Architecture in Large Scale Agile Development

Jutta Eckstein

Independent, Gaussstr. 29, 38106 Braunschweig, Germany
Jutta@JEckstein.com

Abstract. In order to welcome changing requirements (even late in development) agile development should enable the architecture to incorporate these changes and therefore to emerge over time. This implies not finalizing the architecture upfront. Moreover, in small agile teams it is assumed that there is no dedicated role for an architect – instead the whole team should be responsible for the architecture. In large-scale agile development the requirement for an emergent architecture still holds true. However, it is unrealistic to ask members of e.g. ten teams to be equally responsible for the architecture. Moreover, the role and support for the architecture depends not only on the degree of the size but as well on the degree of complexity. In this paper I report on the experience using different models for supporting emergent architecture in large environments that take the degree of complexity into account.

Keywords: agile methods, architect, change, chief architect, complexity, community of practice, emergent architecture, large-scale agile software development, project management, software engineering, technical consulting team, technical service team, uncertainty.

1 Introduction

Agile development focuses on maximizing the business value at all times. In small agile development this is addressed by a cross-functional team, which Scrum called a Scrum Team [1]. The developers on such a team encompass all competencies, skills, and know-how needed to deliver frequently product increments. There are no explicit roles like tester or database expert for the developers in order to stress the joint responsibility for the delivery. This structure allows such a team to work independently in a self-organized manner.

Scaling up agile development does not change the goal of maximizing the business value continuously. However, for large-scale agile development it is crucial to provide a supporting team structure. Thus, instead of structuring teams according to know-how (like user interfaces or databases), activities (like business analysis or testing), or components (as defined by i.e. architectural layers), teams have to be structured –cross-functional– around the business value. Only this allows teams in large-scale agile development to self-organize and to deliver business value frequently and regularly. Such teams are called domain or feature teams [2, 3] and are defined by Larman and Vodde as:

T. Dingsøyr et al. (Eds.): XP 2014 Workshops, LNBIP 199, pp. 21–29, 2014.

> "*A **feature team** […] is a long-lived, cross-functional, cross-component team that completes many end-to-end customer features–one by one.*" ([2], p. 549).

This inherent focus on business value by the team's structure contrasts structuring teams based on components as suggested by Leffingwell, who states:

> "*Components are the architectural building blocks of large-scale systems. Agile teams should organize around components, each of which can be defined/built/tested by the team that is accountable for delivering it.*" ([4], p. 204).

Thus, instead of structuring the teams around the business value, Leffingwell suggests to structure them around architectural components. Consequently, he suggests for scaling-up and implementing what he calls an architectural runway to add more component teams. Yet, large-scale agile development should concentrate on delivering primarily customer value and not primarily components.

However, focusing on the business value still requires an architecture that allows adding features over time. Ideally, we would know upfront what kind of features will have to be added later by knowing the *intent* of the product [5]. Yet, as Kruchten clarifies:

> "*In reality, in most software development projects, we define* Intent *gradually, and it tends to evolve throughout the project under various pressures and demands for changes.*" ([5], p. 7)

This implies that it is not possible to finalize the architecture upfront because the added features might force an architectural change. Thus, focusing on the business value requires that the architecture emerges or rather changes over time. In small agile teams, it is the whole team's responsibility to ensure the evolvement of the architecture without a dedicated role for an architect [1]. In large-scale agile development it is unrealistic to ask all members of the undertaking to decide on architectural issues jointly, because this could be a hundred-plus people.

In this paper, I will examine the different possibilities for supporting emergent architecture in a large environment. The architecture is labeled as emergent, for emphasizing the understanding that it is not possible to stabilize the architecture at the beginning of the undertaking. This means the architecture will change over time. After clarifying architectural complexity, section two will focus on three different models: First on the support of a relatively stable architecture which will only have to adjust to a few changes, thus on low complexity; Next on the opposite–the support for the creation of a new architecture which is accompanied by high uncertainty and frequent changes and therefore on high complexity; And finally on the complexity in between–the support for an architecture that needs to be adaptive in order to deal with some changes and a medium degree of uncertainty. In section three further issues are discussed and section four provides a final conclusion.

2 Supporting Architecture

As Leffingwell mentions:

> *"The larger and more complex the system and the higher the criticality of failure, the more the teams will need to base their daily decisions on an agreed-upon and intentional architecture […]."* ([4], p. 202).

Leffingwell does not explain what is meant by complex, yet the statements still holds true. For example, it makes a difference if the system a team is working on is about to be created, still tremendously changing, or if it is quite stable. These differences mark the complexity of the system and subsequently as well of the architecture [6]. Kruchten emphasizes moreover, that among others the pre-existence of a stable architecture and the rate of change are important dimensions that define the context for a project [5]. The complexity that is important for addressing architectural support is expressed by the relationship between the required changes and the existing uncertainties (see Fig. 1).

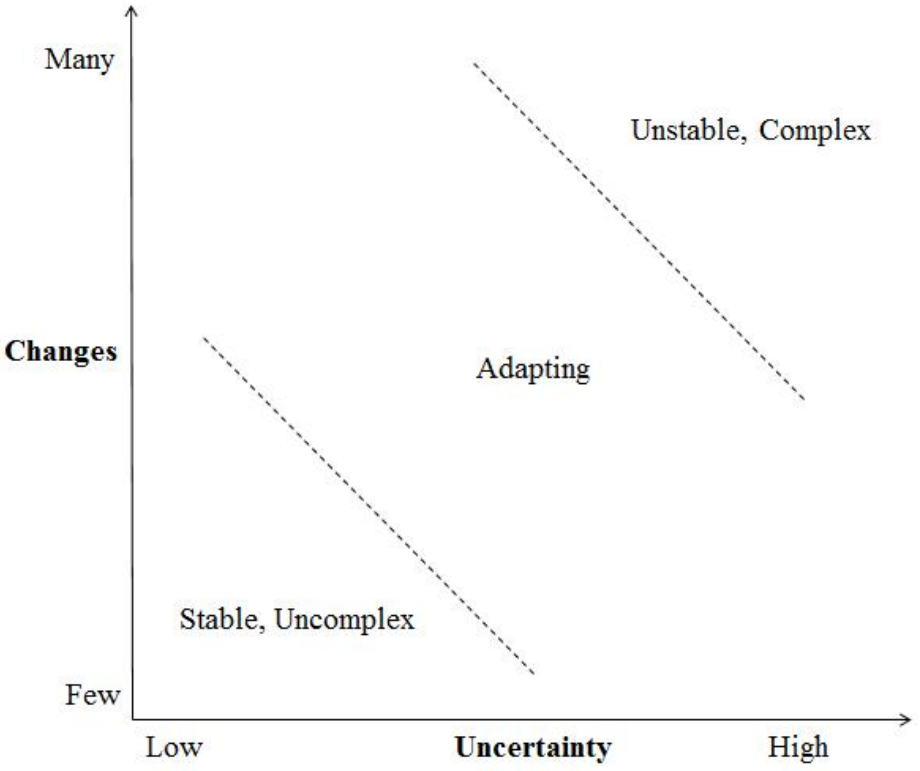

Fig. 1. Complexity of architecture based on changes and uncertainties

This expression of complexity is related to the so-called Landscape Diagram [7] which has originally been created by Stacey [8] and then further developed by Zimmerman, Lindberg, and Plsek [9]. The three subsequent models presented below, deal with those different kinds of architectural complexity.

As figure 1 shows, the complexity of the architecture is defined on the one hand by the uncertainty (x-axis) and on the other hand by the requests for changes (y-axis). According to Kruchten, uncertainty is defined by the uncertainty in the intent, e.g. the business domain; in the work, e.g. the tools or environment; the people, e.g. the know-how; and the final product [5]. For example, the business domain could be new to the developers and/or to the customer, in case the customer wants to enter a new market segment. The technology used to implement the product could be new to the team and could be additionally of cutting-edge without a lot of experiences by other projects, people, or companies. In these cases, it is very likely that uncertainty is experienced as high. The rate of change, mainly in terms of changing (business) requirements, but also in terms of tools or people influences the stability of the architecture. Thus, the architecture will be the more unstable and complex, the more changes and the higher the uncertainties are.

Subsequently will be examined what kind of support is useful for an architecture that falls in an area with only a few changes and low uncertainties; for one that is located in the area with a high rate of changes and uncertainties; and finally one that sits in between with moderate changes and uncertainty.

Examining complexity this way shall help to decide on the necessary architectural support. Thus, in relation to George Box' famous quote: "Essentially, all models are wrong, but some are useful." –here different models are more or less useful depending on the complexity.

2.1 Supporting a Stable Architecture

Typically, long-term projects and long-term product development do not require severe changes in the architecture once they are on track[1]. This kind of development is marked by high certainty in terms of the technology used and of the business requested.

Very often the major concern is to keep the architecture stable and allowing it to evolve gradually with subsequent business needs. We have solved the support of such an architecture in two different ways by either a community of practice or by a chief architect:

Community of Practice. A community of practice (CoP, see [10]) has been suggested also by Larman and Vodde for large scale agile development. In particular, they propose a design/architecture community of practice and define CoP as "an organizational mechanism to create virtual groups for related concerns." ([2], p. 313).

[1] Thanks to Philippe Kruchten for the following additional remark that even for the ones that are not on track, typically no severe changes are required, because performing the changes is too costly and risky (Kruchten's comment while reviewing an earlier version of this paper).

The idea is that every cross-functional feature team covers the role of an architect. This is a role and is therefore not bound to a specific person. However, in practice quite often only a few team members are willing and skilled for taking this role. Whenever an architectural decision has to be made, these "architects" of the diverse feature teams assemble (this could as well happen virtually) and decide upon the request. Sometimes the feature teams decide that the CoP meets regularly in order to monitor any changes and possible improvements within the architecture.

Chief Architect. Instead of a CoP a single person can provide the main support for the architecture. Next to being technologically skilled, the main requirement for this person is to be as well socially skilled. The chief architect (sometimes also called architecture owner) needs to work closely with all different feature teams, which requires architecting by wandering around [11]. This approach allows the chief architect (a) to understand the needs of the teams; (b) to ensure the teams understand the architecture; and (c) to help improving the architecture.

2.2 Supporting an Unstable Architecture

Starting a new project or creating a new product involves most often many uncertainties. Those uncertainties refer to the technology used, the understanding of the requirements, and making the "right" decisions both business and technology wise.

Additionally very often this uncertainty is accompanied by the fact that the team is newly assembled and has to go through different phases until it performs [12]. Moreover, if the undertaking would be started by e.g. ten teams the system would as well be split technology-wise into ten parts [13]. Thus, starting from day one e.g. with ten feature teams is not recommended.

Instead in order to scale, the system has to be enabled to scale. The recommended model for an unstable and heavily changing architecture is to establish a technical service team:

Technical Service Team. Instead of spreading the support for the architecture across all feature teams by a CoP or by asking a single person to provide that support as the chief architect, this role is taken in the context of high complexity by a specific team: A technical service team [3]. The key is that this team provides a service to the feature teams – or in other words, the customers of this team are the feature teams. This means in turn that the feature teams have to act as well as a customer and provide a product owner for that team, who decides on the priorities of the (technical) stories the feature teams require. This is the big difference to a non-agile architecture team which defines the architecture upfront (and sometimes also builds it) but is not driven by the feature teams' requests. Such kind of a non-agile architecture team is often regarded as being disconnected from reality and project members think of them of being located on an ivory tower far away from the actual needs of the projects.

Sometimes the technical service team is as well the starting team [14]. In such a situation, this team creates the base architecture founded on i.e. three key user stories

which will be implemented as well by this team. Only after implementing the base architecture along with these i.e. three user stories, the feature teams will join the undertaking. Then still, depending on the complexity either the technical service team remains as described above and will be guided by a product owner representing the feature teams or the technical service team dissolves in the diverse feature teams.

2.3 Supporting an Adaptive Architecture

If both the requested changes and the uncertainty are moderate, the architecture needs as well moderate support in order to being adaptive. In this situation the architecture is not really stable.

Therefore, it needs more attention than just by a single person as the chief architect. The burden would also be too high for a CoP, because the members of the CoP would be required to synchronize continuously and to focus almost only on architectural issues. As a result, the feature teams would not be able to concentrate on the business value, because at least one of their members would have to concentrate on the architecture at all times. Thus, the recommended model is to establish a technical consulting team:

Technical Consulting Team. This is a mix of the chief architect or the CoP and the technical service team. So like the chief architect, the individual members of this team provide their support by wandering around. And like the CoP, the individual members of the technical consulting team will most often offer their support (in terms of consulting, coaching, mentoring, and pair programming) to a specific feature team during an iteration. Thus, a member of the technical consulting team will act as a regular feature team member during the course of an iteration and is as such as responsible for (or committed to) the iteration goal as every other feature team member. Yet, for the next iteration this person might support a different feature team.

But unlike the CoP, the technical consulting team is typically smaller in number than the amount of feature teams involved in the undertaking. E.g. in one project we had fifteen feature teams, yet only seven team members in the technical consulting team. Thus, not every feature team had the support of a technical consulting team member in each and every iteration. Supporting every feature team in each iteration this is typically not needed for an architecture of medium complexity.

In case a major change in the architecture is required, the technical consulting team provides this change as a service to the feature teams by implementing it, just like the technical service team.

3 Discussion

Different levels of complexity require different models for supporting the emergence of the architecture. See figure 2 for an overview of these different models.

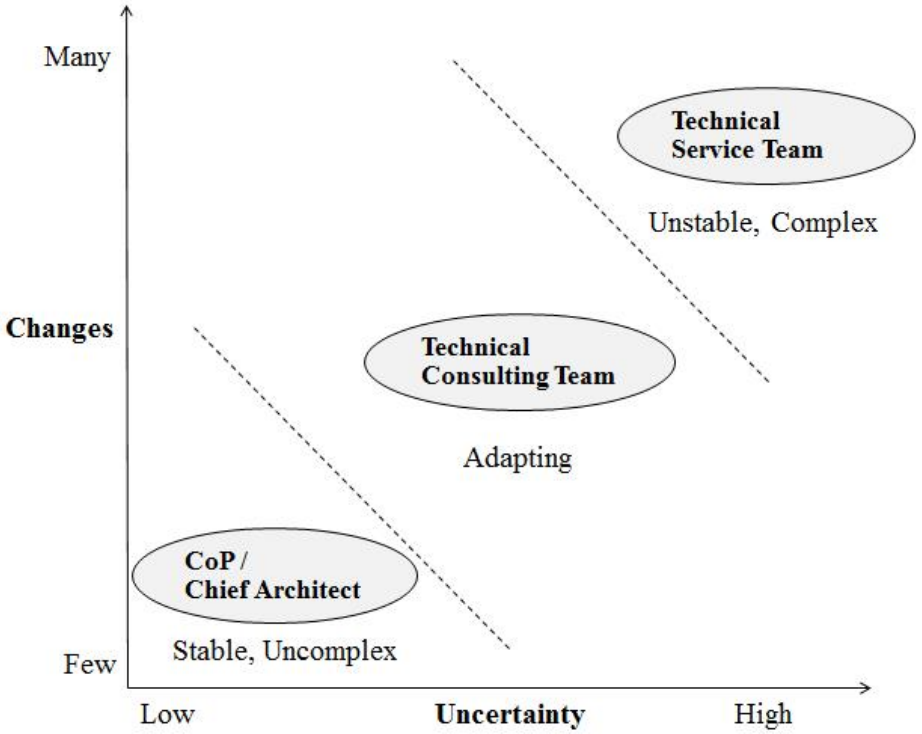

Fig. 2. Different models for architectural support depending on the complexity

The consequences of using the different models in other circumstances than recommended should not be underestimated: For example, if the system experiences many changes accompanied with high uncertainty, a single chief architect would be overwhelmed with the demands. For the feature teams this would mean, either to wait for a decision or to come up with an own one. The latter is not per se a bad idea, yet it could create the problem that different feature teams come up with contradicting solutions to similar problems. This results in breaking the conceptual integrity which in turn makes it harder for both implementing new functionality and maintaining the system.

The diverse teams (technical service team, technical consulting team, or CoP) that support the architecture can organize themselves in different ways. Some of those teams might decide on requiring a leader for the team. This person is then often perceived as the chief architect. However, it is important to distinguish this role from the chief architect in a stable environment who is not the leader of a particular team.

In many cases the complexity will change over time (see Fig. 3). Different developments of the complexity can happen, yet most often the complexity will decrease over time [5]. Most likely the uncertainty will lower, because the business domain and the technology will be better known. Frequently this results in fewer changes, because the uncertainty decreases for the customer as well over time.

As figure 3 shows, the decrease of the complexity over time affects the required architectural support. Thus, often large-scale agile development starts with the support of a technical service team. This team might even be the single starting team before the whole undertaking will be scaled up. As the architecture is getting more and more stable and less architectural services are required for the feature teams, the technical service team will be shrunk and turned into a technical consulting team. While the architecture is stabilizing even more and even fewer changes are required, the technical consulting team disappears – maybe one member remains as the chief architect, maybe all members will become members of the diverse feature teams. Either those people or different members of the feature teams will then ensure the conceptual integrity of the architecture through a CoP.

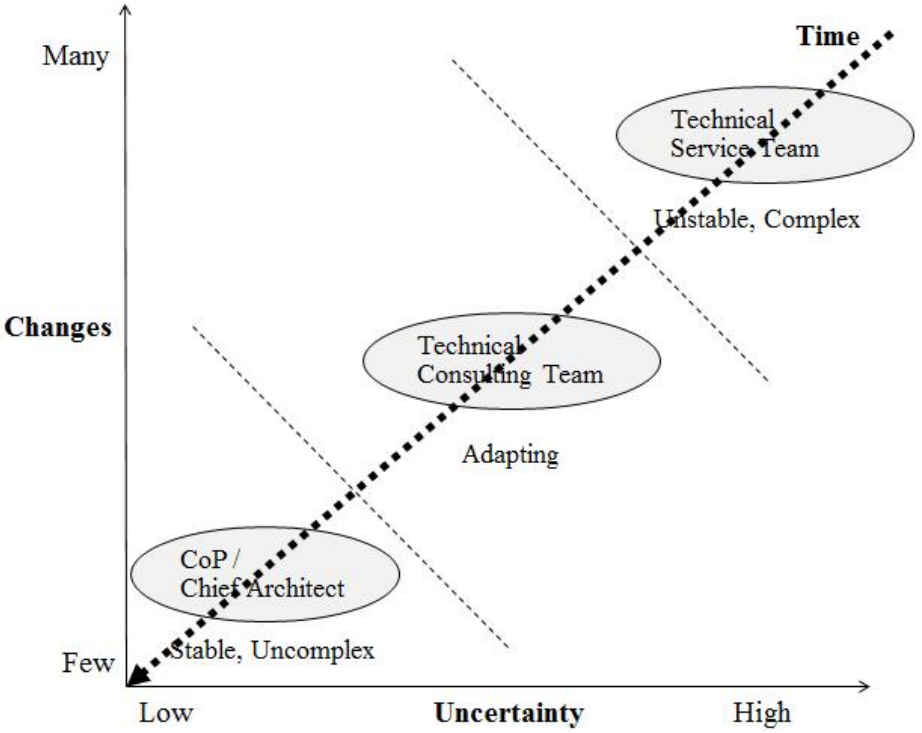

Fig. 3. Complexity decreases over time

4 Closing

Depending on the complexity –defined by the degree of uncertainty and requested changes– different models have been presented for supporting architectural support in large-scale agile development. All of these models have been applied by the author. Those models have not only been used when working on project or product development, yet as well when scaling to product line development or supporting an

organization-wide architecture. Therefore, these models have as well been proven in praxis when scaling up to the whole organization and combining the efforts on supporting different architectures on a higher level.

For large-scale agile development it is essential to provide architectural support without losing focus on the business value. Yet, concentrating on the business value only leads to the loss of conceptual integrity. Thus, both dimensions –business and technology (the latter in terms of architecture) – have to be taken into account for large-scale agile development.

References

1. Sutherland, J., Schwaber, K.: The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game, https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/2013/Scrum-Guide.pdf
2. A survey of current research on online communities of practice. Harvard Business School Press, Boston (2002)
3. Larman, C., Vodde, B.: Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum. Addison-Wesley, Upper Saddle River (2010)
4. Eckstein, J.: Agile Software Development in the Large: Diving into the Deep. Dorset House Publishing, New York (2004)
5. Leffingwell, D.: Scaling Software Agility: Best Practices for Large Enterprises. Addison-Wesley, Upper Saddle River (2007)
6. Kruchten, P.: The frog and the octopus: A conceptual model of software development (2011), http://arxiv.org/pdf/1209.1327 (last accessed: June 18, 2014)
7. Eckstein, J.: Roles and Responsibilities in Feature Teams. In: Šmite, D., Moe, N.B., Ågerfalk, P.J. (eds.) Agility Across Time and Space: Implementing Agile Methods in Global Software Projects, pp. 289–299. Springer, Heidelberg (2010)
8. Holladay, R., Quade, K.: Influencing Patterns for Change. CreateSpace Independent Publishing Platform (2008)
9. Stacey, R.D.: Strategic Management and Organizational Dynamics, 2nd edn. Pitman Publishing, Berlin (1996)
10. Wenger, E.C., McDermott, R., Snyder, W.M.: Cultivating Communities
11. Zimmerman, B., Lindberg, C., Plsek, P.: Edgeware: lessons from complexity science for health care leaders. V H A Incorporated (Curt Lindberg, Plexus Institute) (2008)
12. Peters, T., Waterman, R.H.: Search of Excellence, 2nd edn. Profile Books Ltd. (2004)
13. Tuckman, B.: Developmental sequence in small groups. Psychological Bulletin (63) (1965)
14. Conway, M.E.: How Do Committees Invent? Datamation 14(4) (1968)
15. Eckstein, J.: Agile Software Development with Distributed Teams: Staying Agile in a Global World. Dorset House Publishing, New York (2010)