Source: https://www.infoq.com/news/2016/09/agile-architecture-friends

How Agile and Architecture Parted and Finally Became Friends

by **Ben Linders** on Sep 13, 2016. Estimated reading time: 4 minutes

People stopped seeing the need to define the architecture or do software design due to incorrect interpretation of the agile manifesto, argued Simon Brown, author of Software Architecture for Developers. Many software developers don't seem to have a sufficient toolbox of practices and the software industry lacks a common vocabulary for software architecture. A good architecture enables agility; you need just enough up front design to create firm foundations for setting the direction.
Brown gave the opening keynote about how agile and architecture parted ways and finally became friends at SwanseaCon 2016, the second agile development and software craftsmanship conference for Software Developers, Software Architects, Project Managers, Analysts and Consultants in South Wales. InfoQ is covering the conference with Q&As, summaries, and articles.
The idea behind waterfall is to optimize the stuff you can learn early. Time spent early in development can reduce costs at later stages, said Brown. As an example, he mentioned the Structured Systems Analysis & Design Method (SSADM), which is a waterfall based approach for developing software. It provides an end-to-end life cycle using concepts from systems management, which helps you think about activities at different stages for creating a software solution. Brown also mentioned the Rational Unified Process (RUP), an iterative and incremental framework. RUP should be customized but nobody did that, said Brown, resulting in processes that were too big.

The main problem with waterfall is the long feedback loop. The structure and rigor from waterfall methods helps to develop a high quality product, but without feedback there's a risk of developing the wrong product.

The agile manifesto states that processes and tools are still important with agile, but individuals and interaction are valued higher. Many however interpreted the agile manifesto incorrect by assuming that processes aren't needed anymore. The manifesto also states "working software over comprehensive documentation" which has led to people not seeing the need to define the architecture or do software design. This lead to conflicts between agile and architecture, said Brown.

The first conflict is about the team structure where the question is, if we need a dedicated software architect, or if everybody is a software architect? Principle 11 of the manifesto states that "the best architectures, requirements, and designs emerge from self-organizing teams". The good news is that the manifesto mentions architecture and design, said Brown. He has seen successful teams where the architecture role was spread out, but also teams where then was no responsibility for

architecture and design; where everybody assumed that someone else was taking care of the architecture.

The second conflict has to do with the process. Historically, there's been a tendency towards big design up front (BDUF), argued Brown, where people tried to understand everything up front to define a blueprint. People questioned if it was allowed to do up front design with agile. Evolutionary design tried to provide a solution by doing some design. But it is hard to change software when the architecture turns out to be wrong; refactoring can be costly. Core foundation blocks tend to work best if built in from the start, said Brown.

Brown disagrees that you don't need architecture when doing Test Driven development (TDD). He suggested to up front define the architecture so that TDD can work within boundaries. Also, Brown strongly opposes to taking architecture decisions at the "last responsible moment", as this can and is interpreted as "lets not make any decisions ever".

In order to fix the problems with architecture we need to understand what agile really is, said Brown. The core definition he proposed is:

> Moving fast, embracing change, releasing often, getting feedback.

Agile is a lightweight approach to deliver software, based on a mindset and culture of continuous improvement. Being- rather than doing agile- is what matters, said Brown. But the wording in the agile manifesto unfortunately confuses people; the "x over y" is often misinterpreted.

Principle 9 of the manifesto states that "continuous attention to technical excellence and good design enhances agility". A good architecture enables agility, argued Brown. According to him, agility is a "non functional" or "quality" requirement. With agile you need to decide between how fast you need to move versus the required quality of the software.

Brown questions if there is a design revival, since methods like DSDM Atern, Disciplined Agile Delivery (DAD), and the Scaled Agile Framework (SAFe) all have design elements in them. He stated:

> It's not about creating a perfect end-state, framework or complete architecture. You need a starting point for the team and the thing that you are building so that you can go in the right way together as a team.

Lean and agile both aim to add value and eliminate waste. Defining a starting point is valuable; you need just enough up front design to create firm foundations for setting the direction.

Brown said that software craftsmanship as defined on Wikipedia is too code-focused. Many software developers don't seem to have a sufficient toolbox of practices. There are lots of ways to document software but people often aren't aware of them. If you ask them how they design software, they will say things like "we use a whiteboard" and "we draw boxes that represent components". His experience is that many people don't know how to decompose and which criteria to use for it, and for instance haven't heard of Class-Responsibility- Collaboration (CRC).

Brown suggested that "moving fast in the same direction requires good communication". The software industry lacks a common vocabulary for software architecture. Software development should be considered an engineering discipline. He mentioned the talk progress toward an engineering discipline of software from Mary Shaw, covered in the InfoQ write-up Software – Is it "engineering" yet. Shaw sums up what is needed for software development to become engineering:

> We are in some respects an engineering discipline, but we cannot yet consistently achieve a level of practice that assures computing systems of a quality that satisfies the social contract associated with engineering. We need to continue to infuse scientific and codified knowledge into design and analysis of software.

Although agile and architecture may have parted ways, they are finally friends again after 15 years, said Brown. He stated, "let’s learn from past experience rather than ignoring it".