I Believe!

Forrest Shull

WHAT DO YOU believe is the best way to develop software? And what does that have to do with anything?

A recent paper in the Industry Experience Track of a software engineering conference got me thinking again along those lines. Carol Passos and her coauthors worked with a team applying Scrum at a medium-sized Brazilian software company. In particular, they examined the way the team practiced Scrum, along with key stakeholders’ beliefs about Scrum’s benefits and drawbacks and related practices. They concluded that all the developers’ beliefs originated from personal hands-on experience, but also that, when the developers applied their beliefs in the current project, “past experiences were taken into account without much consideration for their context.”

In short, this paper presents evidence that we as developers create our own mental models of what seem to be good and bad practices on the basis of our experiences. This isn’t surprising, but rather reassuring. Like any other complicated endeavor, software engineering projects should learn from what has worked (and what hasn’t) in practice to inform future efforts. But the paper also points out the danger that our mental models don’t always get triggered for updates often enough; this should be the opposite of reassuring.

Beliefs—Good or Bad?

What exactly are beliefs, and why is it worth being aware of them? In this context, a positive spin would be to say that a belief is a heuristic to support decision-making, based on hard-won experience or other evidence. But with equal validity, we could also say that a belief is a judgment, perhaps anecdotal in nature, which need not be substantiated by hard numbers or rigorous analysis. In either case, beliefs have a positive effect in that they allow us to reach decisions quickly—there’s simply not enough time in the day to research every decision from scratch, even if the right data/study/experience exists with the needed answer. On the other hand, getting to a decision more quickly is not a benefit if the decision is the wrong one.

If we accept that we software engineers might not always update our set of beliefs often enough, is it such a bad thing that we go through life trying to apply the good practices we’ve learned from a previous project to our current one? Tim Menzies would argue that, based on available data, it can indeed be problematic because beliefs generated from one context tend not to transfer well to new ones. Tim is an expert in data mining. He’s spent years looking in project data for observations that our mental models don’t always get triggered for updates often enough; this should be the opposite of reassuring. It’s important to remember that Passos’ study was a small one, in a very particular context, but I doubt you’d argue against her conclusions being relevant to many circumstances in software development. Even the most reflective and conscientious of us can have outdated beliefs about what works well, given how fast our field is changing.
IEEE SOFTWARE IN CONFERENCE NEWS

This year, IEEE Software partnered with several conferences. We gave best-paper awards at two recent events.

ACSEAC ’11 IN AFRICA

IEEE Software helped sponsor the first of what we hope will be a long series of conferences: the 2011 African Conference on Software Engineering and Applied Computing (ACSEAC ’11). Although our readership is international and quite diverse, Africa remains an area underserved by our activities, so we were happy to be involved at the ambitious beginnings of such an event. More than 40 people, predominantly from African countries, attended. The magazine was represented at the event by a past board member, Andy Bytheway of the Cape Peninsula University of Technology.

Our best-paper award went to “Locating Features in Code by Semantic and Dynamic Analysis” by Wu Ren and Wenyun Zhao of Fudan University, Shanghai. The authors applied information retrieval techniques to program understanding, mixing dynamic analysis with semantic analysis. This combination allows chunks of program functionality to be reverse-engineered from program traces and then related to code segments. Thus maintainers who are not familiar with the software can generate a map of which code artifacts relate to which group of program features in the absence of requirements specifications and documentation.

Second place was awarded to “An Automatic and Intelligent Workflow Management Tool Design” by Cedric Philippe and Soulakshme Nagowah from the University of Mauritius (see Figure A). They designed a project management workflow system that embraces new ideas dealing with decisions and constraints, and (if all goes as expected) will be fully automatic. They are deploying ideas about data mining to ensure that new projects are planned with the best possible understanding of recent history, and that all the evidential data that arises from actual project management is harvested and added to the mix.

We thank the panel members who gave time and energy to the selection process: Laban Bagui, Joseph Balikuddembe, Akhona Damane, Hakan Erdogmus, Bob Glass, Julius Stuller, Willem Visser, Rym Wenkstern, and of course Andy Bytheway for all his work on shepherding the awards process through to completion.

Next year the conference will be in Botswana, continuing to provide an opportunity for the software engineering community in this part of the world to communicate and share ideas.

ESEM ’11 INDUSTRY EXPERIENCE TRACK

IEEE Software also sponsored a best-paper award in the Industry Experience Track of the 2011 Empirical Software Engineering and Measurement (ESEM) Conference. Empirical software engineering focuses on using evidence, both quantitative and qualitative, to examine how well software engineering techniques work in practice. It makes a natural complement to the articles we strive for in IEEE Software—ones based on data and experience.

The Industry Experience track, chaired by Brian Robinson, senior principal scientist at ABB, was a particularly important venue. It encouraged reports describing the outcomes (positive or negative) of evaluating or deploying different technologies, ideas, or methods in an industrial setting, as well as the lessons learned.

Papers were nominated by the program committee. A committee representing the conference and the magazine cast votes for the most well-written, practical, experiential, and rigorous paper. Our winner was “Scrum + Engineering Practices: Experiences of Three Microsoft Teams,” by Laurie Williams of North Carolina State University and Gabe Brown, Adam Meltzer, and Nachiappan Nagappan of Microsoft (see Figure B). As one panelist commented, this case study lets researchers “look under the hood of the Scrum development practice. The results are controversial enough to stir up many discussions and ideas about future research in this area.”

Many thanks to our selection committee: Mark Grechanik, Maurizio Morisio, Brian Robinson, and Helen Sharp.

FIGURE A.

ACSEAC ’11 program cochair Sonia Berman delivers the IEEE Software award to Soulakshme D. Nagowah and Cedric C.C. Philippe. (Photo courtesy of Andrew Bytheway.)

FIGURE B.

IEEE Software editor in chief Forrest Shull (on right) and Brian Robinson (on left), chair of ESEM ’11’s Industry Experience Track, congratulate one of the winning authors, Laurie Williams. (Photo courtesy of Emadoddin Livani.)
that provide insight about the relationships that seemed to hold on that project—for instance, whether the number of defects found in a component was related to the way it was developed or its complexity. Although we can find such relationships within projects, applying them across projects without careful thought can be quite dangerous. Tim points to several recent studies showing that lessons learned from one project are simply not applicable to others.\(^2\)

For example, a study of the prediction factors in the Cocomo model across 93 project datasets from one organization showed huge variations in the size of the effects those factors had on overall project effort. In fact, in some cases, the direction of the relationships changed from positive to negative depending on which projects were in the dataset being fit.\(^2\) Also, consider the results of a study by Tom Zimmermann and his colleagues that looked at defect predictors for pairs of projects.\(^3\)

In only 4 percent of the pairs did the factors that worked well for predicting defects in the first project also apply in the second.

Perhaps it’s only prediction models that are so “fragile” and context-specific? Surely, good practices should be more transferable from one project to another? While some practices are certainly transferable in the right context, this is by no means a universal phenomenon. Attempts to aggregate the available evidence on agile practices like test-driven development (TDD) have also shown wide disparities in how well these practices work in different contexts. A systematic review that aggregated the available evidence about TDD, for instance, found 10 studies in which TDD resulted in higher productivity than otherwise and nine studies in which TDD led to worse productivity (and just to complete the picture, six additional studies found no significant effect on productivity at all).\(^4\)

None of this should come as any real surprise, of course. Saying that any software engineering phenomenon is globally true would be like saying that hardware engineers should use the same practices to develop a toaster as to develop a satellite (to paraphrase one of Vic Basili’s favorite sayings). There is simply too wide a set of things accumulated under “software” for us to find one-size-fits-all solutions. However, research over the last several years shows that when we look for similar contexts from which to draw relevant lessons learned, we don’t always know which are the right factors to use in judging similarity. Tim’s research, in particular, has found that the most useful way is to compare contexts based on whether their performance data suggest that similar relationships hold, rather than using context factors to judge whether contexts are similar \textit{a priori}.

The Take-Away

So what’s the message for development teams? Clearly, the take-away is not to discard experience and hard-earned beliefs as ways to manage new projects. Rather, I think it is to develop a sense of healthy skepticism: “Trust, but verify.” Where needed, use your beliefs to steer projects and make the many day-to-day decisions that are needed to make progress. But also be aware of beliefs’ limitations and use them for three things.

First, generate testable hypotheses and then look for ways to support or refute whether those hypotheses hold on the current project. For example, if I believe that excessive coupling leads to poorly maintainable code, I’ll identify the components with the highest coupling and see whether they’re involved in more changes than other components, or whether the changes take longer to make. If I believe that TDD is going to lead to more productivity, I’ll look at whether features or change requests are getting implemented in a timely way. (If I can quantitatively or qualitatively compare features developed prior to and after TDD, so much the better.) Avoid making this an extra, unfunded activity for the developers by looking at what existing data can be leveraged to answer these questions—for example, use timesheet systems to see where effort is being charged, or project tracking systems to see when issues or changes are opened and how long they take to close.

Second, compare and contrast your beliefs with those of other stakeholders—paying attention to context. Find forums, both formal and informal, to share your experiences about what works and what doesn’t. But make sure
to include in these conversations information about what context the beliefs are drawn from, and use that as a way to avoid the “holy wars” that tend to break out when folks claim that one practice or technology always (or never) works. Be sure to include different stakeholders in such forums—for example, not just developers but managers and assurance personnel—and explore how the different stakeholders’ backgrounds impact their understanding of good and bad practice.

Finally, and above all, reflect on where your beliefs originated. Think back to that context and consider whether there are important ways in which it reflects the current one.

Successfully articulating, testing, and comparing beliefs are key steps toward becoming a reflective practitioner. Doing so lets projects continue to use applicable best practices and evolve the ones that don’t fit so well. Such methods are at the heart of what it means to be agile, and to continuously improve. And by the way, doing so might just generate some useful insights—with evidence—for a truly stellar article in IEEE Software.

**References**


