Architecture Spikes

Erik Philippus
Improvemen**T** BV
erik@agile-architecting.com
july 2009

A serious pitfall may lurk when you are using scrum on a large scale for the realization of complicated software-intensive systems. The holy grail of scrum is the delivery of working software within a short time frame. This exclusive focus on functionality could easily turn the project into a 'Feature Factory', with the risk of accumulating architectural deficiencies and technical debt that has an impact on performance of the software and overall product quality. This risk can be mitigated by incorporation of a sound architectural basis for the system under development. Although the enthusiasm for architecture has dampened considerably in most agile circles, I believe that a pragmatic combination of agile and architecture deserves some serious investigation.

The attempt to define the architecture completely before implementation begins, has put many projects on the wrong track. The Big Design Up Front (BDUF) typically results in an ivory tower architecture that in most cases proves brittle in practice. The underlying serial mindset is part of a legacy thought process which we better leave behind. Moreover, when architecture is overkill for what actually is required, development teams will have the tendency to move forward on their own instead of waiting for the architects to finish their work.

So – what is the agile community offering as an alternative? A more or less accepted practice within the agile community is the *architecture spike*. Let's first give a description of the *spike* in agile development. A spike is an experiment that allows developers to learn just enough about the unknown elements in a user story, e.g. a new technology, to be able to estimate that user story. Often, a spike is a quick and dirty implementation, a prototype which will be thrown away. So, when a user story on the product backlog contains unknown elements that seriously hamper a usable estimation, the item should be split into a spike to investigate these elements plus a user story to develop the functionality. This enables the product owner or customer to prioritise the research separate from the implementation of the new functionality. Facing a spike and the associated user story, the product owner should prioritise the spike ahead of the user story to obtain a more reliable estimate for the realization of the user story.

As a rule, the spike will be time-boxed. It sounds a bit like a contradiction to prescribe 'a time-boxed discovery'. But to explicitly address the research part of a user story avoids 'creeping' in the discovery process, so time capping the spike will help to keep the project on schedule. A small piece of advice: don't exaggerate. Don't try to control 'the unknown' by deploying spikes in an attempt to eliminate every perceived uncertainty in user stories on the product backlog. How many spikes you throw into the fray: uncertainty will remain a fact of life.

The concept of the spike is often used in conjunction with an architectural issue. Sometimes the architecture spike is referred to as *Sprint 0*, since the goal is to map out enough future bones to get going and to start working on creating the product backlog. This corresponds with the exploration phase in an XP project, encompassing the tentative user stories and initial architectural modeling. To a certain extend, this activity refers to a painter making preliminary studies of (parts of) the final picture. Along the same line, the architecture spike could be described as (part of) a reference architecture[1].

The idea of the first architecture spike is that whilst setting the initial skeleton and creating a single vision of how it might hang together, not much is set in stone. The goal of this architecture spike is to rough out subsystem boundaries, transition points between software layers, (physical) constraints, etc. Subsystems and other details are typically faked with stubs, although is sometimes desirable to have enough of the software be real in order to actually do some preliminary performance and capacity planning assessment. The architecture spike will often demonstrate that some parts of the architecture are not going to work: illogical partitioning, state held in the wrong place, unbalanced data transfer, wrong footprint, etc. This type of architecture spike has been compared with sight reading in music[2].

I can't get away from the impression that this approach corresponds with a 'Small Design Up Front', validated with experimental code. The problem I have is that it seems to ignore *modeling* as a means to validate the architecture. In my view, it is a mistake to assume you always need working code for a proof of concept. Of course models have their restrictions and pitfalls, but I think it is nonsense to put modeling aside as being 'non-agile'. As Scott Ambler illustrates with his concept of Agile Modeling[3], there is no need to turn back the clock.

But what I think is a much more serious drawback of this type of architecture spike, is the absence of an overall vision on crucial product qualities, such as extendability and maintainability. As I hope we have learned by now, these quality requirements must be build into the system right from the start – moreover, they must be enforced by the architecture. Especially when the software we're going to deliver is part of a portfolio of products, it is vital to translate the product roadmap into a sound architectural foundation to guarantee the required product quality attributes for all members of the product family.

It is of course a laudable goal to focus on working software, delivering value-adding features to end-users. But we tend to forget that in general, the majority of the total cost of software projects comes after initial fielding the system. In most cases, these costs can be rooted back to missing quality attributes, and not to missing features! When the non-functional qualities come at the bottom of the list, the total cost of ownership will rise sharply, while business opportunities are missed. It is a fact that the majority of today's industrial software systems confront their owners and users with costly maintenance or legacy issues. So in my opinion, the pressing question that should be answered during the initial architecture spike(s) is: "what are in the longer term the true value-adding aspects of the software system, apart from the tangible and potentially shippable features?"

As far as I know, the only agile method that makes an attempt to bake quality into the life cycle, is the Agile Architecture Method as proposed by J.D. Meier[4]. Essentially, this method helps to identify key engineering decisions or *hot spots* against the prioritized user stories during each iteration. The main hot spots are not only cross-cutting concerns, such as data access, exception management, logging … etc., but also quality attributes, such as security, performance, reliability … etc.

The Agile Architecture Method offers a structured approach for the creation of candidate architecture(s), identification of relevant spikes, identification of deployment constraints and guidance for inspections throughout the life cycle. I believe that the explicit attention for cross cutting concerns and quality requirements is a strong element of this method, since these are the areas in which high impact mistakes are often made. I find it therefore quite confusing that 'Just in Time Architecture' is presented as a way to find intersections between stories and quality attributes during iterations. In my view, it is impossible to address critical quality attributes 'Just in Time': vital product quality requirements must be satisfied on system level, and by definition they transcend individual modules and iterations.

I have the impression that the Agile Architecture Method doesn't make a distinction between architecture and design. Perhaps that's the reason for the flirtation with the unfortunate phrase 'Just in Time Architecture'. I believe that it is realistic and cost-effective to define architecting as an evolving process, which will flourish by an iterative approach. However, I do think that the essential product quality attributes must be agreed upon upfront, together with their domain-specific definitions, measuring approach and acceptance criteria. And of course the architecture spike (or Spint 0) is a good candidate to nail down these overall quality requirements. Then it is the responsibility of the architect that quality levels appear in the product backlog as acceptance criteria for selected user stories. This approach offers the opportunity to explicitly monitor the required quality attributes further downstream, and to take corrective measures 'Just in Time'.

Using the architecture spike this way, customers are helped to make business value decisions, while the risk of degrading product quality during implementation is minimized, promoting delivery of real customer value.

References:

[1] Spike Code vs. Reference Architecture - David Lambert
http://blog.componentoriented.com/?p=373

[2] Sight reading your architecture - Alex Miller's technical blog
http://tech.puredanger.com/2007/04/25/sight-reading/

[3] Agile Modeling – Scott Ambler
http://www.agilemodeling.com/

[4] Agile Architecture Method - J.D. Meier
http://shapingsoftware.com/2009/03/02/agile-architecture-method/