Engineering Values: From Architecture Games to Agile Requirements

Forrest Shull

“The values to which people cling most stubbornly under inappropriate conditions are those values that were previously the source of their greatest triumphs.” —Jared Diamond, Collapse: How Societies Choose to Fail or Succeed

PERSONAL PRINCIPLES and values are important. Articulating the personal qualities that we prize gives us a way to set goals for ourselves and reflect on whether we’re living up to our own standards—or whether we’re so caught up in the day-to-day grind that we make a thousand little compromises and never notice.

Similarly, professional principles are touchstones that let us aim for rigor in our work rather than spending our efforts on a thousand different trade-offs and kludges. Perhaps such engineering principles are even more important in the early phases of any software life cycle, when the freedom to interpret the problem and design solutions can be almost unbounded.

And yet—when taken to extremes, as was meant to avoid, stifling creativity and leading to bad decisions. Philippe Kruchten brought this point home to me afresh during his recent keynote and the personal interview that followed by enunciating some important values and their potential misuses in software requirements and architecture.

Kruchten’s work has often focused on software architecture, and in his consulting work, he has observed a large number of teams and their dynamics in operation. When I caught up with him at the Better Software East conference, he had just given a keynote summarizing some of the negative behavioral patterns that he has often seen in the field (he referred to them as “games architects play”). I was struck by how many of these initially sounded like positive engineering principles taken to extremes, which seems to be a cautionary tale regarding the potential for rules to get embedded out of context.

For Kruchten, one of the root causes of such games is that, like other hu-
The Pilot Project

Let’s begin with a game Kruchten identified that’s close to my own heart as an empiricist: the pilot project. There’s a solid principle at the heart of this one—namely, a desire to be intentional and cautious about new tools and technologies. A key component of making software engineering an evidence-driven field rather than one at the whim of fads is trying novel approaches in small-scale but realistic settings to see if the claimed benefits can really be obtained in one’s context (and at what price). Surprising as it sounds, I’ve seen organizations go off the rails by mandating widespread adoption of new technologies that turn out to be more complicated in practice than initially realized.

However, the “game” begins when organizations march into pilots with a preconceived idea of the outcome they want to see and eliminate anything that would be an obstacle to getting that predefined outcome—even if those obstacles are, you know, actually related to the way they normally do business.

So how do we monitor ourselves and understand whether confirmation bias is actually coming into play? One heuristic is to count how often we hear, “But we’re not doing [something] on this project because this is just a pilot.” Has “just a pilot” become the mantra to the way they normally do business.

The Pilot Project

Let’s begin with a game Kruchten identified that’s close to my own heart as an empiricist: the pilot project. There’s a solid principle at the heart of this one—namely, a desire to be intentional and cautious about new tools and technologies. A key component of making software engineering an evidence-driven field rather than one at the whim of fads is trying novel approaches in small-scale but realistic settings to see if the claimed benefits can really be obtained in one’s context (and at what price). Surprising as it sounds, I’ve seen organizations go off the rails by mandating widespread adoption of new technologies that turn out to be more complicated in practice than initially realized.

However, the “game” begins when organizations march into pilots with a preconceived idea of the outcome they want to see and eliminate anything that would be an obstacle to getting that predefined outcome—even if those obstacles are, you know, actually related to the way they normally do business.

So how do we monitor ourselves and understand whether confirmation bias is actually coming into play? One heuristic is to count how often we hear, “But we’re not doing [something] on this project because this is just a pilot.” Has “just a pilot” become the mantra to the way they normally do business.
Given that architects typically make decisions quickly, based on experience and in-intuition and can make decisions that we have sufficient experience and “blink”—in which we tell ourselves another example Kruchten gave is another detailed solution element?

ACSEAC 2013 will be held in Nairobi, Kenya.

we’ve hand-picked a team that’s exceptionally good and might have obtained a successful outcome regardless of the technology or process they apply.

Blink

Another example Kruchten gave is “blink”—in which we tell ourselves that we have sufficient experience and know-how and can make decisions quickly, based on experience and intuition. Given that architects typically face a wide range of potential solutions for any system, being able to quickly make assessments about good strategies is important. After all, one recent best-seller has argued that such intuition is an important attribute of brilliant decision makers (M. Gladwell, Blink: The Power of Thinking without Thinking, Back Bay, 2007).

When done by architects who truly have the experience, such quick and intuitive assessments can be valuable in focusing efforts on the path with the most likely chance of success. The difficulty comes when cognitive biases come into play and lead to this ideal being applied in circumstances in which it has no business.

How do we recognize when we’re stumbling into the negative “blink” game? Perhaps by paying attention to how much the architects are really listening to the requirements folks. Do requirements analysts get to articulate what they’re looking to do with the system before they start hearing about pull-down menus, Apache servers, and other detailed solution elements?

Analysis Paralysis

Some of Kruchten’s games spring not from a lack of rationality but from being too rational. For example, the opposite of “blink” is over-focusing on analysis to the detriment of actually getting things done. As with the other games, it’s hard to argue with the desire to do a thorough and rigorous analysis.

But too often, we might simply be afraid of making decisions, so continued analysis becomes a convenient decision-avoidance strategy. For architects, this game can also take the form of a million questions and requests for clarification to the requirements team. While it can always be healthy to challenge requirements, extreme forms of this effectively push off any responsibility for engineering decisions to other stakeholders. In either case, while you’ve been trying to know everything about everything, you’re potentially blocking progress on other fronts as well.

Kruchten’s response to how to mitigate this tendency is to come to terms with the fact that we can’t make perfect decisions. On almost any development project, we have to make progress, even if it’s imperfect. Rather than aiming for perfection, it’s often healthier instead to aim to have some kind of traceback that lets the team recover if it reaches a bad state.
A Healthy Perspective
What if I have a plausible and rational-sounding story that reassures me that I’m holding fast to my values when I’m actually applying them inappropriately? Kruchten recommended not aiming to avoid cognitive biases entirely (since we can’t easily change the result of millions of years of human evolution) but rather to have someone on the team who thinks differently—that is, someone who doesn’t have the same biases—who can challenge the status quo and help us see what we’ve missed.

Lest this all start to seem a little abstract, I also had the chance to talk with someone who’s been putting similar approaches into practice: Ellen Gottesdiener is an internationally recognized expert in agile practices. In working with various organizations, Gottesdiener and her colleague Mary Gorman have developed an approach to requirements management in an agile context that is proving useful (www.discovertodeliver.com). As we talked, I realized that Gottesdiener was presenting an approach that, serendipitously, seemed designed to help avoid the traps and pitfalls that Kruchten had presented.

The agile teams that I’ve been part of had typically represented requirements by means of user stories. Gottesdiener noted that teams often start at this level because it helps them make some quick progress, but there are limits. In fact, she’s found that teams benefit from looking more holistically at requirements and advocates thinking about the potential options for the system across seven dimensions (users involved, interfaces to other entities, actions or capabilities provided, data used by the system, controls enforced, environmental constraints, and quality attributes). Teams explore options along these dimensions and evaluate and reevaluate them. They sketch plans for implementing options according to three time horizons:

• generalized ideas of what the product is evolving into;
• more specific options to be included in the next release; and
• detailed descriptions of the work allocated to the current iteration.

It’s important to note that all three levels of planning aren’t created by doing a lot of time-consuming, deep thinking upfront. But at least the team can begin development having identified some big chunks of functionality that can be fleshed out into stories as work continues (aiming to avoid analysis paralysis).

The methodology also encourages the team to think about what value is being provided by work across all three time horizons—and what value is being provided to different stakeholders. The point of identifying all of these dimensions and time horizons isn’t that this is the magic analysis that will reveal all important factors. Rather, this set of terminology and attributes provides the means for having a structured conversation among the various stakeholders, who likely have different backgrounds and possibly even vocabularies. (In short, the structure is a support mechanism for implementing Kruchten’s advice of getting input from different perspectives which can help avoid falling into biases—it ensures that all stakeholders are actually talking about the same things and eliciting relevant information.)

The important thing to realize is that as always with agile, we’re looking for ways to get quick feedback. It’s important to informally value the stories and periodically revisit those decisions—but we need to avoid long and expensive analyses. As Kruchten noted, it’s not realistic to aim for making perfect decisions. This may mean that rework later becomes necessary as we...
get a better understanding of what is needed. But nothing is free; the cost of rework needs to be evaluated against the benefit of having gotten a better idea of customer needs and tested the system's ability to meet them.

Essentially, what Gottesdiener presented was a software “process” in the best sense of the term: an approach that can be tested (and, dare I say, piloted) as to the value it provides, and one that advocates specific steps not for their own sakes, but as guideposts that guard against us falling into the games and cognitive biases that we can recognize in any human endeavor.

As always, my conversations with Philippe and Ellen contained more details and fascinating anecdotes than I could include in this article. If you are interested in this subject matter, you can find the full interviews at https://www.computer.org/software-multimedia/march-april-2013.

FORREST SHULL is a division director at the Fraunhofer Center for Experimental Software Engineering in Maryland, a nonprofit research and tech transfer organization, where he leads the Measurement and Knowledge Management Division. He’s also an adjunct professor at the University of Maryland College Park and editor in chief of IEEE Software. Contact him at fshull@computer.org.

IEEE SOFTWARE CALL FOR PAPERS

Special Issue on Architecture Sustainability

Submission deadline: April 1, 2013 • Publication: November/December 2013

Change is the norm in software, and architects must address and embrace it to ensure the architecture can endure different types of change via efficient maintenance and evolution throughout its entire life cycle. Some changes might only affect local parts of a system, while others have more systemic impact that can involve various stakeholders and disciplines.

In general, architects can employ three approaches to handle change: refactoring, renovating, and rearchitecting. Such strategies contribute to the overall area of architecture sustainability—but have yet to be treated as a coherent whole. The aim of this special issue is to encourage such discussions.

Possible topics include but are not limited to:
• State of the art and practice in architecture sustainability
• Recognizing architectural sustainability problems early by architecture introspection (performed by the architects themselves) and assessment (performed by a third party)
• Fundamental elements of architecture refactoring, rearchitecting and renovating, how they complement each other, and strategies for recognizing when they are appropriate
• Ensuring architecture sustainability through architecture governance
• Processes, tools, and technologies to support refactoring, renovating, rearchitecting, and their governance
• Managing traceability between requirements and architecture decisions that lead to architecture success or problems
• The effect of architectural drift or erosion on architectural sustainability and potential countermeasures

Questions?
For more information about the focus, contact the guest editors:
• Paris Avgeriou, University of Groningen, The Netherlands, paris@cs.rug.nl
• Michael Stal, Siemens Corporate Technology, Germany, michael.stal@siemens.com
• Rich Hilliard, software systems architect, USA, r.hilliard@computer.org

Submission Guidelines
Manuscripts must not exceed 4,700 words including figures and tables, which count for 200 words each. Submissions in excess of these limits may be rejected without refereeing. The articles we deem within the theme and scope will be peer-reviewed and are subject to editing for magazine style, clarity, organization, and space. We reserve the right to edit the title of all submissions. Be sure to include the name of the theme or special issue for which you are submitting.

Articles should have a practical orientation and be written in a style accessible to practitioners. Overly complex, purely research-oriented or theoretical treatments are not appropriate. Articles should be novel. IEEE Software does not republish material published previously in other venues, including other periodicals and formal conference/workshop proceedings, whether previous publication was in print or electronic form.

Full author guidelines: https://www.computer.org/software/author.htm
Submission details: software@computer.org
Submit an article: https://mc.manuscriptcentral.com/sw-cs