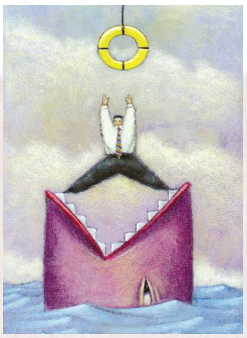
In Methods We Trust?

Luke Hohmann, SmartPatents Inc.

The degree to which you trust your environment—including co-workers, software tools, and systems—has a dramatic impact on the performance of your entire software development team. Consider communication. Better communication—higher bandwidth, if you will—is easier when trust exists. You won't feel as if you're missing anything when you walk past an important meeting, because you trust your co-workers to represent you well and to inform you of any key decisions that might affect your work. Furthermore, when you trust your co-workers, you feel more free to ask for help and to offer it. And if you can't trust someone during a code review, what are the chances the code review will be any good?

I've become increasingly aware of another aspect of trust: the trust of a method. In *Object-Oriented Analysis and Design with Applications* (Benjamin Cummings, 1994), Grady Booch defines a method as "a disciplined process for generating a set of models that describe various aspects of a software system under development, using some well-defined notation." More generally, I think of a method as a disciplined approach to problem solving that will produce one or more well-defined outcomes.

Editor: James Bach, ST Labs Inc., 3535 128th Ave., SE, 3rd Fl., Bellevue, WA 98006; jamesb@stlabs.com



The degree to which you trust your environment has a dramatic impact on the performance of your entire software development team.

There appear to be two philosophies when it comes to trust and methods. One camp soundly rejects methods as a basis for trust. This camp believes that the only valid approach to problem solving is to identify specific problems systematically in their environment and solve these problems, one by one. In other words, if people in this camp can't see a specific and direct need to generate a model or follow a process step, they don't. Quite often these people reject methods completely.

The other camp takes the opposite approach, performing every process step by preparing all of the models (and other outcomes) defined by their method. Sometimes developers execute this approach reluctantly in order to satisfy management. Other times developers or their managers embrace methods as a panacea for dealing with all the problems they face, thinking, "If I just perform each step, then we'll create the system our customers want."

So why is the issue of trust in relation to methods important? Ultimately, *all* software is built according to some process. It may not be defined or repeatable, as is the theoretical process prescribed by a method, but there *is* a process. Indeed, all human problem solving proceeds according to some process and is supported or hindered by some structure. The question is not whether you build software according to a process—you do—but whether you trust the process you use.

Do you trust your development process to generate accurate schedules? Do you trust it to generate high-quality, easily maintainable source code? Do you trust that your process will generate highly usable systems?

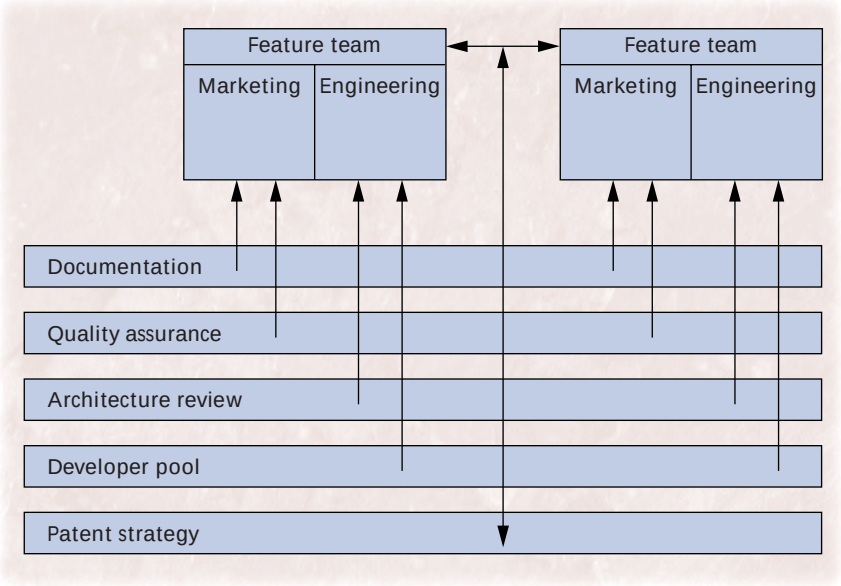
TRUST AND METHOD

The level of trust is not a property of a method or a process, but a relationship you have with the method or process. An example should help clarify what I mean. At SmartPatents, we don't have a method for hiring developers, in that we don't rigorously generate a set of models describing each candidate. But we do have a process. And we trust that process.

Interestingly, I've found that asking developers the degree to which they trust their software development process produces a wide variety of answers. I'm not asking whether they *like* it, but whether they *trust* it.

My own attitude toward methods is reflected in the following two observations:

- To assume blindly that the software processes espoused in a book will solve all of your problems is naive at best, and more than likely just plain wrong. No book can tell you how to deal exactly with the problems in your environment, and few methods, if any, are effective unless customized to the specific needs of your team.



This "feature team" arrangement is somewhat common among development organizations that produce commercial software. It is less common among other kinds of development organizations. The feature teams are given nearly complete latitude in deciding and controlling their own software process. Not shown are the specific deliverables of the feature teams or the relationship of the different feature teams and development activities to the system architecture itself.

- To reject the processes espoused by methodologists is equally wrong. Methodologists have been able to gain a set of experiences that cover a wide range of situations. They have captured these experiences, as best they can, in their methods. At the very least, reviewing a method can help you think of questions you might want to ask yourself to make certain you aren't forgetting anything that could enhance your own process.

I tend to side with the methodologists. I assume that methodologists are smart and experienced people, and that I can usually get more value simply by trying what they recommend rather than by inventing everything on my own. I'm not claiming that every method is as good as every other method. If I'm not certain whether (according to a particular method) a certain process step should be followed, I will try to follow it. You might say I tend to trust an established method when working in unknown territory.

Suppose you asked me to write an embedded, real-time microprocessor control system for an MC68000-based hardware device. Since I've never designed such a system, I'd head down to a bookstore and see if I could find a book that described a method for building such systems. I'd read that book, think through the process it proposes, and then try to create the models it prescribes.

Others may decide, on the basis of their own experience, that they will only create the models they know they need to create, worrying about other models (or process steps) later. Developers of this type tend to trust themselves. I'll leave it to you to assess the relative advantages and disadvantages of each approach.

A METHOD FOR TEAM ENVIRONMENTS

How do these ideas scale to a team environment, in which most professional software is created? At SmartPatents, we have organized the development group and marketing organization into a set of cohesive feature teams. Each feature team is completely responsible for one or more related product features. Certain horizontal layers provide a core set of "services" to the feature teams.

Feature teams do not solve every problem associated with software development. Indeed, they create some. Consider communication between feature teams. If the first feature team chooses Fusion for documenting classes while the second feature team chooses Booch, the teams can't share diagrams. Sharing diagrams is the primary goal of the Unified Modeling Language (UML), which defines a common way to communicate information about software and system models, while at the same time letting individual teams and organizations choose the most appropriate method of creating and using these models. Because of this advantage, SmartPatents selected UML as the primary means of documenting models.

By giving feature teams as much latitude as possible in defining their own process, we create an environment that, over time, leads to processes developers can trust. But getting to this trustworthy process is itself complex. If you don't already have a process you trust, how will you create one? One way to do it is to find people you trust and to ask them what process or method they use, and then adopt one of their methods as your own. Another way to create a process you trust is to take a method that was generally designed for your kind of problem and critically examine it in the context of your given situation. In this case, you'll want to employ the parts that seem to make sense. If you're not sure, plan to try the process step, method, or model out anyway. You might find the following questions helpful in making your decision:

- Does what the method propose help you understand the problem and visualize its solution?
- Does the method help others understand your understanding of the method and your solution?

Then, put the process you've chosen into action by following the method's steps and preparing the models you've selected. When you're finished, you'll

have gained personal experience in what works and what doesn't work in your environment. More importantly, you'll have gained that much more knowledge about how to make these decisions in the future. Keep track of what works and what doesn't. It is this kind of track record that forms the foundation of trust.

Examining a method and enacting a new process takes the one thing we never have enough of: time. It takes time to examine a method critically and make a conscious decision to do something new. And there are no guarantees that the changes you make will actually improve your ability to generate software, although making conscious decisions about your process should help you trust it. Quite simply, choosing to modify your current process in the hope of generating a better process requires a leap of faith. Of course, if you don't think you have the time to make any changes to your existing process, then you must trust your process enough to leave it alone.

Trust is a quirky—but very important—factor in human relations. We need to learn to trust our co-workers and our processes. Rejecting a method simply because you haven't tried it doesn't strike me as very effective.

One last thing. Read through this article and replace every occurrence of the word "method" with "CASE tool." Get my point? ❖

Luke Hohmann is vice president of engineering at SmartPatents Inc., a provider of analytical software tools for intellectual property management. He especially enjoys his mission of consulting, writing, and speaking about the sociology of software development. His recent book, Journey of a Software Professional: A Sociology of Software Development *(Prentice Hall, 1996), treats aspects of trust in development environments.*

Contact him at lhohman@smartpatents.com

IEEE INTERNET COMPUTING

CALL FOR PAPERS

THE EVOLVING ARCHITECTURE OF THE INTERNET

March/April 1998

Guest Editor: Miroslav Benda, Boeing

miro.benda@boeing.com

This special issue looks at evolving Internet architectures of Web-based applications, mission-critical Web sites, and hardware platforms, as well as the architectures that link these parts together.

Topics include:

- innovative architectures of Web-based information systems
- architectures linking Internet and legacy systems
- e-commerce architectures
- architectures of secure Web sites
- trade-off analyses of architectural alternatives
- trends in architecture (past and present)
- frameworks linking Internet architectures with business process architectures

SUBMISSION DEADLINE: 17 OCTOBER 1997

For submission instructions, see http://computer.org/internet/auguide.htm

Call for Articles

Networking Security

Security strategies for the emerging broadband environment

Submission deadline: January 15, 1998

Publication date: August 1998

As the explosive growth of networking technology continues to redefine the rules for maintaining the privacy and integrity of electronic data, there is a growing concern over security as an endpoint issue. Larger enterprises can typically afford technical experts to configure firewalls and other security devices. Smaller enterprises and homes, however, often do not have the resources to create a level of security suitable for the emerging broadband market. This special issue will focus on the integration of networking and endpoint security.

Guest Editors

Patrick W. Dowd
Univ. Maryland
p.dowd@ieee.org

John McHenry
National Security Agency
jtmchen@afterlife.ncsc.mil

Contact the guest editors in advance of submission.
Electronic submissions only: PostScript, Acrobat, Word, FrameMaker.
See the full call on p. 93 of this issue.

COMPUTER
Innovative technology for computer professionals