Elaboration on an Integrated Architecture and Requirement Practice

Prototyping with Quality Attribute Focus

Stephany Bellomo, Robert L. Nord, Ipek Ozkaya
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA, USA
sbellomo@sei.cmu.edu, rn@sei.cmu.edu, ozkaya@sei.cmu.edu

Abstract— Projects seeking rapid, sustainable delivery are combining agile and architecture practices to manage competing goals of speed in the short term and stability. In a recent study, we interviewed eight government and commercial project teams that have adopted incremental and iterative software development approaches and identified a mix of Agile and architecture practices that teams apply to rapidly field software and minimize disruption and delay. In this paper, we elaborate one practice from this study, *Prototyping with quality attribute focus*, to gain a better understanding of how this practice works and what the benefits of the approach are. As we analyzed this practice, we observed that it leverages rapid feedback cycles weaving requirements and architecture, characteristic of the Twin Peaks concept, at three levels: feature development/sprint, release, and portfolio planning levels. We also observed that each of these cycles have differing degrees of separation and cadences. We also describe several regularly occurring integration points within the Scrum framework that allow for synching (weaving of architecture and requirements). We describe the practice in some detail and also discuss a few enablers that keep the practice working smoothly.

***Index Terms*—agile software development, architecture, quality attribute, prototyping, release planning, requirements, software development practices, architecture trade-off**

I. INTRODUCTION

Projects seeking rapid, sustainable delivery are combining agile and architecture practices to manage competing goals of speed in the short term and stability over the long term[1][2][3]. This paper stems from a study in which we interviewed eight project teams identifying a set of practices that enable rapid delivery. The practices that emerged from the study represent a mix of Agile practices, architecture practices and practices that combine these together (we refer to these as integrated practices)[4][5][6]. In this paper, we elaborate one of the more frequently used integrated practices from the study, *Prototyping with quality attribute focus* (shown in Figure 2). This practice integrates prototyping (often leveraged on Agile projects to reduce uncertainty instead of developing lengthy requirements specification documents [7][8]) and architectural focus (consideration of quality attribute requirements during prototyping).

During our interviews we captured several examples of *prototyping with quality attribute focus* practice from teams in different organizations. In this paper, we specifically focus on examples from Team A and Team B (as we refer to them). We begin with an example from Team A. Team A was giving a user demo of a prototype concept when they received unexpected feedback that system performance was slow. The discovery of a performance issue with the prototype concept resulted in weeks of delay. Several problems contributed delay. Due to business pressure, quality attribute aspects of the prototype concept were largely ignored. The architect and product owner had not been collaborating on decisions so the problem was a surprise to the team. The prototyped code was tightly coupled with the development code so the team couldn't make changes to the prototype without holding up the whole release. All these led to additional delays. Finally, the team wasn't prepared to do rapid tradeoff analysis of performance-related design options. So, rather than elaborating the prototyped user story in a smooth spiral fashion as depicted in the Twin Peaks model [9], Team A experienced delays. Figure 1 depicts limitation of trade-off analysis causing delay and impacting subsequent elaboration spirals.

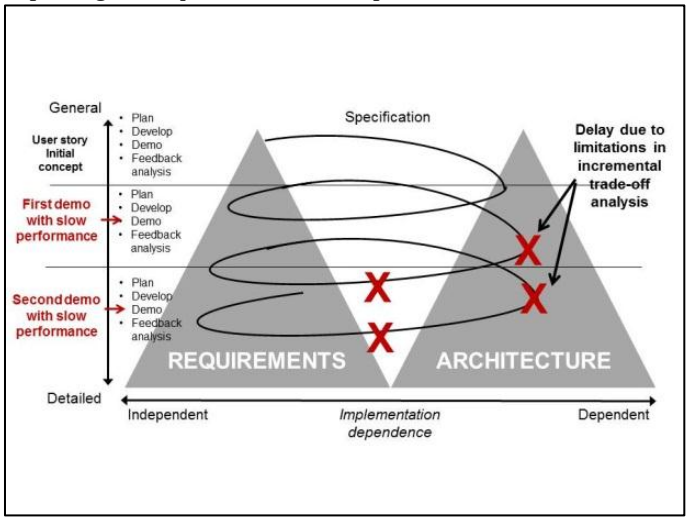

Figure 1: Team A example of delay shown using Twin Peaks model

In this paper, we analyze Team B's *Prototyping with quality attribute focus* practice. Team B's practice examples demonstrate successful use of prototyping for validation of requirements and design concepts including quality attribute-related considerations. A high-level summary of key observations from our analysis of Team B's *Prototyping with quality attribute focus* practice are summarized below:

- Close collaboration between the architect and product owner on Team B at several integration points woven into the project software development lifecycle allow for weaving of architecture and requirements which enables the team to reduce the expectation mismatch as well as risk due to late discovery of requirements (particularly quality attribute requirements).
- Team B's prototyping approach as described in this paper, as well as competency in rapid architectural analysis and a flexible architecture, additionally contribute to the team's ability to smoothly elaborate requirements and architecture that naturally emerges from prototype feedback.

II. BACKGROUND

Here we provide a very minimal overview of the findings of the study from which this practice emerged as a backdrop. We interviewed eight project teams from government and commercial organizations that have adopted incremental and iterative software development practices (such as agile) [4][5]. A set of practices that enable rapid delivery emerged from the study. These practices spanned the software development lifecycle and included a mix of different types of practices; Agile practices, architecture practices and a practices that combine both. Some practices were more widely used than others. A summary of the practices from our interviews are shown in Figure 2 ordered from the most to least used. Integrated practices are shown bold.

PRACTICE SUMMARY
1. Release planning with architecture considerations
2. *Prototyping with quality attribute focus*
3. **Release planning with joint prioritization**
4. Test-driven development with quality attribute focus
5. Dynamic organization and work assignment
6. Release planning with legacy migration strategy
7. Roadmap/vision with external dependency management
8. Root cause analysis to identify architecture issues
9. Dedicated team/specialized expertise for tech insertion
10. Technical debt monitoring with quality attribute focus
11. Focus on strengthening infrastructure (runway)
12. Retrospective and periodic design reviews
13. Use of standards and reference models
14. Backlog grooming
15. Fault handling or performance monitoring
16. Vision document with architecture considerations

Figure 2: Practices summary table

At the time of the interview, Team B was leveraging the *prototyping with quality attribute* focus practice on a project developing a web-based analysis software system. The software had been in production and use for twelve years and used the Scrum development framework. They had organized software development into two week sprints and six to twelve month product releases.

III. PRACTICE DESCRIPTION

As they described their practice, Team B's emphasized that prototyping has been important to the organization in the past but is becoming increasingly important for their survival. As an industry company, they explained that government and budget cuts mean the consequences of bad choices become even bigger; a mismatch in expectations can mean the end of a project. Consequently, requirements validation, technology validation, and architecture validation have all become very important to them. They explained that the value of prototyping (to their team) is that it helps the team ensure that they are delivering what business stakeholders expect. In addition, they said that prototyping also helps them make better estimates, plan incremental deliveries, validate technical feasibility for new capabilities, and lay groundwork for the real implementation.

Team B also explained that the quality attribute focus is very important to them saying, "A quality attribute focus enhances all those benefits of prototyping." They further explained that prototyping generates design ideas, but new idea generation is a secondary benefit. Prototyping is part of a "validate early and often" development philosophy. Vague or complex requirements, technology integrations, and architecture changes are important things teams need to validate. The Team B's prototyping with quality attributes practice is summarized in the following bullets and illustrated in Figure 3:

- **RL-1:** The product owner and architect agree that a prototype of a feature should be developed in order to get early feedback on the architecture's ability to meet quality attribute requirement (prototyping activities and conventional Scrum sprints are planned at the same time at the beginning of each release cycle).
- **RL-2:** The first prototype concept is developed on a separate branch of code (not the development branch) and is targeted for development in a future sprint.
- **RL-3:** The team walks the product owner and a subset of users through a prototype concept demonstration during the sprint user demo. Feedback on the prototype is gathered.
- **RL-4:** The team holds a post-user demo meeting to discuss feedback from the sprint user demo. If feedback has design implications, the team rapidly develops architectural trade-off options and provides them to the product owner.
- **RL-5:** The architect and product owner collaborate to select design options (as required) and changes are incorporated into the release plan. Steps RL-3 to RL-5 are repeated until all feedback is incorporated.
- **RL-6:** The product owner decides when all feedback has been adequately addressed and approves migration of the prototype concept into the development environment. If the prototyped code was developed on a separate branch, the prototype code is merged into the development branch. If the prototype concept is done in a separate tool or

environment, it is then implemented in the development environment.

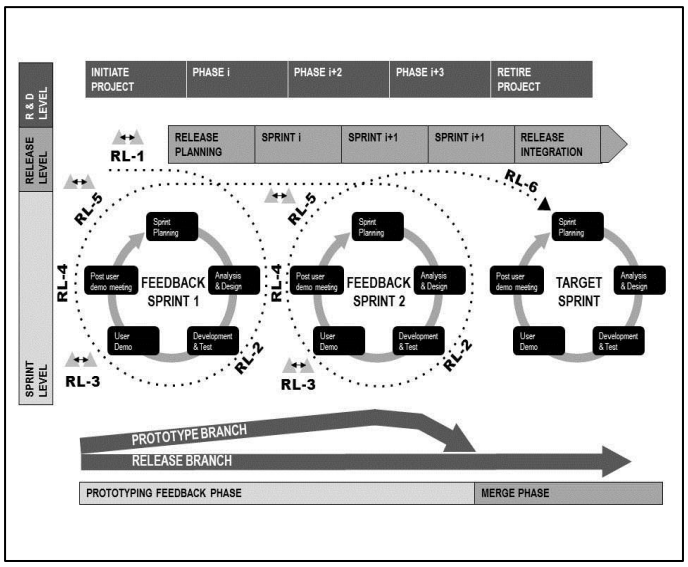

Figure 3: Release Level Prototyping Steps

The prototyping with a quality attribute focus practice and sprint feature development both leverage feedback points in the Scrum lifecycle. When described at this high level they may they seem very similar, but there they are separate and distinct. The details which clarify these differences are summarized in Figure 4, Prototyping Rules of the Road.

IV. Practice Analysis

In this section we present our analysis findings from the elaboration. For this practice elaboration, we conducted three phone interviews with Team B. The prototype lead was present at all three interviews. The first interview was a short call with the prototype lead focused to gather project context. The prototype lead and the chief architect were both present for the second interview which was more structured and recorded. The third interview was a short call with the prototype lead to gather more detail about the practice for this elaboration. This call was not recorded, but detailed notes were taken. We then review the data from all three interviews to derive these observations.

Feedback-driven weaving of architecture and requirements

We observed that Team B weaves architecture and requirements by fostering informal, but regularly occurring, collaboration between the architecture stakeholders (architect/team) and business representatives (product owner/users) as part of their Scrum management activities. This bringing together of the architecture and requirements sides allows the team to elaborate requirements earlier in the lifecycle avoiding surprises from unanticipated prototype feedback. Three places where this we observed that this occurs is: release planning, sprint user demo, and post-user demo feedback analysis. These three integration touch points represent small feedback loops (shown on Figure 3 with Twin Peaks symbol).

Integration at release planning. (Shown in Figure 3 at step RL-1)**.** The product owner and architect collaborate on key requirements and design for the initial prototype concept in the beginning of a release planning cycle. Trade-offs are discussed as required. Because prototyping and feature development resources are shared, the product owner weighs the value of the prototype changes against the value of other features in the release and determines which should move forward.

Integration at the sprint user demo (Figure 3, RL-3). During the sprint user demo the product owner and users share feedback on the prototype concept with the architect. During the user demo, the architect may also begin to ask users questions to try to get at unstated requirements gently probing for more information. The team explained how this probing works through example. The team was assigned to develop a prototype for a feature; however, no quality attribute requirements were included in the prototype concept description (user story). However, when the team demonstrated the prototype to the user the team said they got a "feeling" that the user didn't like it. So, the architect informally asked a few more questions (during and after the demo) until they identified an emerging performance requirement. By probing further and elaborating the requirement, the team was able to start working on a performance design improvement early avoiding unanticipated discovery of this requirement late in the lifecycle.

Integration at the post-user demo analysis. (Figure 3, RL-5) This is the integration point when the product owner and architect collaborate on design trade-offs that may result from prototype feedback. In these cases, there may be several design options and trade-offs that need to be considered (or there may be no design considerations). Working together, the architecture and requirements sides discuss options as required.

Overview of Prototyping Rules of the Road (for this team)

Team B described several key elements that define their prototyping practice shown as rules of the road in Figure 4.

Release Level Prototyping Rules of the Road	
Rule 1	Prototyping should be done at least a full sprint cycle before targeted feature development so there is time for at least one feedback cycle (never in the current development sprint cycle).
Rule 2	Prototyping work should <u>not</u> be done in the same branch of code or environment as where the current feature development is work.
Rule 3	Not all features need to be prototyped, but for those features that are determined to require prototyping should not be skipped. (We explore criteria for determining what to prototype in the Discussion section).
Rule 4	There is no separate "prototyping team"; the same team members that develop features develop feature prototypes.
Rule 5	The product owner prioritizes prototype development and feature development work at the same time during release planning. The product owner can stop prototype

	work at any time or trade off a current prototyping effort for development of new feature.
Rule 6	To the extent feasible, prototyping should be done in an environment technically similar to the target environment.
Rule 7	Prototyped features are usually demonstrated at the weekly user demo feedback sessions (these are the same user feedback sessions where developed features are demonstrated) to take advantage of scheduled access to the stakeholders.
Rule 8	Minimalistic prototyping is encouraged. Objectives to achieve validation of the concept to be prototyped (whether it be to validate a requirement or an architectural design) should be well defined and prototyping depth and breadth should be in accordance.
Rule 9	The product owner and a subset of users (subject matter experts) jointly provide feedback during prototype demonstrations.
Rule 10	Validation of critical requirements and design concepts is the focus of the prototyping practice, not generating new and novel design ideas

Figure 4: Release Level Prototyping "Rules of the Road" (from Team B)

The team gave an example to illustrate the importance of Rule 1 (prototype prior to the target sprint). They were pressed for time and decided to not start prototyping prior to the target sprint (for a feature that they said needed prototyping). Since the team started the prototype during target development sprint (not before as the team usually does), when the team received feedback there was no time to incorporate it. In this example the team also broke Rule 2, and did not prototype in a separate environment from the development environment. As a result they could not separate prototype-related changes from other development work and the whole release was delayed.

We observe that these rules are really guideposts, not hard-and-fast rules, and should be applied as appropriate. For example, Team B also explained that Rule 6 is encouraged but is not always feasible or cost effective. They explained that the decision to prototype in an environment that is technically similar to the development environment (or target environment) depends on a lot of things, particularly the focus of the prototype. For example, if the team is validating user interface requirements, they may want the prototype to visually be accurate so they need to use the actual tools for building that interface. They explained that the team also considers the cost (time, resources, etc.) involved in building a technically similar environment against the value derived from the prototype. In Rule 8 the Team B explained that the use of minimal prototypes is strongly encouraged and that prototyping should reflect the depth and breadth necessary to validate the desired requirement or concept. They suggest that detailed development and design that is not directly related to validating the prototype concept should not be part of the prototype concept development. This supports Royce's notion that unjustified early precision in requirements and planning are counterproductive giving the illusion of progress but leaving important areas gray [10].

Architecture-related factors to enable rapid response

We observed some other factors in Team B's examples that contribute to the effectiveness of the prototyping approach. We summarized these here.

Rapid architecture trade-off analysis. As prototype feedback is collected from users during a user demonstration (integration point RL-3), the team and architect must be prepared to quickly respond with architectural trade-off options. Team B suggested that the following items help enable rapid architectural analysis during prototyping for them:

- Knowledgeable, involved, and vocal architect
- Good understanding of how the system behaves
- "Key architecture documentation"

With respect to the last bullet, Team B explained that their project was lacking in "key architecture documentation". They said they would have been able to respond to prototype feedback more rapidly if they had key architectural views (or some type of representation). The Team A example also illustrates the importance of rapid trade-off analysis in the prototyping practice. Because Team A did not have the ability to rapidly re-evaluate design options there was additional delay. Rapid architecture trade-off analysis is shown in in the context of the practice execution in Figure 5.

Flexible architecture. Team B described their software product as "mature and flexible" explaining this allowed them to experiment more freely with prototype concepts. Perhaps we are seeing signs of the idea suggested by Royce that when projects have reached a mature state they can better balance their resource investments between defensive efforts (such as bug fixes, feature commitments, and schedule commitments) and offensive efforts (such as new integrations, new innovations, improved performance, earlier releases, and higher quality) [10]. Flexible architecture to support prototype experimentation is shown in in the context of the practice execution in Figure 5.

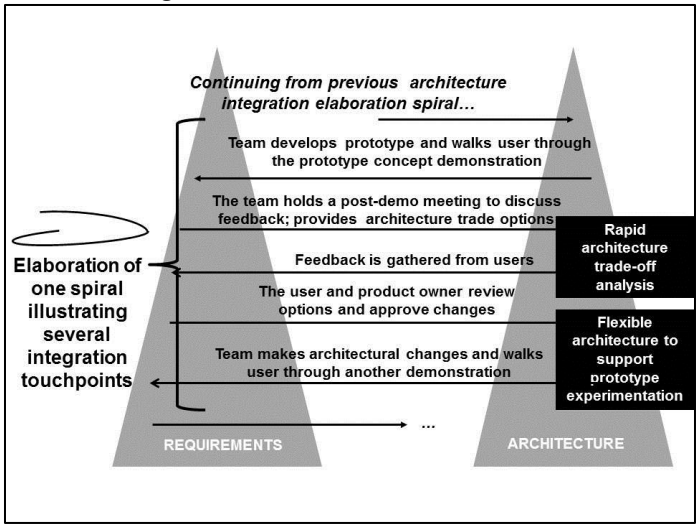

Figure 5: Example enablers for rapid architecture-side elaboration

V. Discussion

The team described another prototyping practice, *Research and Development (R&D) prototyping* practice. The R&D prototyping practice description starts at the portfolio level shown at the top of Figure 3. The first phase of the R&D prototyping practice is separate from the Scrum sprint development cycle. The second phase feeds into it. The team summarized some of the differentiating factors between R&D prototyping and release level prototyping as:

- The R&D prototype is typically funded by the organization, rather than the client.
- The work is done in an R&D environment using tools and hardware typically purchased by the organization (not the development project).
- R&D prototypes are often prototypes of infrastructure components or features that serve as foundational capability for multiple features or products.
- R&D prototype concept is not shared with the prototype concept until the team feels it is "ready to be shared".

Team B gave the following example to illustrate R&D prototyping. In the first phase, the Team B's organization decides to develop an R&D prototype of a server clustering capability (to enhance web-based system performance) with hopes that this capability could be offered to multiple clients. The clustering prototype is developed in the organization's R&D environment (not a project environment). There are several internal feedback discussions between the architect and the Team B's organizational business stakeholders reviewing the prototype. The R&D prototype is presented to the product owner when/if there is an appropriate opportunity do so. If the prototype is accepted by the product owner, the second phase of the R&D practice begins. At this point the R&D practice merges with the Scrum development cycle and follows steps **RL-1** through **RL-6** (Figure 3). For Team B, this generally means rewriting the code or installing and configuring tools into a project environment.

From the examples gathered from Team B, we observe three levels of weaving requirements and architecture: Scrum feature development level, release level prototyping, and R&D prototyping level. Traversing from bottom to the top of Figure 3, feature development level contains integration points at the user demo and post-user demo meetings. The release level weaves architecture and requirements when the product owner and architect come together during release planning. The R&D prototype level supports integration points at the portfolio level and also leverages integration points at the release/sprint levels (if the prototype is accepted by the product owner). The release level prototyping only looks forward a few sprints at a time and is generally for smaller feature prototyping efforts. R&D level prototyping looks further ahead than releases to consider prototypes that support the organization's product roadmaps and product portfolios. At the R&D prototyping level the team has the option to decouple the prototyping environment from the development environment.

There are several areas we would like to investigate further. Team B said that the mature nature of their software architecture was an enabler for rapid and effective prototyping. We would better understand what architecture structures enabled release and R&D level prototyping. We would also be interested in learning more about the influence of business pressure the prototyping practice. Team A was still in the early stages of its software product life. We would like a better understanding of the relationship between project maturity and consideration of quality attributes in prototyping. Team B also noted that quality attribute focus is difficult to achieve in prototyping if projects don't define quality attributes well. They suggested that sample quality attribute requirements for enterprise systems could be useful and worthwhile to explore. Perhaps this suggests applicability of generic quality attribute scenarios for prototyping on iterative, incremental projects [11].

We also observed differing degrees of separation and cadence in the R&D prototype practice. This raises several questions for future investigation with respect to the parameters that influence successful weaving, as well as when it is appropriate to move from one level to the next:

- What are the criteria for determining what should be developed as a feature, prototyped at the release level, and prototyped at the R&D level? Could be business driven (need high level of requirements validation) or architecturally driven (need to validate architectural changes)?
- What are the appropriate time bounds for each level?
- What is the optimal size of prototyped efforts at each level?
- How is the prototyping effort measured? How are prototyping artifacts valued in terms of team productivity and product quality?
- How much prototyping is appropriate and when is it best utilized?
- Is the approach of focusing on high-risk prototyping (through skeletal development) over feature-driven prototyping counter to Scrum or complementary?

VI. Conclusion

Counter to the traditional practice of conducting formal and separate requirements and architecture reviews, we observe through this practice elaboration that natural integration points throughout the Scrum framework (such as sprint planning, demo, and retrospective, release planning meeting, user demo, and post-user demo feedback analysis) can provide opportunities for weaving architecture and requirements into the incremental development lifecycle. The natural rhythm of Scrum lifecycle provides a time-bound structured feedback to identify potential hidden requirements.

Team B's prototyping with quality attribute focus practice requires collaboration between architect and product owner which is not present in Scrum. Requirements analysis and prioritization are done by the product owner and architectural design is done by the development team in Scrum [12]. The problem with this approach is that no one really has the whole picture which can leave room for unwelcome surprises. For example, if important information, such as the performance requirement in the Team A example, is not discovered or

shared until late in the development lifecycle the project is likely to encounter unexpected delay when the discovery is made. In addition Scrum, being a project management framework, does not provide much guidance in terms of incorporating architecture practices into the development lifecycle.

This practice also sheds light on several aspects of the development effort that position the team to respond quickly and efficiently when prototype feedback suggests architectural change. The ideas suggested in the Team B's prototyping rules of the road as well as the suggested enablers for rapid trade-off analysis and flexible architecture may be provide useful insights for other projects that would like to leverage the benefits of prototyping with quality attribute focus.

This practice elaboration provides an example of how architecture practices was integrated into Scrum incremental development (for example, the weaving of probing style requirement elicitation as part of the user demo). The idea of integrating practices is beginning to gain traction in the Agile community. In a recent blog posting, Ken Schwaber described "Scrum And" as a path of continuous improvement in software development beyond the basic use of Scrum [13]. In the future, we would like to elaborate some of the other integrated practices listed in Figure 1 to see what other new insights can be gained.

ACKNOWLEDGMENT

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

This material has been approved for public release and unlimited distribution. DM-0000168

REFERENCES

[1] Director of Defense Research and Engineering, "Rapicapability fielding toolbox study," Final Report, March 2010. http://www.cogility.com/Documents/Rapid_Capability_Fielding-Public_Release.pdf

[2] M. Denne and J. Cleland-Huang, "Software by Numbers," Prentice Hall, 2003.

[3] M. Hotle, D. Norton, and N. Wilson, "The end of the waterfall as we know it." Gartner Research, August 20, 2012.

[4] S. Bellomo, I. Ozkaya, R. Nord, "A Study of Enabling Factors for Rapid Fielding, Combined Practices to Balance Tension between Speed and Stability " (ICSE Conference 2013)

[5] A. Martini, L. Pareto, and J. Bosch, "Enablers and inhibitors for speed with reuse," Proceedings of the 16th Software Product Line Conference, ACM, New York, v. 1, pp. 116-125, September 2012.

[6] F. Bachmann, R. L. Nord, and I. Ozkaya, "Architectural Tactics to support rapid and agile stability." CrossTalk: The Journal of Defense Software Engineering, Special Issue on Rapid and Agile Stability, May/June 2012.

[7] K.Beck et al., Agile Manifesto, http://agilemanifesto.org/

[8] Boehm B. A spiral model of software development and enhancement. IEEE Computer, May 1988, 21(5): 61{72.

[9] Hall, Jon G., et al. "Relating software requirements and architectures using problem frames." *Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on*. IEEE, 2002.

[10] W. Royce, "Measuring Agility and Architectural Integrity", International Journal of Software and Informatics, Volume 5, Issue 3, 2011

[11] L. Bass, P. Clements, R. Kazman, "Software Architecture in Practice, Third Edition." Addison-Wesley, October 5, 2012

[12] K. Schwaber and J. Sutherland, "Scrum guidebook," Scrum.org and Scrum Inc., 2011.

[13] K. Schwaber, (blog) "Telling it like it is," http://kenschwaber.wordpress.com/2012/04/05/Scrum-but replaced-by-Scrum-and/ , April 5 2012.