Architecting for Large Scale Agile Software Development: A Risk-Driven Approach

Ipek Ozkaya, SEI
Michael Gagliardi, SEI
Robert L. Nord, SEI

Abstract. In this paper, we present lessons we learned while working with a large program, helping it to modernize its very large transaction-based system that operates 24x7, while adopting agile software development. We focus on two agile architecting methods we used that provide rapid feedback on the state of agile team support: architecture-centric risk factors for adoption of agile development at scale and incremental architecture evaluations.

Introduction

Over the past decade of their use, applying agile development methods to large-scale projects has brought its challenges [1, 2]. Challenges are exacerbated when organizations must deal with increased size of software and increased complexity in orchestrating large engineering and development teams, and when they have to ensure that the systems developed will be viable in the market for several decades. Understanding and systematically resolving the challenges becomes even harder for organizations that must adapt their existing processes to agile development (e.g., adapting the DoD acquisition lifecycle or switching from a lengthy development cycle based on other methods to an agile development cycle).

In this paper, we present lessons we learned while working with a large program, helping it to modernize its very large transaction-based system that operates 24x7, while adopting agile software development. We focus on two agile architecting methods we used that provide rapid feedback on the state of agile team support: architecture-centric risk factors for adoption of agile development at scale and incremental architecture evaluations.

Agile project teams recognize that there is a desired software development state that enables them to quickly deliver releases that provide stakeholder value [3]. This involves getting platforms and frameworks, as well as supporting tool environments, practices, processes, and team structures in place to support efficient and sustainable development of features. Conducting a review of architecture-centric risk factors for adoption of agile development at scale uses architecture- centric criteria to examine the organization and project context. The review serves to identify and mitigate key risks to achieving a desired software development state, when there is a need to use agile development and architecture-centric practices in concert. A common adoption risk is losing the focus on architecting activities that help maintain the desired state, enable cost savings, and ensure delivery tempo when other agile adoption activities take center stage [4]. Conducting incremental architecture evaluations that can be incorporated into agile development sprints/iterations assists in mitigating such a risk.

Organizations that must design, develop, deploy, and sustain systems for several decades and manage system and software engineering challenges simultaneously can dispense with neither agility nor attention to enduring design.

What is Scale and Why Manage It?

The amount of software in software-reliant systems has increased tremendously; the software has become more complex, and additional orchestration is needed between the teams that build, integrate, and test the software components. Understanding what is meant by large scale is important, as often several different challenges may be implied that must be teased apart. We investigate scale from three perspectives: scope, team, and time.

Large-scale systems often are large in scope in terms of the amount of new technology being introduced and the number of the features being added, independent components or COTS tools being integrated, users and user types to be accommodated, external systems with which the system communicates, configurations of components that can be configured for deployment, and so on. One or more of these aspects of scope may be present. As scope increases, the team and time dimensions are likely to increase as well.

The team dimension of scale is typically the most often-addressed aspect of scale in agile software development. Practices such as Scrum of Scrums are meant to address orchestration of multiple development teams in concert. There are often teams within or across organizations that are external to the core product development team (e.g., quality assurance, system integration lab, project management, and marketing) that must collaborate and provide input to product development. This brings additional challenges: Careful orchestration is necessary where these teams must seamlessly come up to speed with agile development and collaboration across organizational and lifecycle (e.g., system engineering, assurance) boundaries.

The time dimension of scale relates to both the duration of development and lifecycle time of the system. The system will need to be in development and operation for a longer period than systems to which agile development is typically applied, requiring attention to future changes and possible redesigns, as well as to maintaining several delivered versions. Over time, technology (hardware, software, sensors, effectors, etc.), threats, and features will change in various ways. In response to technology and threat changes, the system will undergo planned or unplanned upgrades. In addition, different planning rhythms may need to be kept in sync, which includes lifecycle budgeting and planning, individual milestone planning, and sprint planning.

Scaling agile projects in any of these dimensions involves a relationship to the architecture of the developed system and the use of architecture practices. For example, when there are multiple teams, the existence of some amount of explicit architecture documentation becomes important to coordinate the work across teams. Or, when a system exists for decades, a focus on the architecture of the system becomes important to ease the evolution of the system over time.

Architecture-centric Risk Factors for Adoption of Agile Development at Scale

When one or more of the scope, team, or time factors are of critical importance, incorporating architecture practices into agile development must take high priority, especially if an organization is new to adopting agile development practices.

In our recent engagement with a large organization, the key dimensions of scale were defined this way: the system has (1) gone through and would go through several technology upgrades, (2) served a large user base, and (3) been in operation and expected to be so for several decades. The organization faced the challenge of moving away from its existing phase-based software development approach to adopting agile development practices, while dealing with several dimensions of scale.

The first step was to conduct a risk analysis that combined a focus on architecture and agile practices. We conducted the risk analysis through several interviews and a scenario walk-through meeting, in addition to examining the working documents of organization-wide adoption plans. We conducted interviews with technical team members as well as managers, where they were in the same sessions as well as in separate sessions. The scenario walk-through meeting presented the team members with possible adoption scenarios and allowed them to analyze their possible outcomes within the organization.

The key areas we investigated included the following: business/acquisition and organizational climate, technology environment, ability to respond to change, project/team support, attention to quality attributes and architecture, customer collaboration, and productivity measures. Here we identify some common risks and factors worthy of attention in each of these areas.

Business/acquisition climate: Without clear identification of business and mission goals that reflect stakeholder concerns and success factors and strategies, adoption of agile practices at scale will entail resolving added challenges. Government organizations must pay special attention to contracting mechanisms. While all mechanisms potentially can be used in software vendor relationships, contracts should not be a barrier to building knowledge within a consistent team and improving communication [5].

Organizational climate: Adoption of agile development practices will require educating the teams about new practices that may not be familiar to a hierarchical organization with phase-based practices and high regulation checkpoints. Prior history of the waterfall processes and arms-length relationships among stakeholders, developers, and acquirers will prolong the time it takes to adopt agile development. These risk factors can affect both decision making in general and technical progress. Such impacts may occur, for example, when organizational barriers make it difficult or impossible to convene committees for technical, architectural, or design reviews.

Technology environment refers to having a robust development and design infrastructure in place to support the development teams. Automated testing and continuous integration practices require ongoing attention to support agile development adequately. To take advantage of agile development practices, such as nightly builds and configuration control, the right infrastructure should be in place and the necessary training provided. Such a technology environment, with automated tests, nightly builds, and configuration control mechanisms, is also tightly related to architectural requirements and the rate of change requests that the system must support. This includes requirements and requests involving new features, but also those that might necessitate architectural changes.

Response to requirements change: The inability to get a handle on the requirements management process in a volatile and large-scale product environment will hinder the success of the project. A special point of caution is to avoid focusing the requirements view on functional requirements; architecturally significant requirements must be continuously addressed as well.

Project/team support highlights attention to granting the authority to the downstream teams and arming them with the necessary skills and knowledge to succeed in an agile context while paying attention to the long-term goals of the system. New roles typically assigned when applying agile methods (such as product owner, Scrum master) have differing responsibilities from the roles in the existing phase-based waterfall program structures. Such differences may necessitate educating not only teams but also management and the customer.

Quality attributes emphasize the architecturally significant requirements of the system. An effort in architecture-focused acquisition can bring forward the key, architecturally significant concerns, such as integration and security. Often at-scale, multiple systems must be orchestrated, which requires continuous management of the key architecturally significant requirements, in addition to development of features.

Architecting activities must be integrated into agile development. The view that keeps these separate often creates silos, architecture conformance issues, and unexpected rework costs in later stages of the development effort. Architectural decisions, however, also must be made with consideration to the goal of iterative development.

Customer collaboration is key to success in any development effort. In an agile development context, customer collaboration is an essential part of the activities, for example, in sprint planning with Scrum. Communication with both internal and external stakeholders must be open and documentation should not be used as a substitute for communication.

Productivity measures in an agile context must incorporate a working system as well as continuous planning. If Scrum is chosen as the project management paradigm for agile development, understanding of relative estimation versus absolute estimation will be necessary. Obtaining the measures requires continuous monitoring and improving of estimates based on lessons learned. Providing working demos to the customer and

establishing done criteria as potentially shippable features must be incorporated into the iteration and release planning cycles. These cycles also require architecting activities to be seamlessly integrated into the sprints.

Figure 1 shows a risk profile example. Having such a risk profile, along with an understanding of the factors contributing to the risks, allows an organization to set priorities in adopting agile practices at scale. This figure also demonstrates a common profile that organizations might face in adopting agile practices at scale. The areas that commonly demonstrate high risks tend to be organizational climate, response to requirements change, and attention to quality attributes and architecture. While business/acquisition context, productivity measures, and project/team support also have issues related to adoption at scale, the risks associated with them would partially resolve if attention were given to the high-risk areas. Customer collaboration and technology environment areas follow as low-risk areas.

Figure 1: Example of risk profile summary

The organization we worked with chose to adopt Scrum as a project management methodology and educate its teams and management in agile development, in addition to keeping a significant focus on architecture.

The typical high-risk nature of architecture and quality attribute areas, as shown in Figure 1, were also issues. In order to ensure that key architectural evolution decisions, risks, and high-priority quality attributes were managed in concert with agile development, we conducted incremental architecture evaluations with the organization, which we describe next.

Incremental Architecture Evaluations

The goal of focusing on architectural evaluation as part of the modernization and agile adoption activities was (1) to identify architectural risks and risk themes of the "as-is" system as well as in migrating the "as-is" system to the target architecture, and (2) to provide actionable recommendations to address the risks themes. The evaluation method was based heavily on the Architecture Tradeoff Analysis Method® (ATAM®) and its principles, but due to constraints, was conducted incrementally [6].

The constraints on the evaluation were twofold. First, the availability of stakeholders and the architects' time to participate in the evaluations was limited to a small number of hours per week. The architects were available for eight hours per week, total. Specific stakeholder's availability was limited to fewer than four hours per week per stakeholder. Second, the documentation for software architecture was inadequate to perform the architecture evaluation per ATAM's criteria. However, we decided to proceed with the evaluation based on the architects' knowledge and to use whatever relevant, useful documentation was available, acknowledging the associated risks of proceeding with the evaluation. This approach enabled us to test an evaluation approach that could also be incorporated seamlessly with other development activities in agile development planning, for example, Scrum sprint and release activities.

Architecture Evaluation Kick-off

It can be challenging to identify the architectural mismatches and impediments to migration from the "as-is" system to the target architecture. This task becomes much more difficult when architecture documentation is lacking.

The evaluation began with a kick-off meeting with the evaluation team, the program office, and the architects. The evaluation team presented the evaluation approach, the program office presented the business drivers, and the architects presented the architecture overview. Despite insufficient architecture documentation, the participants decided to proceed with the evaluation, using the architects' knowledge and any acceptable, relevant documentation that was available. During the architecture evaluation sessions, the scribe would capture any undocumented architecture approaches in the analysis template as the architects described them.

The participants agreed that there would be a series of incremental architecture evaluation sessions on a weekly basis. A notional schedule was developed, starting with mission thread and scenario generation, to be finished within two weeks. Once the mission threads and all of the high-priority scenarios were identified and agreed upon, the schedule for the weekly architecture evaluation sessions would be developed and vetted. The participants also agreed that the evaluation schedule would be flexible and adaptable based on the evaluation progress made on a weekly basis.

Mission Thread and Scenario Generation

The operational end-to-end mission threads were developed by meeting with the operational system manager and technical lead and documenting the most critical threads, using the Mission Thread template for end-to-end mission threads. Table 1 shows a mission thread example, where Table 1(a) shows a summary of the mission thread, including the summary description, and Table 1(b) decomposes the thread into possible steps, with engineering considerations that impact system and software architecture decisions. Table 1(c) specifies the over-arching quality attributes for the mission thread, with engineering considerations that impact the system and software architecture decisions.

Four unique operational mission threads were identified and developed with the operational manager and technical lead and vetted with the system stakeholders. The mission threads established the end-to-end operational context for the evaluation and identified end-to-end quality attribute considerations to be evaluated. This was accomplished in a series of three two-hour meetings with the evaluation team, operational manager, and technical lead in the first week of the evaluation.

Name	Long-term customer order placement with supervisor override
Vignette (Summary Description)	Multi-channel order placement with outsourced fulfillment. The primary business activities are taking orders for products, fulfilling orders, and providing customer service (for example, order changes, return authorization) Orders can be placed over the internet or by calling a customer service agent. Payment processing for credit card payments is handled by a third party (Intuit, ChasePaymentech, etc.). If the payment processing service is not available, the call center agent accepts credit card payments without authorization, and incur the fraud risk. If the service is down for more than 8 hours, the agent accepts credit card information but hold sending the order to the fulfillment service until the payment processing service is restored and payments can be authorized. Order fulfillment is outsourced to a partner, who handles inventory management, order assembly and packing, and shipping and tracking (Amazon, Webgistix, Archway, etc.). There is a service level agreement (SLA) that all orders placed before 4:00pm Eastern Time will be shipped the same day. Our visibility into their process is limited to tracking the shipment on the shipper's web service using the tracking number.
Nodes Actors	Customer Call Center Agent Call Center Supervisor Systems shown in Architecture Overview
Assumptions	1. Customer A is a long-time customer of the company, with high lifetime value. a. What does this mean exactly? More considerations for timeliness, coupons, and so on, need to be analyzed. b. What about high value business customer B? Needs more analysis regarding integration into our system. c. What about family / household value? 2. Shipping charges are computed and applied automatically, but a Supervisor can override this for a particular order. 3. Our visibility into their process is limited to tracking the shipment on the shipper's web service using the tracking number.

Table 1(a): Partially filled mission thread example

Mission Steps	Description	Engineering Considerations, Issues, Challenges
1	Customer A places an order on the web site. The order value qualifies for free shipping.	
2	Payment authorization is obtained for the order. (<5 sec)	
3	The order is recorded and sent for fulfillment.	
4	Customer A's customer history is updated.	
5	….	

Table 1(b): Mission thread steps elaboration

Quality Attribute	Engineering Considerations, Issues, Etc.
Call Center Routing Performance	
Call Center Screen Pop Performance	
Call Center Routing Accuracy	
Order Accuracy	
Time-to-market	
Modifiability	
Testability	
Availability	
Usability for Call Center Agents	
Migratability	
Scalabiility	

Table 1 (c): Quality attribute requirements related to the mission thread

The scenarios were generated in a series of small scenario-generation sessions with the architects and selected stakeholders. The ATAM utility tree was used to capture and elicit scenarios. Figure 2 shows a utility tree example. The utility tree was initially seeded with the quality attributes identified in the business driver's presentation and the operational mission threads. Some scenarios were derived from the mission threads. Each session would begin with a review of the prior session's results. Each session would then elicit quality attribute considerations and scenarios from the attending stakeholders, insert them into the utility tree and then rank each of the scenarios according to degree of difficulty (ranked by architect) and importance (ranked by management)—high, medium or low. The evaluation team, program office, and architects would then discuss the need for any additional scenario generation sessions. Once the utility tree was deemed to be finished, the evaluation team, architects, and program office met to develop an incremental evaluation schedule based on the high-priority scenarios and mission threads. They agreed that the mission threads would be the last evaluation sessions once the high-priority scenarios were all evaluated, in order to evaluate the architecture in an end-to-end manner.

Architecture Evaluation Sessions

There were four mission threads and 59 scenarios generated, based on a utility tree with nine quality attributes. The architecture evaluation sessions covered all of the highly ranked scenarios followed by the four mission threads. The evaluation sessions were grouped based on the quality attributes and the mission threads. The mission thread sessions were held last so that the end-to-end evaluation would follow the scenario-based analyses. The scenario-based evaluation sessions were roughly ordered based on what we considered the more critical quality attributes for the system (e.g., availability, performance, and maintainability).

The session participants decided that there would be three two-hour architecture evaluation sessions per week. The sessions would focus on specific, pre-determined scenarios, based on the quality attribute utility tree and mission threads (e.g., the first week of evaluation sessions would focus on the important availability scenarios, with intent to utilize an evaluation backlog flowing into the next week's session schedule, if necessary).

We expected to analyze two scenarios per session, since the documentation was poor.

Once we finished all the scenario analyses for a quality attribute, we would develop the schedule for the next quality attribute set of scenarios. We repeated this for the entire utility tree and then scheduled the end-to-end mission thread analysis sessions, expecting one thread per two-hour session. The entire set of evaluation sessions required 23 sessions over the course of eight calendar weeks. The evaluation team met to develop risk and non-risk themes after the scenario analyses of each quality attribute and mission thread.

Architecture Evaluation Results

The evaluation sessions resulted in the discovery of numerous, unique, architecture risks and non-risks. The number of risks found was within the typical number and quality of risks found in an architecture evaluation for a system of this type and size. The length of each evaluation session was extended due to the lack of architecture documentation and the need to document roughly the architectural approaches as they arose during the sessions.

The end-to-end mission thread analyses uncovered many risks that were not identified in the scenario-based evaluation sessions. These risks dealt with architectural decisions that supported end-to-end operational processing of data that is difficult to identify when examining parts of a system (as is typical of a scenario walkthrough).

Overall, the incremental architecture evaluation sessions allowed the team to work within the constraints of the engineering team. The schedule developed to conduct such an evaluation approach focused on priority of the scenarios and mission thread steps identified. The approach created tasks that were about two-to-three hours in length and resulted in concrete development and architecture artifacts. This particular evaluation was not incorporated with Scrum sprint activities because the organization had not completed the sprint planning yet. However, the definition of the tasks and the technical outputs created an example of how incremental evaluation could be incorporated into a Scrum development approach by balancing development tasks with a focus on architecture evaluations.

Takeaways

Embracing the principles of agile software development and software architecture provides improved visibility of project status and improved tactics for risk management.

There are different aspects of scale that are manageable with different approaches, such as scope, team, and time.

We see evidence that a systematic architecture-centric review of organizational and project factors, as this organization used, is essential for understanding risks and dealing with the challenges arising in large-scale software development.

We believe the incremental evaluation approach applied to this system could be beneficial in an agile development approach, where small, short architecture evaluation sessions could be implemented in agile sprints. The artifacts we exemplify here, such as mission threads and quality attribute utility trees, could be invaluable in both helping with backlog management and augmenting sprint planning.

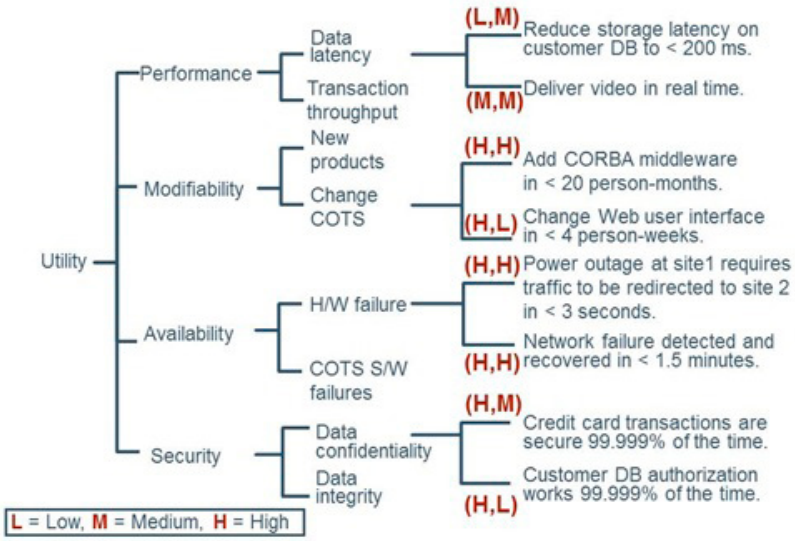


Figure 2: Utility tree example

Acknowledgements/Disclaimers:

Copyright 2013 Carnegie Mellon University

This material is based upon work funded and supported by the DoD 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.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

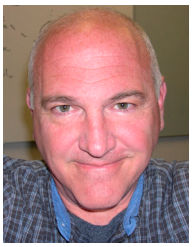
This material has been approved for public release and unlimited distribution. Architecture Tradeoff Analysis Method®, ATAM® are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM-0000094

ABOUT THE AUTHORS

Ipek Ozkaya is a senior member of the technical and works to develop empirical methods for improving software development efficiency and system evolution with a focus on software architecture practices, software economics, and requirements management. Her latest publications include multiple articles on these subjects focusing on agile architecting, dependency management, and architectural technical debt. Ozkaya serves on the advisory board of the IEEE Software magazine.

Software Engineering Institute
4500 Fifth Avenue, Pittsburgh, PA
Phone: 412-268-3551
Fax: 412-268-5758
E-mail: ozkaya@sei.cmu.edu



Michael Gagliardi has more than 25 years' experience in real-time, mission-critical software architecture and engineering activities on a variety of DoD systems. He currently works in the SEI Research, Technology, and System Solutions Program on the Architecture-Centric Engineering Initiative, and is leading the development of architecture evaluation and quality attribute specification methods for system and system-of-system architectures.

Robert L. Nord is a senior member of the technical staff at the SEI and works to develop and communicate effective methods and practices for software architecture. He is co-author of the practitioner-oriented books, Applied Software Architecture and Documenting Software Architectures: Views and Beyond and lectures on architecture-centric approaches.

REFERENCES

1. Grant, T. "Navigate the Future of Agile and Lean." Forrester, January 10, 2012.
2. Leffingwell, D. Scaling Software Agility. Upper Saddle River, NJ: Addison-Wesley, 2007.
3. Bachmann, F., Nord, R. L., and Ozkaya, I. "Architectural Tactics to Support Rapid and Agile Stability." Special Issue on Rapid and Agile Stability, CrossTalk, 25.3 (2012): 20-25.
4. Brown, N., Nord, R., and Ozkaya I. "Enabling Agility Through Architecture." CrossTalk 23.6 (2010): 12-17.
5. Lapham, M. A., Miller, S., Adams, L., Brown, N., Hackemack, B., Hammons, C., Levine, L., and Schenker, A. Agile Methods: Selected DoD Management and Acquisition Concerns, October 2011, Technical Note, CMU/SEI-2011-TN-002.
6. Bass, L., Clement, P., and Kazman, R. Software Architecture in Practice, 3rd ed. Addison Wesley, 2012.