How much architecture? Reducing the up-front effort

Michael Waterman, James Noble, George Allan
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
Email: Michael.Waterman@ecs.vuw.ac.nz, kjx@ecs.vuw.ac.nz, George.Allan@ecs.vuw.ac.nz

***Abstract*—A key part of software architecture is the design of the high level structure of a software system – an exercise in planning ahead. Agile software development methods discourage planning ahead, encapsulated by the Agile Manifesto philosophy "[we value] responding to change over following a plan". Development without architecture planning risks failure. This leads to an apparent paradox: how can you design an architecture while using a methodology that promotes not planning ahead?**

This paper introduces Grounded Theory research that is exploring the factors that affect how much architecture planning industry practitioners do up-front – in other words, how much architecture? Early results show that the experience of the architects and predefined or template architectures both help to reduce the architectural effort required without sacrificing the benefits of a full architecture design.

***Keywords*-Software architecture, Software development management, Software engineering**

I. Introduction

Software architecture can be defined as "the set of significant decisions about the high level structure and the behaviour of a software system" [1]. Architecture design is therefore an exercise in planning ahead; however one of the key principles of agile development is to not plan ahead [2], codified by the extreme programming (XP) mantra "you ain't gonna need it" [3]. But even agile development usually benefits from some level of up-front architecture design. This raises an apparent paradox [4]: how can you design an architecture while using a methodology that promotes not planning ahead?

This research will explore this paradox and investigate the effects that the architectural skills, judgement and tacit knowledge of the development team, and the methods that they use, have on the level of up-front architecture design in their agile development projects.

Section II describes the research background; section III describes the research method, Grounded Theory; section IV presents the work done to date; and section V presents some results from early interviews. Finally section VI draws conclusions from the results and outlines the future directions for this research.

II. Background

Architecture planning involves making system-level decisions such as the hardware platform, development technology and architectural patterns. Any changes required at the architectural level may therefore affect the whole system [5]; not having a suitable architecture defined before development starts may significantly increase the cost of development through large amounts of refactoring.

Pure agile development methods appear to promote unplanned, emergent architecture [6]. The Agile Manifesto, at the heart of agile, gives the agile characteristic of *responding to change* higher value than the traditional development characteristic of *following a plan* [2] that includes designing the system in full before handing over to the developers, the so-called 'big up-front design'. Agile practitioners often see architecture as being anti-agile [7] and therefore go for the "no up-front design" approach of development, and rely on refactoring to maintain code quality. Problems caused by a series of ad hoc, unplanned decisions can cause projects to gradually fail – "the death of a thousand cuts" [8]. If not a straight-out failure, they leave the project exposed to the risk of missed milestones and exceeding budget constraints.

Just enough up-front design is an intermediate solution in which agile developers make only the architectural decisions up-front that must be made to allow development to get started [9]. Just enough up-front design can be implemented in a number of ways. In general, the set-up phase of the project (often called 'iteration 0' [7]) includes some up-front design, with ongoing architectural refinement during the iterative development.

Agile methods are however generally silent on how to identify the architecturally significant requirements that should be included in iteration 0, how to perform incremental architecture design and how to validate architectural features [5].

How can an architect or developer determine what the correct amount of "just enough up-front design" is, and thereby answer the question "how much architecture?"

"How much architecture?" does not have a single correct answer. Booch (2007) and Fairbanks (2010) noted that a particular system may not even have a single correct architecture [10], [11]. How much up-front architecture depends strongly on context: Abrahamsson et al. (2010) listed a number of technical and business factors that affect context, including size, criticality, architecture stability and the business model [5]. It goes further than this however:

978-0-7695-4657-5/12 $26.00 © 2012 IEEE
DOI 10.1109/AgileIndia.2012.11

Spinellis and Gousios (2009) included the programmers themselves as part of the context [12]. Bass et al. (2003) commented that any two architects are likely to produce different architectures for the same problem with the same boundaries, for "software architecture is a result of technical, business and social influences" [13] – and this includes "the background and experience of the architects". In other words, architecture depends not only on the technical and business constraints of a system, but also on the experience of the architect and of the development team, on their judgement and abilities, and on what *they believe* to be the correct architecture.

Agile development adds another complicating factor: the requirements are not known *a priori*, and therefore the minimum amount of architectural effort cannot be rationally determined *a priori*.

This research is exploring how agile practitioners determine how much architecture is planned up-front in their systems – that is, how much architecture? In particular, this paper focuses on "how can agile teams reduce the up-front effort without sacrificing the benefits of an up-front design?"

III. Research method

The most appropriate type of research for investigating up-front architecture is qualitative research. Qualitative research is used to investigate people, interactions and processes, and as noted above architecture is very dependent on the architects themselves and the development teams. In addition qualitative research is generally *inductive* – it develops theory from the research, unlike *deductive* research which aims to prove (or disprove) a hypothesis. Because of the scarcity of literature on the relationship between architecture and agile [14], it has not been possible to develop hypotheses (or a hypothesis) to test using deductive methodologies.

Grounded Theory was selected because it allows the researcher to develop a substantive theory that explains the processes observed in a range of cases [15].

In this Grounded Theory research, data was collected through semi-structured interviews with agile practitioners who have knowledge of architecture, such as architects, developers, project leaders and team managers. To maintain confidentiality, we refer to our participants using numbers (P1, P2 and so on).

The steps of Grounded Theory are *open coding* the data to identify points of research interest; *constant comparison* to derive *concepts* from the codes and *categories* from the concepts; using these categories and their inter-relationships to derive the emerging theory; *memos* are written to develop the relationships between codes, concepts and categories, and to aid their development; and *selective coding* to focus the core category.

For example in an interview, participant 'P2' commented their team used the architectures recommended by the vendor for the particular type of system they were developing:

> *"...so that [CQRS] paper plus the Windows architecture guidelines plus the off-the-shelf componentry that comes from Microsoft, it means we don't really write much! Which is good!" (P2)*

This was coded as "using vendor guidelines".

The emerging codes were constantly compared with existing codes from this and earlier interviews. Codes that had similar themes to this example included "using template architectures", "using tools to simplify decision making" and "using a product-defined architecture". These codes were combined into a concept called "using predefined architecture". Figure 1 shows the relationship between the underlying codes and the concept graphically.

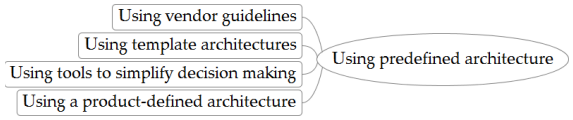

Figure 1. An example of a concept emerging from its codes

Grounded Theory uses iteration to ensure a wide coverage of the factors that may affect the emerging theory: later data collection is dependent on the results of earlier analysis.

IV. Work done

We have completed eleven initial interviews with agile practitioners from New Zealand and the United Kingdom identified at focus group meetings. The participants cover a range of roles, including contractors, developers, architects, business analysts and managers. The organisation types are also varied, and include government departments, development consultancies, several companies producing mass-market software for their own customers, and a self-employed developer. The participants also come from a range of agile development projects with diverse domains, customers, levels of agility and architectures. These interviews have provided very general data about the participants' experience with agile development and architecture. Architectural documentation is also gathered where possible.

The interview data has been open coded, and a number of concepts have emerged. A group of these concepts relate to reducing the amount of effort in up-front decision making: "using predefined architecture", "intuitive architecture", "having architectural experience simplifies decision making" and "being familiar with the architecture".

These related concepts have been grouped into a category called "simplifying architecture design". The concepts within this category are descibed in more detail below. Other concepts have emerged but have not formed categories yet.

A. Using predefined architecture

Participants talked of a number of types of predefined architectures such as template architectures, vendor-recommended architectures and tools that automate design.

Some participants noted that template architectures, where the team selects an off-the-shelf architecture and "fills in the gaps", are becoming increasingly common:

> *"[...] a lot of vendors now have what they call candidate or template architectures, it's just going to be one of those." (P4)*

P2 commented that their team used the architectures recommended by the vendor and by third parties for the particular type of system they were developing:

> *"...so that [CQRS] paper plus the Windows architecture guidelines plus the off-the-shelf componentry that comes from Microsoft, it means we don't really write much! Which is good!" (P2)*

Both P4 and P6 suggested that the architect's job is becoming simpler as a result of tools that help them design an architecture, with decisions that used to be considered architectural now being considered part of development (that is, they are design decisions):

> *"What used to be architectural decisions ten years ago can now almost be considered design decisions because the tools allow you to change these things more easily." (P4)*

and

> *"...software development has moved on a lot [...] you can expect either to have your code more intelligent and diverse or at least to have some kind of generation template that allow the code, if it is repetitive, to be written for you so you don't have to go and write every line yourself." (P6)*

B. Intuitive architecture

A number of participants noted that the high-level architecture could be determined intuitively. When P1 was asked how he decided on his architecture, he replied:

> *"It was just something that I had to do!" (P1)*

P4 answered the same question similarly, describing the architecture as being "obvious", and explained it thus:

> *"...so because I know it needs to scale to 2000 users, [...] the simplest thing that will work, taking into account the fact that it's got to meet these quality requirements, means that it's got to be three tier, and means that instead of just writing it to a flat file I will have a database..." (P4)*

C. Having architectural experience simplifies decision making

Two contradictory views emerged on experience: having experience simplifies decision making, and lack of experience simplifies decision making:

P3 and P6 gave examples of experience simplifying decision making:

> *"If you have a team of really, really experienced professionals then you can get away with a lot less [design up-front] because a lot of it is going to be implicit anyway." (P3)*

while P6 noted:

> *"Very often, if you're experienced, you'll know, you could write down three options and you'd know if you do it this way you'll have these problems; you can make a value judgement much more efficiently." (P6)*

On the other hand P1 gave an example where a lack of experience restricted his options to just one:

> *"With the prototype I chose ASP because I knew it, I could do VBScript, I didn't know C#" (P1)*

He then noted that if he knew C# he would have used it, implying the older technology (ASP/VBScript) was possibly not the best solution.

D. Being familiar with the architecture

While some participants felt that being experienced was helpful, other architects suggested that familarity with the particular architecture was important to them:

> *"Part of my current productivity might be because I am working in a system that is in an architecture that I'm very, very familiar with" (P6)*

while P8's team, though experienced, recognised they were not familiar with the architecture and where its traps and pitfalls lay, so sought assistance:

> *"[A particular organisation] had done something [with] similar processes, so we were able to go to them and, 'what's your architecture, can we have a look at it, why have you made these decisions?', which was very useful." (P8)*

On the other hand, P7's team were not able to seek assistance and when they ran into problems they put the blame on the unfamiliarity with the chosen architecture:

> *"Oh, you can always say it's because we didn't think about it enough, but I think the reality is we haven't done precisely this before, and not with this particular tool set, so it's unfamiliarity." (P7)*

V. Findings

Participants have talked about a number of methods or factors that reduce the amount of up-front architecture design effort. These factors are not unique to agile development, but they are certainly aligned with the agile practices of less up-front design.

Predefined architectures allow the architects to get more architecture design done up-front with less effort. These architectures may be off-the-shelf architectures or template architectures where the development team simply has to fill in the details. In particular, vendors' development frameworks often have recommended ("candidate") architectures that developers can choose that will work best for their

particular problem. One participant noted that certain frameworks or tools allow changes to be made more easily, making architects' jobs simpler as a result – architectural work is becoming more like design work or even configuration work – and therefore making it less important if that decision later proves to be less than ideal or incorrect.

These template architectures allow a team to reduce the amount of work required to develop an architecture, and therefore the amount up-front planning that is required but retain the benefits of an explicitly design architecture. They can also allow changes to the architecture to be made a lot more easily, making it less important to get architectural decisions correct up-front.

Having a broad architectural experience simplifies the decision making process through *tacit knowledge*: as you become more experienced you learn the advantages and disadvantages of different solutions, better understand what is involved with the alternatives, and therefore become better at designing architecture, so that an experienced architect can make decisions implicitly or intuitively with little explicit architectural effort. In effect, the architecture becomes "obvious" or they "just know". Boehm observed that much of the agility in agile methods comes from the tacit knowledge of the team members [16]; inexperienced architects or developers don't have this knowledge and have to refer to written knowledge (or tools) to help them make decisions. Experienced designers are also better at making judgement calls and knowing how to spend their time most efficiently so they don't spend time on things they may have to refactor or throw away. (One participant noted a time when experience did not mean a quicker decision – it did, however, mean a better decision because the architect was able to consider more options and select the most appropriate one.) Curiously, participants have also noted that having *narrow* experience can simplify the decision making process as well – for a different reason: they may know only one solution and therefore not have to choose between alternatives!

Even experienced architects may run into problems if they are not familiar with the architecture they are working with because they may not be aware of the areas that can cause problems. Familiarity allows the team to know where to focus their efforts and to avoid problems.

VI. Conclusions

Predefined architectures allow a team to reduce the amount of work required to develop an architecture, and therefore the amount up-front planning that is required – very relevant to agile development – but retain the benefits of an explicitly design architecture. They can also allow changes to the architecture to be made a lot easier, making it less critical to get them right up-front.

Architectural experience is important because it allows decisions to be made implicitly through tacit knowledge – in line with the agile preference to avoid excessive written documentation. Familiarity with an architecture allows the team to avoid problems caused by not recognising areas that may cause problems later.

These preliminary results are based on an initial analysis of eleven interviews. The next steps of this research will be additional interviews and the analysis of the ensuing data. Our focus for additional interviews will be on expanding the category included in this chapter, which relate to reducing the amount of architectural design, plus developing additional categories that will also help determine the balance between architecture and agile, and thus answer the question "how much architecture?"

References

[1] P. Kruchten, *The Rational Unified process – an Introduction*. Addison Wesley, 1998.

[2] K. Beck *et al.*, "Agile manifesto." http://agilemanifesto.org/, 2001.

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

[4] P. Kruchten, "Agility and architecture: an oxymoron?," *SAC 21 Workshop: Software Architecture Challenges in the 21st Century*, 2009.

[5] P. Abrahamsson, M. A. Babar, and P. Kruchten, "Agility and architecture: Can they coexist?," *IEEE Software*, vol. 27, pp. 16–22, 2010.

[6] D. Mancl, S. D. Fraser, and B. Opdyke, "Workshop: architecture in an agile world," in *SPLASH*, pp. 289–290, ACM, 2010.

[7] R. Nord and J. Tomayko, "Software architecture-centric methods and agile development," *IEEE Software*, vol. 23, pp. 47–53, Mar.–Apr. 2006.

[8] G. Booch, "The defenestration of superfluous architecture accoutrements," *SAC 21 Workshop: Software Architecture Challenges in the 21st Century*, June 2009.

[9] B. Martin, "The scatology of agile architecture." http://blog.objectmentor.com/articles/2009/04/25/the-scatology-of-agile-architecture.

[10] G. Booch, "The irrelevance of architecture," *IEEE Software*, vol. 24, pp. 10–11, May–Jun. 2007.

[11] G. Fairbanks, *Just enough software architecture: A risk driven approach*. Marshall and Brainerd, 2010.

[12] D. Spinellis and G. Gousios, eds., *Beautiful architecture*. O'Reilly, 2009.

[13] L. Bass, P. Clements, and R. Kazman, *Software Architecture in practice*. SEI Series in software engineering, Addison-Wesley, 2 ed., 2003.

[14] H. P. Breivold, D. Sundmark, P. Wallin, and S. Larson, "What does research say about agile and architecture?," *Fifth International Conference on Software Engineering Advances*, 2010.

[15] A. Strauss and J. Corbin, "Grounded theory methodology," in *Handbook of Qualitative Research* (N. K. Denzin and Y. S. Lincoln, eds.), Sage Publications, Inc, 1994.

[16] B. Boehm, "Get ready for agile methods, with care," *Computer*, vol. 35, no. 01, pp. 64–69, 2002.