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—This experience report builds on an earlier study in which we interviewed eight project teams that were using iterative incremental lifecycles. In the study, we captured the practices the teams felt contributed to rapid delivery. We 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 the study, *prototyping with quality attribute focus*. We compared two experiences in prototyping focused on quality attribute considerations applied on Scrum projects. We observe through interviews that feature development and prototyping practice spans multiple levels: feature development/sprint, release planning, and portfolio planning. We also observe other factors including rapid trade-off analysis, flexible architecture, and adoption of a set of enabling prototyping guidelines. The analysis of the observations sheds light on several aspects of the practice that enable the team to respond quickly and efficiently when prototype feedback suggests architectural change.

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

I. Introduction

This experience report builds on an earlier study in which we interviewed eight project teams (government and commercial) that were using iterative incremental lifecycles. In the study, we captured the practices the teams felt contributed to rapid delivery. Several of the practices we captured combined Agile practices and architecture practices (we refer to these as integrated practices) [1]. A summary of the practices from our interviews is shown in Table 1, ordered from the most to least used. Integrated practices are shown bold.

We elaborate one of these integrated practices called *Prototyping with quality attribute focus.* We focus on this practice for two reasons. First, this practice was mentioned by every team we interviewed as among the most used practices. Second, prototyping to understand quality attribute properties can be quite difficult–much harder than for understanding functions–due to the complexities of creating the conditions for testing quality attribute scenarios (e.g., requiring a demanding test workload, a very large database, a particular security user interaction).

This practice integrates prototyping (often leveraged on projects to reduce uncertainty instead of developing lengthy requirements specification documents) [2] [3] and an architectural focus of quality attribute requirements. Prototyping to address uncertainty is often discussed in the Agile community. For example, spikes refer to prototyping activities that are time boxed and typically meant to be thrown away [4]. Tracer bullets are exploratory activities in which work is typically not thrown away [5].

TABLE 1: PRACTICES SUMMARY TABLE

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

In this paper, we draw observations from examples captured in interviews as well as from a follow-up interview centered specifically on prototyping with a quality attribute focus. Although the practice was recognized by all teams, we were interested in better understanding why some teams were more successful in applying the practice than others. We compare experiences from two teams, Team A and Team B (as we refer to them), that represent the two ends of the spectrum. Team A encounters significant disruption and delay while Team B has a smoother approach resulting in the continuous delivery of features. A summary of key observations from our comparison and analysis (focused in the requirement and architecture area) are summarized below:

- We observe that Team B weaves architecture and requirements during prototyping through collaboration between the architect and product owner at three points: release planning, user demo, and post-user demo feedback analysis. We suggest that reoccurring integration points may help Team B identify emergent quality attribute requirements and address them earlier.

- In addition, we captured two architecture practices that enable rapid response when prototype feedback suggests architectural change: rapid architecture trade-off analysis and development of flexible architecture. We suggest these factors contribute to Team B's ability to respond quickly and competently to prototype feedback.
- Finally, we observed that Team B's prototyping guidelines help the team avoid disruption and delay when they receive impactful prototype feedback. Particularly important guidelines, as per Team B, are prototyping prior to the target sprint and prototyping in a separate environment.

Our hope is that by exploring these integrated practices we can identify what makes them effective and gain insight into concepts which may prove to be generalizable upon further study.

II. Practice Description

We begin with an example of a prototyping practice from Team A, responsible for developing a case management system. The project objective was to replace a legacy software system in use for many years. Team A was using the agile practice Scrum and had not yet released a production version of the software. Team A's initial prototyping practice did not have a quality attribute focus. They were mostly focused on prototyping business features to get feedback on look and feel. At the time we interviewed them, Team A was developing prototypes during the feature release cycle (as part of the sprint cycle) and in the same code branch as the rest of the code.

Due to business pressure, quality attribute aspects of the system were largely ignored by Team A in both development and prototyping efforts. Team A was giving a user demo of a prototype concept developed as part of their sprint release cycle when they received unexpected feedback that system performance was too slow. Several issues turned this feedback into big delays for the team. First, the problem was discovered late (during sprint release); there was not time in the schedule to address feedback-related changes. The product owner and architect did not routinely discuss demo feedback, so new features were given higher value by the product owner, and the performance problem was ignored for several sprints. When the team decided to rework the prototype concept, there was no separate environment to experiment in without impacting the development code. Finally, the team wasn't prepared to do rapid trade-off analysis to compare performance-related design options. So, rather than elaborating quality attribute requirements in a smooth spiral fashion, Team A experienced disruption and weeks of delay.

Team B also used Scrum and was developing a web-based analysis software system that had been in production and use for twelve years. They had organized software development into two-week sprints and six-to-twelve-month product releases. The example provided here describes Team B's release level prototyping practice. At this level the team looks forward a few sprints at a time and is generally prototyping smaller features. In the examples on which we based the steps below Team B generally followed a spike, however, Team B gave examples of creating both throw away and non-throwaway prototypes using this practice. Team B makes the whether to create a throw away prototype at release planning time. The example provided below is not a throw away prototype. The work in this example is done on a separate branch and later merged into the feature development branch. Team B's prototyping with quality attributes practice is summarized in the following bullets and illustrated in Fig. 1:

- **RL-1:** The product owner and architect agree that a prototype of a feature should be developed to get early feedback on the architecture's ability to meet quality attribute requirements (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-2 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, it may be merged into the development branch, or it may be recoded if the prototype is thrown away (Fig. 1 illustrates a merged prototype). If the prototype concept work is done in a separate tool or environment, it is then implemented in the development environment.

III. Practice Analysis

To elaborate their use of prototyping practice further, we conducted three phone interviews with Team B. The first and third interviews were teleconference calls with the team member responsible for developing prototypes on the team alone. During the second interview the chief architect was also present. The interview on which this paper was based was not recorded; however, detailed notes were taken. We reviewed the data from all three interviews to derive these observations.

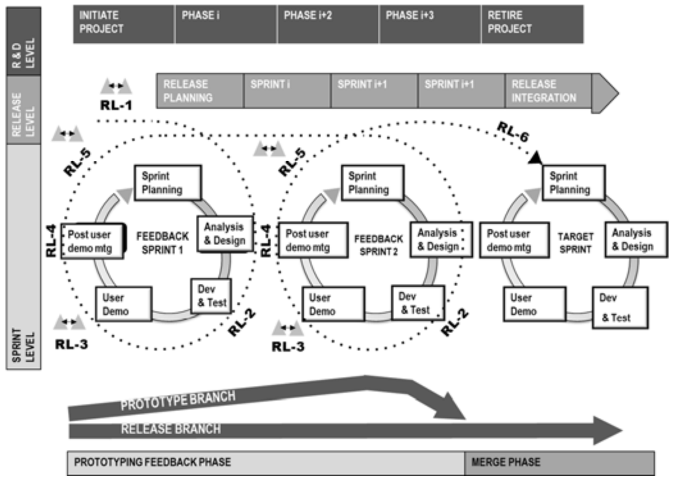

Fig. 1: Release-level prototyping steps.

A. Feedback-Driven Weaving of Architecture and Requirements

The key aspect of the practice that we observed from interviews with Team B is a weaving of architecture and requirements into their Scrum lifecycle through an informal, regularly occurring, collaboration between the architecture stakeholders (architect/team) and business representatives (product owner/users). Bringing together the architecture and requirements stakeholders allows the team to elaborate requirements earlier in the lifecycle, avoiding late surprises from unanticipated prototype feedback. Three integration points we observed were: release planning, sprint user demo, and post-user demo feedback analysis. These three integration touch points represent small feedback loops (shown in Fig. 1 with Twin Peaks symbol).

Integration at release planning. (Fig. 1, 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 (Fig. 1, 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 get at unstated requirements, gently probing for more information. The team explained how this probing works through an example. The team was assigned to develop a prototype for a feature; however, no quality attribute requirements were included in the prototype concept description (described as a user story). When the team demonstrated the prototype to the user, the team said they got a "feeling" that the user didn't like it. At that point, the architect informally asked a few more questions during and after the demo until the team 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. (Fig. 1, RL-5) 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 stakeholders discuss options as required.

B. Overview of Prototyping Guidelines

Table 2 summarizes some of the guidelines Team B follows when they develop prototype concepts.

The team gave an example to illustrate the importance of guideline 1 (prototype prior to the target sprint). In this example, the team determined that a feature needed to be prototyped. Since the team was pressed for time they started the prototype during the target development sprint; not before as the team usually does. When the team received feedback, there was no time to incorporate the feedback into the prototype concept. In this example the team also acted against guideline 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. Consequently, the whole release was delayed.

Team B explained these concepts are guideposts, not hard-and-fast rules; therefore, discretion should be used when applying them. For example, Team B also explained that guideline 6 is encouraged, but is not always feasible or cost

TABLE 2: SUMMARY OF PROTOTYPING GUIDELINES

Summary of Team B Prototyping Practice Guidelines	
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 sprint cycle).
2	Prototyping work should not be done in the same branch of code or environment as the current feature development work.
3	Not all features need to be prototyped, but features determined to require prototyping should not be skipped.
4	There is no separate "prototyping team"; the same team members that develop features develop feature prototypes.
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 a new feature.
6	To the extent feasible, prototyping should be done in an environment technically similar to the target environment.
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.
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.
9	The product owner and a subset of users (subject matter experts) jointly provide feedback during prototype demonstrations.
10	The focus of the practice is the validation of critical requirements and design concepts, not the generation of new design ideas.

effective. They explained that the decision to prototype in an environment technically similar to the development environment (or target environment) depends on many factors, particularly the focus of the prototype. For example, if the team is validating user interface requirements, they may want the prototype to be visually accurate so they need to use the actual tools for building that interface. Team B 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. Regarding guideline 8, 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 not directly related to validating the prototype concept should be avoided. This supports Royce's thought that unjustified early precision in requirements and planning are counterproductive, giving the illusion of progress but leaving important areas gray [6].

C. Architecture-Related Factors to Enable Rapid Response

We observed that using rapid architecture trade-off analysis and designing a flexible architecture for incorporating changes also enabled Team B to prototype effectively. We provide an example flow of activities that incorporates these enabling factors in Fig. 2, using the Twin Peaks model as a backdrop [7].

Rapid architecture trade-off analysis. As prototype feedback is collected from users during a user demonstration (integration point RL-3, Fig. 1), the team and architect must be prepared to quickly respond with architectural trade-off options. Rapid architecture trade-off analysis is shown in in the context of the practice execution in Fig. 2. Team B suggested that the following items help enable rapid architectural analysis during prototyping:

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

With respect to key architecture documentation, Team B explained that they would have been able to respond to prototype feedback more rapidly if they had had representative

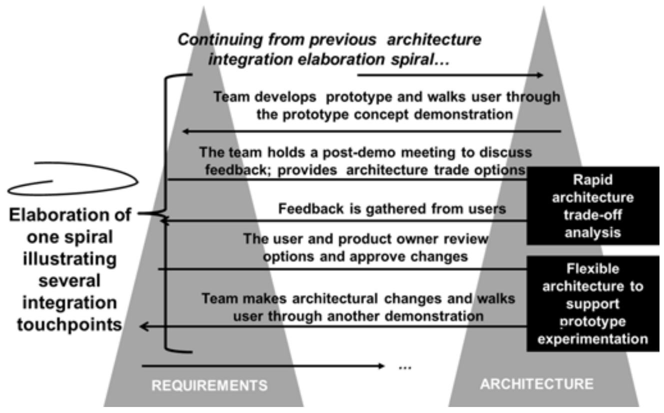
Continuing from previous architecture integration elaboration spiral...
Team develops prototype and walks user through the prototype concept demonstration
The team holds a post-demo meeting to discuss feedback; provides architecture trade options
Feedback is gathered from users
The user and product owner review options and approve changes
Team makes architectural changes and walks user through another demonstration
...
Elaboration of one spiral illustrating several integration touchpoints
Rapid architecture trade-off analysis
Flexible architecture to support prototype experimentation
REQUIREMENTS
ARCHITECTURE

Figure 2: Example enablers for rapid architecture elaboration.

architectural views. 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.

Flexible architecture. Team B described their software product as "mature and flexible", explaining that 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) [6]. Flexible architecture to support prototype experimentation is shown in in the context of the practice execution in Fig. 2.

D. Summary of Comparison

Table 3 provides a brief comparison between Team A and Team B prototyping practices. We observed that one key difference is the types of prototyping the teams have adopted. Leffingwell breaks spike type prototyping into two categories: technical spikes and functional spikes. Technical spikes are used to research technical approaches in the solution domain. Functional spikes are used whenever there is significant uncertainty as to how a user might interact with the system (e.g., mock-ups, wireframes, page flows) [4]. We note that Team A followed a functional spike prototyping approach only when they began the project but after poor system performance during several user demos, they moved more toward a quality attribute focus prototyping approach. Team B described prototyping examples supporting both functional and technical type spikes (technical spikes are more focused on exploring technical risk) [4].

TABLE 3: PROTOTYPING PRACTICES COMPARISON

		Team A	Team B
Requirements and architecture integration	Integration release planning	Y	Y
	Integration at the user demo	Y	Y
	Integration at the post-user demo analysis	N	Y
Architecture-related factors	Rapid architecture trade-off	N	Y
	Flexible architecture	N	Y
Prototyping guidelines from Team B	1: Prior to target sprint	N	Y
	2: Separate branch	N	Y
	3: Not all prototyped	Y	Y
	4: Same team	Y	Y
	5: Product owner prioritizes	Y	Y
	6: Technically similar	Y	Y
	7: Prototype feedback	Y	Y
	8: Minimalistic	Y	Y
	9: Combined feedback	Y	Y
	10: Concept validation	Y	Y

We also note differences in addressing feedback, architecture practices, and prototyping guidelines. We observe that Team B weaves architecture and requirements at these points: 1) release planning, 2) sprint user demo, and 3) post-user demo analysis. Team A leverages the first two integration points in their prototyping practice, but not the third. Based upon Team B's practice description and examples, we observe that the weaving of these integration points into the Scrum development framework enables the Team B architect to collaborate with the product owner regarding the architectural implications of prototype feedback. This helps Team B architect to identity emergent quality attribute requirements and to address them earlier.

With respect to architecture related factors we observe that, unlike Team B, Team A did not have rapid architecture trade-off analysis and flexible architecture. This resulted in additional delay when quality attribute-related requirements were discovered during a prototype demo because the architecture did not support incorporating changes with ease.

With respect to comparing the teams against Team B's prototyping guidelines, we found that the two areas where Team A and Team B diverged: 1) Prototyping prior to the target sprint, and 2) Prototype work is done in a separate environment. Examples provided by Team A suggest these may have contributed disruption and delay when performance issues were identified with a prototype concept.

IV. Research & Development Prototyping

In Section 2, we described the release level prototyping in which the team looks forward a few sprints at a time and is generally prototyping smaller features. In this section, we describe R&D level prototyping, also described by Team B, which looks further ahead to consider prototypes that support the organization's product roadmaps and product portfolios (R&D prototyping shown near top of Table 2). The team summarized some of the differentiating factors between R&D prototyping and release level prototyping:

- R&D prototypes are typically funded by the organization, rather than the client. R&D prototype 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 prototypes are not shared with the product owner until the organization determines they are "ready to be shared".

At the R&D prototyping level the team has the option to decouple the prototyping environment from the development environment. We use an example shared by Team B to illustrate R&D prototyping. In the first phase of the R&D prototyping, the organization decides to develop an R&D prototype of a server clustering capability (to enhance web-based system performance). It is hoped this capability may be offered to multiple clients. For the first phase of the prototyping effort, the prototype is developed in the organization's R&D environment. There are several internal feedback discussions between the architect and the organizational business stakeholders reviewing the prototype. If the prototyping effort is successful and a decision is made to share it, the R&D prototype is presented to the product owner. If the prototype is accepted by the product owner, the second phase of the R&D practice begins. In this second phase (post-product owner acceptance), the R&D prototype practice follows steps RL-1 through RL-6 (Section 2). Team B explained that at the R&D level they are generally following a Spike throw-away approach so if the prototype is accepted it will need to be rewritten "from scratch" to incorporate the work into the development environment.

V. Discussion

With Scrum, requirements analysis/prioritization are done independently by the product owner; architectural design is done independently by the development team [8]. The problem with this approach is that no one really has the whole picture when it comes to understanding the impact of feedback on the architecture, which can lead to unwelcome surprises. The natural rhythm of the Scrum lifecycle provides a time-bound, structured, yet less formal, way to identify potential hidden requirements and analyze them as an alternative to traditional separate requirements and architecture activities (such as architecture reviews). Based upon examples gathered from Team B, we observe three levels of weaving requirements and architecture that enable the architect and product owner to collaborate and together understand the whole picture. Traversing from the bottom to the top of Fig. 1, the Sprint level contains integration points at the user demo and post-user demo meetings. The release level integrates architecture and requirements when the product owner and architect come together for release planning. The R&D prototype level supports integration points at the portfolio level and release/sprint levels (if the prototype is accepted by the product owner).

There are several areas we would like to investigate further. Team B said that the mature nature of their software architecture enables rapid and effective prototyping for their team. We would like to better understand which architecture structures specifically enabled prototyping efforts for them. We are also interested in learning more about the influence of business pressure on the prototyping practice. Team A was still in the early stages of its software product life. We would like to better understand 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. The team suggests that example 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 development projects [9].

We observed differing degrees of separation and cadence in the feature development and prototype practices. 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 in the R&D prototyping process:

- Are the levels described through Team B's prototyping the right levels?
- 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? To what extent does business (need high level of requirements validation) or architecture (need to validate architectural changes) influence the decision?
- 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?

VI. Conclusion

In this paper, we compared two experiences in prototyping focused on quality attribute considerations applied on Scrum projects. The analysis of the practice sheds light on several aspects of the development effort that position Team B to respond quickly and efficiently when prototype feedback suggests architectural change. Ideas suggested by Team B that provide the framework for understanding factors that contribute to the successful application of prototyping with quality attribute focus include requirements and architecture integration points (integrated into the Scrum lifecycle), rapid trade-off analysis, flexible architecture, and adoption of some key prototyping guidelines (as per Team B).

The practice we elaborate here is one of the several integrated practices found in our study. 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 [10]. In the future, we would like to elaborate some of the other integrated practices to see if other 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] 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)

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

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

[4] D. Leffingwell. "Agile Software Requirements: Lean Requirements Practices for Teams, Programs and the Enterprise". Addison Wesley, 2011.

[5] B. Venners, "Tracer Bullets and Prototypes A Conversation with D. Hunt and D. Thomas, Part VIII", http://www.artima.com/intv/tracerP.html, April 21, 2003

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

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

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

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

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