November 12, 2010

Why Common Ground Is Critical to the Future of Software Engineering: And Why It Won't Be Easily Attained

I've been busy over the last few weeks. In early October I was in Milan for the 3rd Semat Workshop (http://www.semat.org/pub/Main/WebHome/3rd_Semat_Workshop_Report.pdf). The following week I spent helping a client assess an unanticipated problem—more about this in a moment. The next week I spoke at an IEEE Symposium and at Rochester Institute of Technology (RIT) (http://www.gccis.rit.edu/how-semat-can-change-future-software-engineering-personal-reflection). And last Saturday I spent the day in discussion with Dr. Tom McBride from the University of Technology, Sydney.

Tom is currently planning a number of research projects addressing issues related to Semat. We started our discussion by reaffirming the fact that Semat is not trying to compete with other industry initiatives including ISO standards and the CMMI. I pointed out to Tom that Semat seeks the essentials of software engineering—the common ground we can all agree to. I emphasized that "it has to be small—not hundreds of elements–maybe just twenty or so." Tom replied, "Good luck. It's not that easy. You can capture a small number of elements and get people to agree to them, but that won't provide real value."

If we had this conversation a few weeks earlier I probably wouldn't have had a good reply, but in this case I was quick to respond with; "The real value doesn't come from the essentials themselves, but rather from helping software practitioners recognize the conditions where they fail to apply them correctly, and the options they have in those situations."

I had been thinking about this in Milan where we demonstrated through the Architectural Spike how to translate Scrum into the Semat structure. At RIT I took this notion further motivating the need for Semat by comparing Scrum–arguably the most popular Agile Method– to TSP, a disciplined team software process developed by Watts Humphrey.

Many believe Scrum and TSP rest at opposite extremes on the agile-discipline method continuum. I told those in attendance—mostly software engineering students and RIT faculty– that what I was about to say was based on my experiences with clients and while some of it may be controversial, I suggested they focus less on my conclusions and more on the process I used. My experience from helping clients who have used these approaches indicates they may actually have far more in common than most realize. I identified nine critical practices both approaches shared.

Both Scrum and TSP have large supporting camps and both have growing bodies of evidence to support their success. However, not all projects that use these methods succeed, and when you look close at the projects that run into trouble the root causes often turn out to be, not what is different between these two approaches, but rather "common ground" elements that too often get missed. Let me explain this further.

In my book I have two relevant case studies I refer to as LACM and GEAR [1]. LACM developed one of the best approaches I have ever seen to integrate agility into their existing CMMI processes, but today they are experiencing unanticipated difficulties. Their approach fundamentally was to remove non-essential detail that was getting in the way of their people getting their job done. Eliminating the non-essentials from the process descriptions brings focus to critical business essentials. In my book I refer to these critical essentials as the "must dos." I have never found an organization that—once they try it– doesn't think this approach is on target.

However, due to business decisions LACM has fallen away from their focus on the "must dos." During a recent evaluation it wasn't difficult to connect current costly troubled projects to their repeating weakness of lost focus on critical essentials. As a result key weaknesses previously identified that could have been corrected are surfacing again and costing the company not just dollars, but potential future business.

GEAR is an R&D organization that is employing innovative process improvement approaches to rapidly achieve CMMI Level 3 in support of strategic business goals. Streamlined "must do" processes, and appropriate guidelines have been produced and

agreed to in the organization, but the process deployment phase isn't going as smoothly as desired.

While key stakeholders agreed to what every project "must do" at GEAR, the problem is every project isn't doing it. The power of "must dos" has been proven to work to achieve an appropriate balance of discipline and agility at a reasonable cost [2]. The "must do" approach is the same idea that Semat is trying to bring to the broader Software Engineering community through the concept of common ground, or the "essentials" of Software Engineering.[3] Until organizations try it they often don't appreciate how a focus on essentials differs from previous process approaches, nor are they able to reap the benefits.

I have been looking closer at why GEAR is struggling to follow its own agreed to common ground "must dos", and my initial findings are very interesting. In some cases people are reading more into common ground than intended. One simple example, a program manager was interpreting the word "plan" as requiring a comprehensive formal document, which was never the intent.

How do we address such issues? This is where coaching and guidance become critical to effective process deployment. One myth that I highlight in my book is relevant here:

Myth: If an organization is agile, it requires less process training.

At RIT, in a side meeting, Dr. Jorge Diaz-Herrera, the Dean of the College of Computing and Information Sciences, referred to a recent article by David Parnas related to the lack of discipline in software development and our failure to apply what is known in the field today. Parnas say, "Much of the fault lies with our teaching. Computer science students are not taught to work in disciplined ways," and he adds, "Disciplined design is both teachable and doable." [4]

There seems to be a tendency to think that essential common ground elements are so fundamental that they don't require training. There is also a tendency to think that to provide real value we need to make things complicated. But when we examine closely where projects get into trouble, the real value people need to help them get their job

done may lie in separating the essentials from the detail, as opposed to clouding them in complicated facts.

Today software development is not limited to the large software intensive projects that seem to garner much attention. Software is being developed in small organizations, by research scientists, electrical engineers and others. Software development is by no means limited to those in computer science and software engineering fields.

Parnas tells us, "Anyone who observes engineers at work knows that exercising due diligence requires a lot of 'dog work'. The dull, but essential, work begins in the design phase and continues through construction, testing, inspection, commissioning, and maintenance." [4]

There are essentials in software engineering, but we haven't brought them to light as in other engineering disciplines. Parnas tells us that, "Many of us preach about the importance of determining the requirements a software product must satisfy, but we do not show students how to organize their work so they can systematically produce a requirements specification that removes all user-visible choices from the province of the programmer. Some of us advise students to avoid dull work by automating it, but do not explain that this does not relieve an engineer of the responsibility to be sure the work was done correctly."

When you bring process maturity to an agile organization you actually need more, not less, training. This is because the documented processes cannot address every possible scenario, nor do we want them to because history has shown such detail can mask the essentials that are most important. Explaining to your people the likely scenarios, and the options they have in common situations is where the real power of common ground can be found. But it must be trained and guided, not hidden in long wordy documents. This is why I claim that common ground is critical to the future of software engineering, but it is also why I claim it won't be easily attained.

[1] Integrating CMMI and Agile Development: Case Studies and Proven Techniques For Faster Performance Improvement, Paul E. McMahon, Addison-Wesley, 2010

[2] Refer to BOND Case Study in Chapters 4 and 5 of [1].

[3] Refer to www.semat.org

[4] Risks of Undisciplined Development, David Parnas, Communications of the ACM, 10/2010 Vol 53 N