Software Architecture and Agile Software Development—A Clash of Two Cultures?

Philippe Kruchten
Electrical and Computer Engineering
University of British Columbia
Vancouver, BC, Canada
+1 (604) 827-5654
pbk@ece.ubc.ca

ABSTRACT
Software architecture is taking a bad rap with the agilists—proponents of agile and lean software development approaches: "BUFD big up-front design", "YAGNI You Ain't Gonna Need It", "massive documentation", "smells of waterfall", it is pictured as a typical non-agile practice. However, certain classes of system, ignoring architectural issues too long "hit a wall" and collapse by lack of an architectural focus. 'Agile architecture': a paradox, an oxymoron, two totally incompatible approaches? In this tutorial, we examine the real issues at stake, beyond the rhetoric and posturing, and show that the two cultures can coexist and support each other, where appropriate. We define heuristics to scope how much architecture a project really needs, to assign actual value to an otherwise invisible architecture; and we review management and development practices that do work in the circumstances where some significant architectural effort is needed, when you are actually going to need it.

Categories and Subject Descriptors
D.2.11 [**Software Engineering**]: Software Architecture. D.2.9 [**Software Engineering**] Management – *Software Process Models*. K.6.1 [**Management of computing and information systems**] Project and People Management. K.6.3 [**Management of computing and information systems**] Software management – *Software Process*

General Terms
Management, Design, Human Factors.

Keywords
Agile process, software architecture.

1. TUTORIAL OBJECTIVES
The overall objective of the tutorial is to demonstrate that there is actually very little conflict, other than misunderstandings, between the use of an agile software development method (XP[1], Scrum [2], Lean [3], Agile RUP, DSDM [4]), and a healthy, early focus on architectural design [5, 6]. Using the conceptual model from ethnosociology of *culture* and intercultural conflicts, we provide the attendees with tools they can use to deal with this tension – agility ←→ architecture – in their own projects.

ICSE '10, May 2-8, 2010, Cape Town, South Africa

2. TUTORIAL CONTENT
Agilistas world-wide continue to perceive and describe *software architecture* a something of the past, or even 'evil'; they equate it with *Big Up-Front Design* (BUFD) (i.e., a bad thing), leading to massive amount of documentation, implementing features YAGNI (You Ain't Gonna Need It). They see a low value of architectural design, assert that a metaphor should suffice in most cases and the architecture should emerge gradually sprint after sprint, as a result of succession of small refactoring. Conversely, in places where architectural practices are well developed, agile practices are often seen as amateurish, unproven, limited to very small web-based socio-technical systems.

The tension seems to lie on the axis of *adaptation versus anticipation*, where agile methods want to be resolutely adaptive: deciding at the "last responsible moment" or when changes occur, and they perceive software architecture to be pushing too much on the anticipation side: planning too much in advance. And where architects may be tempted to anticipate too much, too early.

Spencer-Oatey defines culture as "a fuzzy set of attitudes, beliefs, behavioral norms, and basic assumptions and values that are shared by a group of people, and that influence each member's behaviour and his/her interpretations of the 'meaning' of other people's behaviour" [7]. Agility is more a culture than an engineering method and probably to a lesser extent is the less visible community of software architecture [8]. They both have developed, based on specific beliefs, a set of values ('agile manifesto' [9]), and behaviors (methods, practices, artifacts), sometimes also rituals and jargon that tend to detract outsiders. As for any clash of two cultures, the members of one culture go through several phases:

- *Ethnocentrism*: my culture is right, is the reference by which other cultures must be judged, others are bad, etc. The attitude is defensive, dismissive, and coercive.
- *Ethnorelativism*: leading to acceptance of other viewpoints, values, and possibly attempts to coexistence and integration. It may guide the definition of new behaviors, and from this new values and possibly evolution of the beliefs.

I personally stand with one foot in each of these two cultures: architecture and agility, and like other ethnic cultures, I see two parties not really understanding the real issues at hand, stopping at a very shallow, caricatural view of the "other culture", not

understanding enough of the surroundings, beliefs, values of the other one, and stopping very quickly at judging behaviors.

The issues of trying to understand the apparent conflict and reconcile the two sides are in multiple dimensions:

1) **Semantics**: What do we mean in this project or organization by "architecture"? The concept of architecture often has fuzzy boundaries. In particular, not all design is architecture. Agreeing on a definition is a useful exercise, and a good starting point.

2) **Scope**: How much architectural activity will you actually need? Most software development projects have a de facto, implicit architecture when they start; they will not need much of an architectural effort.

3) **Lifecycle**: When in the lifecycle should we focus on architecture? Well, early enough, as "architecture encompasses the set of significant decisions about the structure and behaviour of the system" [10]: these are the decisions that will be the hardest to undo, change, refactor. Which does not mean an only focus on architecture, but interleaving architecture "stories" and functional "stories" in early iterations.

4) **Role**: Who are the architects? On large, challenging, novel system, you may need a good mix of experience, of "architectus reloadus"–maker and keeper of big decisions, focusing on external coordination– and "architectus oryzus"–mentor, prototyper, troubleshooter, more code-facing and focused on internal coordination– to follow Martin Fowler's metaphor [11].

5) **Documentation**: How much of an explicit description of the architecture is needed? While in most cases, an architectural prototype, starting with a walking skeleton, for example, will suffice, and one or a small number of solid metaphors to convey the message, there are circumstances where more explicit software architecture documentation will be needed: to communicate to a large audience, to comply with external regulations, for example.

6) **Method**: How are we identifying and resolving architectural issues? While we saw above that agile methods are not opposed to the concept of architecture, they are all rather silent on how to proceed to identify architecturally significant requirements, to perform incremental architectural design, to validate architectural features, etc. There are architectural methods, see for a quick inventory, but they are not well known.

7) **Value and cost**: All agile approaches strive to deliver business value early and often. The problem seems often that while the cost of architecture is somewhat visible, its value is hard to grasp, as it remains very invisible. An approach such as the *Incremental Funding Method* [12] may allow casting the right compromise between architecture and functionality, without falling into the trap of "Big Up-Front Design".

In this tutorial, I propose a method to help define how much architecture is needed for a given project, using the 8 dimensions listed above. This is done in a workshop mode, where the various opposing factions have to draw a common map of the territory. This map will help define a common language, a common mental model of where the various practices fit and contribute to the project success.

We then define a management and technical process to weave architectural activities in an agile iterative process, called the "zipper model", in which a balance is struck between development items or features that provide visible value (to the business) and more invisible features such as architecture.

The concept of *technical debt* is then introduced, using Steve McConnell's taxonomy [13], to show the (possible) negative consequence of a lack of focus on architecture, or to at least make the decisions to not focus on architecture, or to defer it to a later stage, a more enlightened one. We define also heuristics (based on the project's context) as to when deliberately incurring technical debt is a reasonable approach.

Throughout the presentation, a case study of a large organization (which shall remain unnamed) and its unfortunate attempts at both ends of the spectrum [agility ←→ architecture] will be used to illustrate the proposed approach.

In the movie *The Matrix Reloaded* [14], the Architect tells Neo: "The first matrix I designed was quite naturally perfect… a triumph equaled only by its monumental failure. I have since come to understand that the answer eluded me because it required a lesser mind, or perhaps a mind less bound by the parameters of perfection."

3. REFERENCES

[1] Beck, K. 2000. Extreme Programming Explained: Embrace Change. Addison-Wesley.

[2] Schwaber, K. and Beedle, M. 2002. Agile Software Development with SCRUM. Prentice-Hall.

[3] Poppendieck, M. and Poppendieck, T. 2009. Implementing Lean Software Development--From Concept to Cash. Addison Wesley.

[4] Stapleton, J. 1998. DSDM, Dynamic Systems Development Method: The Method in Practice. Addison-Wesley.

[5] Bass, L., Clements, P., and Kazman, R. 2003. Software Architecture in Practice. 2nd ed. Addison-Wesley.

[6] ISO/IEC 42010 CD1, 2010. Systems and software engineering—Architectural description, committee draft #1. (February 2010) ISO.

[7] Spencer-Oatey, H. 2000. Culturally Speaking: Managing Rapport through Talk across Cultures. Cassel.

[8] Kruchten, P. 2007. Voyage in the Agile Memeplex: Agility, Agilese, Agilitis, Agilology. ACM Queue. 5, 5 (July/August). 38-44.

[9] Agile Alliance, 2000. Manifesto for Agile Software Development. Available at: http://agilemanifesto.org.

[10] Kruchten, P. 2003. The Rational Unified Process: An Introduction. 3rd ed. Addison-Wesley.

[11] Fowler, M. 2003. Who needs an architect? IEEE Software. 20, 4 (July/August). 2-4.

[12] Denne, M., and Cleland-Huang, J. 2004. The Incremental Funding Method: Data-Driven Software Development. IEEE Software. 21, 3. (May/June), 39-47.

[13] McConnell, S. 2007. Technical debt. Accessible at: http://blogs.construx.com/blogs/stevemcc/archive/2007/11/01/technical-debt-2.aspx.

[14] Wachowski, A. and Wachowski, L. (writers and directors) 2003. The Matrix Reloaded. Warner Bros