work, please visit www.sei.cmu.edu. As always, if you have any questions, please don't hesitate to email us at info@sei.cmu.edu. Thank you.