Editor: **Eoin Woods**
Artechra
eoin.woods@artechra.com

Driving Agile Architecting with Cost and Risk

Eltjo R. Poort

A LOT HAS BEEN SAID and written about the relationship between agile development and architecture. Roughly four years ago, *IEEE Software* even featured a special issue on it ("Agility and Architecture: Can They Coexist?" *IEEE Software*, vol. 27, no. 2, 2010). Now it looks like the debate is starting to settle down: we see agile methods that include architecting, such as the Scaled Agile Framework, and we see architecting frameworks such as TOGAF (the Open Group Architecture Framework) adding agile elements. As the dust settles, what have we learned? At CGI—a global IT and business process services provider—architects have learned to use economic drivers such as risk and cost to enhance their work's agility.

Five pieces of advice can help architects become more effective in an agile world without having to implement new methods or frameworks. They describe changes in attitude or behavior rather than complete practices or principles, so they're easy to digest and apply. The ideas are based on a solution architecting approach called Risk- and Cost-Driven Architecture (see the sidebar). Core to the approach is the use of risk and cost to determine the architectural significance of concerns. Agility is achieved by keeping the architecture lightweight, addressing only those concerns that are especially risky or costly. A risk- and cost-driven backlog of architectural concerns balances the generally value-driven product backlog to achieve "just enough anticipation" in the evolution of software solutions.

Decisions Are Your Main Deliverable

One of the criticisms of architecture from the agile community is based on the misconception that an architect's purpose in life is to deliver "an architecture," commonly interpreted as a piece of documentation—which, according to the Agile Manifesto (http://agilemanifesto.org), is valued less than working software. This is a poor representation of what real architects do every day: they look for architectural concerns to address, figure out the options they have for addressing those concerns, and then decide the best course of action given their current context (the three circles in Figure 1). Looking at it this way, the architect's main deliverable isn't a document but a stream of decisions.[1]

This way of looking at architecture work is perfectly compatible with the agile mindset, regardless of whether these decisions emerge from early implementation and refactoring, from careful upfront modeling, or from a combination of both. In agile projects, decisions often

 0740-7459/14/$31.00 © 2014 IEEE

emerge from a group process with shared ownership, but even then it makes sense to look to the architect as the person who safeguards the overall design's conceptual integrity. Thus, the role of the architect is to make sure that the group's decisions are consistent across the whole solution, even when multiple teams are working on it simultaneously.

Keep a Backlog of Architectural Concerns

Architects working in an agile environment don't have a preapproved, fixed set of specifications they can base their designs on. Even architects working in a traditional waterfall context can't rely on the environment to remain fixed because they're expected to design a future-proof solution that anticipates and can survive a certain amount of change.

One of the tools used in the agile world to embrace change is a product backlog: an ordered list of requirements waiting to be implemented. A backlog can easily be reordered to accommodate changes in requirements or the way they're valued over time—making it easier to embrace change than if an extensive plan had been drawn up. As Figure 1 shows, the architect's backlog consists of the architectural concerns to be addressed because they determine her workload. By frequently reassessing the priority of concerns in the backlog, the architect becomes more flexible in dealing with new business requirements and emerging insights.

Let Economic Impact Determine Your Focus

So how do we prioritize the concerns in our backlog? Which should be addressed first? A decade ago, Martin Fowler wrote that "architecture is about the important stuff—whatever that is."[2] We've found that considering economic impact helps us determine what's important. In other words, we've found that an estimate of a concern's likely economic impact is a good indicator to assess its architectural significance. When the concerns are about what to build, business value is crucial. Many architects, however, spend most of their time worrying about how to build it—in which case, risk and cost are key. When you use risk and cost to determine the focus of your attention, you'll not only ensure your economic impact on the project, but you'll also be able to easily explain your priorities to business stakeholders in terms that they'll understand.

Using economic impact as an indicator of architectural significance also eliminates the fallacy that architecture should only be about concerns

RISK- AND COST-DRIVEN ARCHITECTURE

Risk- and Cost-Driven Architecture (RCDA) is an approach developed at CGI. It was originally intended for internal use by architects shaping solutions to tight deadlines in bids and contracts. Due to its agility, scalability, and wide range of applicable solutions, RCDA has gathered wider interest and is now used by some of CGI's clients. RCDA was validated by published, peer-reviewed research,[1] and is a recognized architecting method in the Open Group Certified Architect program (OpenCA).

Reference

1. E.R. Poort and H. van Vliet, "RCDA: Architecting as a Risk- and Cost Management Discipline," *J. Systems and Software*, vol. 85, no. 9, 2012, pp. 1995–2013.

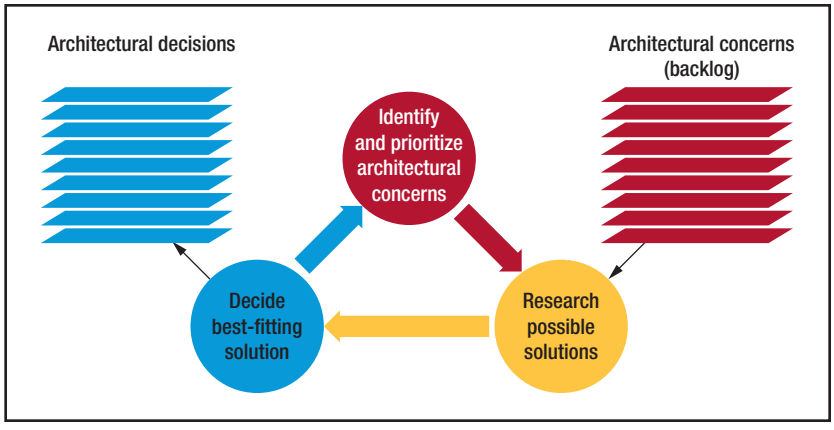

FIGURE 1. The architect's daily job: an "architecting microcycle." Architects watch out for architectural concerns to address, figure out the options they have for addressing those concerns, and then decide the best course of action given their current context.

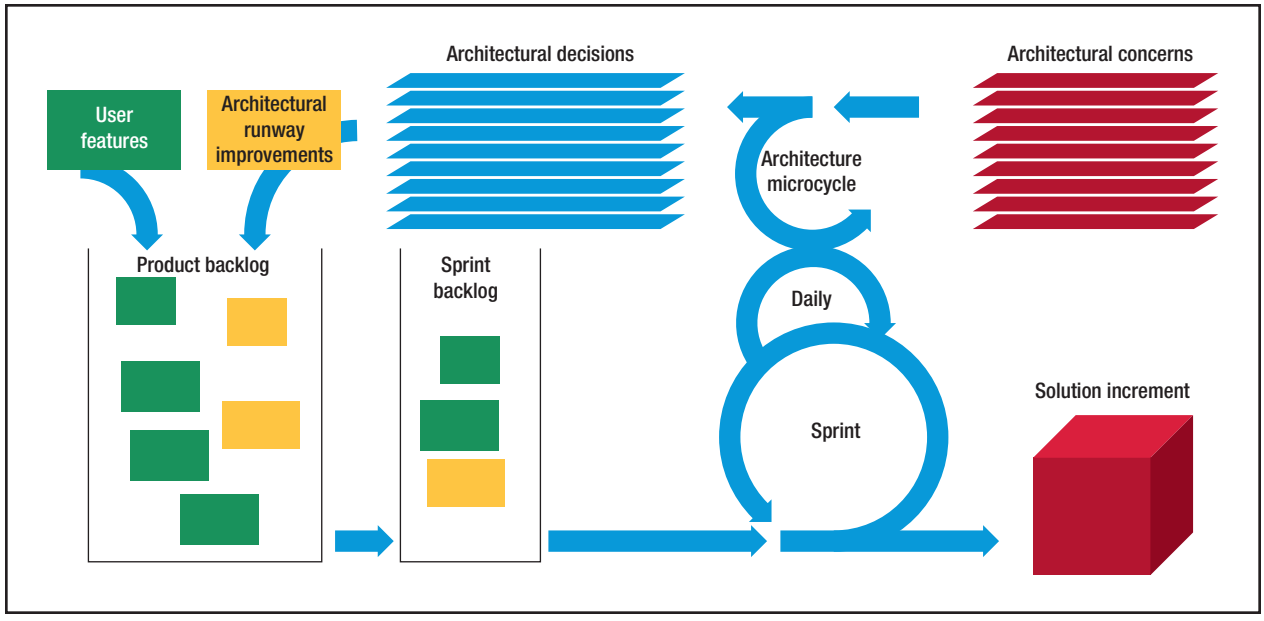

FIGURE 2. Scrum context: aligning the architecture microcycle with the daily stand-up meeting to facilitate collective architectural decision making.

at a high level of abstraction and not about details. Sometimes, the devil is in the details, and some low-level design decisions can be very risky—they should be on the architect's radar, or even coded by the architect.

Keep It Small

Even in large projects, there are two very good reasons to stick to minimal architecture:

- Architecture is hard to change, and the bigger the architecture, the harder it is to tweak. Having too many architectural decisions becomes ballast as conditions change, making it difficult to respond quickly; they can also unnecessarily restrict the design space for the individual development teams working within the architecture.
- An architect can only safeguard conceptual integrity if she can understand the architecture in its entirety. Too many details, and the architect risks losing the overview required to maintain consistency across a complex solution.

In most plan-driven projects, you can identify an architecture milestone because it's the moment of committing to the architecture: after this milestone, reversing key architectural decisions becomes very costly and time-consuming. The optimal amount of "up-front" architecting to be done before this milestone is best determined by careful consideration of three factors, namely, size, criticality, and volatility,[3] rather than through dogmatic slogans like, "You ain't gonna need it." Bigger, more complex solutions and solutions that are more business critical require more up-front architecting; in a more volatile environment, less up-front architecting is better. Many agile projects, however, have no architecture milestone and need a different way to determine how small they should keep the architecture.

Just Enough Anticipation

How do architects in agile projects determine the right amount of architecture? According to the first piece of advice above, the architecture is a flow of architectural decisions. This flow should be ahead of solution development and delivery with just enough anticipation.

The tools at your disposal to determine the right amount of anticipation are dependency analysis, technical debt control, and economic consideration of future options[4]:

- Use *dependency analysis* to determine which architectural components are needed to realize anticipated user stories.
- Use *technical debt control* to prevent the solution from deteriorating when too many user fea-

tures are added without taking the time for refactoring. Identify architectural debt and plan its resolution in time. Here, we aren't talking about implementation debt that can be measured by code analysis tools. With architectural debt, we mean structural imperfections and technological gaps that hinder agility and can petrify the whole solution if left unaddressed.
- Weigh your options by careful *economic consideration*. Models like Net Present Value can yield objective insight into the right timing for implementing architectural decisions. Use these techniques to discuss the right amount of anticipation with greedy product managers, worried operational staff, and other stakeholders. Economic predictions, however imprecise they sometimes are, still tend to be more convincing than references to agile dogmas, generic design principles or gut feelings.

The Scaled Agile Framework (http://scaledagileframework.com) uses the metaphor of a runway that's continuously being extended while in operation, so that it's always just long enough to accommodate the new planes that are anticipated (the planes in the metaphor are upcoming solution requirements). The new, bigger planes can only land after the runway is extended for them: dependency analysis determines which runway extensions are required to land which planes. Sometimes you can temporarily extend the runway with an inferior material for the sake of speed: this represents technical debt that will have to be repaid (repaved) at some point to prevent accidents. All decisions (when to extend or repave the runway) should be based on sound economic reasoning.

That's it: five pieces of advice for agile architecting. Decisions are your main deliverable. The decisions address concerns that you keep in a backlog, prioritized by economic impact: risk and cost. These decisions result in a minimal architecture, with just enough anticipation. There's a lot more to say on agile architecting, pertaining to topics like project organization (is the architect a member of the development team?) and development process (how do you achieve short feedback loops?). These five pieces of advice are limited to what you should be able to apply in any organization or process.

The advice can be applied in agile project methodologies like Scrum (see Figure 2), where the architecting microcycle of Figure 1 is used to add architecture runway improvements to the product backlog. These architectural improvements complement user features to create a balance of anticipation in the product backlog. But the advice is equally applicable to architects working in plan-driven projects, who also often have to time-box their work dealing with changes and emerging insights. Regardless of the project methodology, architects have to make sure they address the most significant architectural concerns, where risk and cost prove to be good indicators of architectural significance.

References

1. J. Tyree and A. Akerman, "Architecture Decisions: Demystifying Architecture," *IEEE Software*, vol. 22, no. 2, 2005, pp. 19–27.
2. M. Fowler, "Who Needs an Architect?," *IEEE Software*, vol. 20, no. 6, 2003, pp. 11–13.
3. B. Boehm, "Architecting: How Much and When?," *Making Software: What Really Works, and Why We Believe It*, A. Oram and G Wilson, eds., O'Reilly Media, 2010, pp. 141–186.
4. N. Brown, R.L. Nord, and I. Ozkaya, "Enabling Agility Through Architecture," *CrossTalk*, vol. 23, no. 6, 2010, pp. 12–17.

ELTJO R. POORT is a solution architect at CGI. Contact him at eltjo.poort@gmail.com.