Agile Architecting Methods for Large Scale Agile Software Development

Michael Gagliardi, Robert Nord, Ipek Ozkaya

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

STC, April 2013
Agenda

Dealing with scale
Example legacy system
Architecture-centric risk factors while adopting agile software development
Iterative-incremental evaluations using mission threads within Agile development
Take-aways
Challenges in Adopting Agile Software Development

Agile software development techniques have reduced time to system deployment.

Agile techniques, as often practiced, may not readily scale to government projects as a result of several misconceptions

- **Fallacy**: architecture will emerge as a result of numerous local refactorings
- **Fallacy**: requirements and design up front are *completely wasteful*
- **Fallacy**: Current government contracting mechanisms do not lend themselves to effective agile development
- **Fallacy**: We are doing agile, we do not need to document the requirements
Why Is Software Architecture Important?

The right architecture paves the way for system success. The wrong architecture usually spells some form of disaster.

Represents earliest design decisions
- hardest to change
- most critical to get right
- communication vehicle among stakeholders

*First* design artifact addressing
- performance
- modifiability
- reliability
- security

Key to systematic *reuse*
- transferable, reusable abstraction

Key to system *evolution*
- manage future uncertainty
- assure cost-effective agility
Today’s Challenge Dealing with Large Organizational Changes

Yesterday’s Agile: Teams got better at building software
- Velocity
- Reliability
- Code quality
- Improvement
- Cohesion

Today’s Agile
Moving the rest of the business
- Priorities are larger than the development team
- Collaboration is critical
- Timelines have changed

A Closer Look at Scale: Scope

- Is the project in a new domain or technology?
- Does the project have new requirements such as standards compliance, system testing, and integration lab environments?
- Is there a need to align systems engineering and software development activities?
A Closer Look at Scale: Team

- Are there multiple teams that need to interact, both internal and external to the organization?

- What are the dependencies between the work products of system and software engineers?

- Does the end-to-end delivery of features require resources from multiple teams?
A Closer Look at Scale: Time

- Does the work require different schedule constraints for releases?

- How long is the work product expected to be in service?

- How important are sustainability and evolution?
Agenda

Dealing with scale

Example legacy system

Architecture-centric risk factors while adopting agile software development

Iterative-incremental evaluations using mission threads within Agile development

Take-aways
Example Legacy System Overview

A legacy system was partially modernized

- Exists as a combination of Legacy (75%) and Modernized (25%)
- Exists in two data centers for recovery from catastrophic failure, and under-the-cover data mirroring (one active, one passive)

Modernized portion is:

- COTS based, relational database, service oriented, transactional, batch entries with hundreds of transactions, data warehousing, 24/7/365 operation
- Software is provisioned to processors at configuration time to form round robin load sharing
- Complex interactions between agency users and many participating government agencies and commercial users
- Specialized interfaces to the external users
- Business workflows are implemented using reliable messaging queues between business processes
**Architecture-centric Risk Factors in Adopting Agile Software Development**

<table>
<thead>
<tr>
<th>Business/Acquisition</th>
<th>Response to Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Culture</td>
<td>Team Support</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quality Attributes</th>
<th>Customer Collaboration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Productivity Measures</td>
</tr>
</tbody>
</table>

Understand symptoms by investigating Agile software development and software architecture principles and practices in the context of the product and the project.
Architecture-centric Risk Factors in Adopting Agile Software Development

Business/Acquisition

• Business and mission goals are clearly identified reflecting stakeholder concerns for both the short term and long term considerations of the system.

• Success strategies (e.g. roadmaps, product portfolios) defined and clearly communicated.

• Clear contracting mechanisms are established.

• Team and stakeholder communication barriers are cleared.
Response to change

- Dynamic environment and changing requirements are understood.
- Necessary technology and processes are identified to respond to change.
- Impact of uncertainty on the project is acknowledged.
- Waste is identified and tradeoffs managed (e.g., technical debt and defects).
Quality attributes

- The importance of quality attribute requirements is understood.
- Quality attribute requirements are defined and tied to business goals.
- Means for analysis of necessary quality attributes are in place and used to predict system properties.
- Measurement environment is in place to monitor the implemented system quality and “done” criteria.
Architecture-centric Risk Factors in Adopting Agile Software Development

Architecture

- Evidence is provided that the architecture satisfies quality attribute requirements.
- Appropriate functional requirements are assigned to architecture elements.
- Architectural issues (e.g., technical debt) are tracked and managed.
- Timeline of critical architectural decisions is clear and scheduled.
### Example Risk Profile

The organization accepted their risks and realized the big undertaking ahead.

- They decided to take small steps and started by focusing on architecture while getting up to speed on Scrum.

We continue to apply the risk factors analysis focusing on agile and architecture issues with both government and industry organizations.

<table>
<thead>
<tr>
<th></th>
<th>Risk profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business/Acquisition</td>
<td></td>
</tr>
<tr>
<td>Organizational climate</td>
<td></td>
</tr>
<tr>
<td>Technology environment</td>
<td></td>
</tr>
<tr>
<td>Response to requirements change</td>
<td></td>
</tr>
<tr>
<td>Project/team support</td>
<td></td>
</tr>
<tr>
<td>Quality attributes</td>
<td></td>
</tr>
<tr>
<td>Architecture</td>
<td></td>
</tr>
<tr>
<td>Customer collaboration</td>
<td></td>
</tr>
<tr>
<td>Productivity measures</td>
<td></td>
</tr>
</tbody>
</table>

- **Red** high risk
- **Yellow** medium risk
- **Green** low risk
Agenda

Dealing with scale
Example legacy system
Architecture-centric risk factors while adopting agile software development
Iterative-incremental evaluations using mission threads within Agile development
Take-aways
Example Legacy System Architecture Overview

Software architecture mostly undocumented

Data architecture documented in some areas

Original architects no longer on the job

Software maintained by developers, primarily using the software itself and some detailed design documents

Outstanding software problem reports take a long time to resolve

Limited weekly availability of architects for evaluation
Mission Threads and Scenarios Are Critical

We have used mission thread augmentations to develop system specific scenarios for legacy system architecture evaluation.

Legacy system specific scenarios are developed with the system of system, enterprise architecture, and legacy system stakeholders.

Scenarios are also derived from the Mission Thread augmentations that pertain to the specific legacy system
• From each Mission Thread step augmentation
• From each Quality Attribute augmentation

The mission threads and scenarios are used to evaluate the legacy system & software architecture.
Legacy System Architecture Evaluation – Approach

Based on Mission Thread Workshop and Architecture Trade-off Analysis Method concepts

Developed three end-end business threads, based on the business drivers and elicited from business and operations stakeholders. Representing the three major end-end capabilities of the legacy system.

Augmented the three end-end business threads:

- Augmenting the steps in the threads; eliciting any legacy system specific scenarios for each step. All quality attribute concerns generated scenarios.
- For the over-arching quality attributes, developed a legacy system specific utility tree; eliciting concerns and scenarios, using MTW templates.

Developed over 60 quality attribute scenarios elicited from the stakeholders.

All of the scenarios are taken together and prioritized (just as in Phase 1 of ATAM).
Legacy System Architecture Evaluation – Approach

Evaluation team makeup: 3 SEI evaluators with facilitator, 2 Subject Matter Experts from Program Office

Due to limited weekly availability of architects and lack of documentation, we decided to hold a series of architecture evaluation sessions. Three times a week, two hours per session.

Decided to focus evaluation sessions on specific topics, e.g., performance, availability, maintainability, etc.

When all the focused sessions were completed, we executed end-to-end thread analysis sessions.
Legacy System Architecture Evaluation - Results

The focused sessions resulted in identifying over 100 architectural risks, 25 non-risks.

The end-to-end thread evaluation sessions resulted in identifying 15 additional risks, mostly dealing with end-to-end issues.

Seven architecture risk themes were generated, along with impacts and recommendations. Customer was very satisfied with the results.
Legacy System Architecture Evaluation – Lessons Learned

The end-to-end business threads set the proper context for scenario generation for the specific legacy system

Executing the end-to-end thread sessions last was very beneficial:

- Put all the previously identified risks into context
- Helped to focus on end-to-end type risks
- Helped to understand (and document) the architecture end-to-end

Lack of a documented architecture was a burden

- Slowed down the evaluation sessions and extended the schedule. We quickly abandoned the notion of capturing architect’s hand drawings.
- Never sure if there were places in the architecture that needed further evaluation
Legacy System Architecture Evaluation – Lessons Learned

We were satisfied that we had covered the architecture when we finished the end-to-end thread sessions.

We needed 20 evaluation sessions (2 hours each), spanning six weeks. This is not out of line for the total amount of time needed for architects to support an architecture evaluation.

The architects were very cooperative and open about the process and provided good information, even though they weren’t the original architects.
A Closer Look at the Sprint Design-Dev-Test Cycles

Architecture-Centric Adaptation of the Program’s Agile Software Development Quick Reference Guide

Develop System Architecture

Prior to Release 1 Planning Cycle

Release 1 Planning

Sprint 0

Design-Dev-Test

SW Arch Dev

Sprint 1

Sprint 2

Sprint n

Release Integration (if Needed)

Release 2

Release X

System Architecture Description Document

Baseline System Architecture Description and Initial Software Architecture Specification

Input to Release Planning Cycle which will include Software Architecture Development

Business Drivers, Utility Tree, and QA Scenarios

Quality Attribute Workshop

Initial System Architectural Design

Initial System Architecture Evaluation

Initial System Architecture Readiness Review

System Arch Design

Initial Software Architecture Specification

Development System Architecture

Prior to Release 1 Planning Cycle

Release 1 Planning Cycle

Sprint 0

Design-Dev-Test

SW Arch Dev

Sprint 1

Sprint 2

Sprint n

Release Integration (if Needed)

Release 2

Release X

System Architecture Description Document

Baseline System Architecture Description and Initial Software Architecture Specification

Input to Release Planning Cycle which will include Software Architecture Development

Business Drivers, Utility Tree, and QA Scenarios

Quality Attribute Workshop

Initial System Architectural Design

Initial System Architecture Evaluation

Initial Software Architecture Specification

Development System Architecture
A Closer Look at the Sprint Design-Dev-Test Cycles

Architecture-Centric Adaptation of the Program’s Agile Software Development Quick Reference Guide

Develop Software Architecture

During Sprint 0

Release 1 Planning

Sprint 0

SW Arch Dev

Design-Dev-Test

Sprint 1

SRD

Sprint 2

SRD

Sprint n

SRD

Release Integration
(if Needed)

Release X

Baseline System Architecture Description and Initial Software Architecture Specification

Activities

Medium Weight and Limited Scope

Specifying Architecturally Significant Requirements

- Develop Initial Software Architecture
- Evaluate Architecture
- Analyze Architectural Risks
- Mitigate Architectural Risks

Artifacts

- Quality Attribute Scenarios
- Software Architecture Description
- Architecture Evaluation Results
- Risk Mitigation Approaches
- Updates to Architecture Description
A Closer Look at the Sprint Design-Dev-Test Cycles

Architecture-Centric Adaptation of the Program’s Agile Software Development Quick Reference Guide

Artifacts

• Release Backlog (RBK)
• Sprint Backlog (SBK) = Subset of Product Backlog (PBK) and Release Backlog (RBK) for a Sprint
• Sprint Review & Demo (SRD) Letter
• System Design Document (SDD)
• Security Artifacts (SR, SAR, POA&M)
• Burnup and Burndown Charts [to Fulfill Similar Function to Earned Value Management (EVM)]
• Updates to POP, TEMP

Activities

1st Iteration

- Iteration Planning
- Requirements Management
- Mock-ups/Use Cases/Simulations
- Product Backlog (PBK) and Release Backlog (RBK) for a Sprint
- Develop Test Cases and Scripts
- Compliance Bodies Participate as Defined in the TEM
- Evaluate Architecture Against Scenarios
- Specify New Quality Attribute Scenarios
- Architecture Evaluation Results
- Risk Mitigation Approaches
- Quality Attribute Scenarios
- Lightweight and Limited Scope
- IF Architectural Risks are Discovered:
  - Analyze Architectural Risks
  - Mitigate Architectural Risks
  - Document Architectural Changes

Baseline System Architecture Description

Initial Software Architecture Specification

Baseline Software Architecture Description
A Closer Look at the Sprint Design-Dev-Test Cycles

Architecture-Centric Adaptation of the Program’s Agile Software Development Quick Reference Guide

Artifacts

- Release Backlog (RBK)
- Sprint Backlog (SBK) = Subset of Product Backlog (PBK) and Release Backlog (RBK) for a Sprint
- Sprint Review & Demo (SRD) Letter
- System Design Document (SDD)
- Security Artifacts (SR, SAR, POA&M)
- Burnup and Burndown Charts [to Fulfill Similar Function to Earned Value Management (EVM)]
- Updates to POP, TEMP

Activities

- Iteration Planning
- Requirements Management
- Mock-ups/Use Cases/Simulations
- Develop Test Cases and Scripts
- Compliance Bodies Participate as Defined in the TEMP, TPA
- Update Progress Reports for Transparency
- Sprint Planning Review (SPR)
- Sprint Review & Demo (SRD)
- Final Review - Acceptance Test Readiness Review (ATRR)
- Architecture Risk Mitigation
- Specify New Quality Attribute Scenarios
- Evaluate Architecture Against Scenarios
- IF Architectural Risks are Discovered:
  - Analyze Architectural Risks
  - Mitigate Architectural Risks
  - Document Architectural Changes

Baseline System Architecture Description and Initial Software Architecture Specification

Baseline Software Architecture Description
A Closer Look at the Sprint Design-Dev-Test Cycles

Architecture-Centric Adaptation of the Program’s Agile Software Development Quick Reference Guide

Artifacts

- Release Backlog (RBK)
- Sprint Backlog (SBK) = Subset of Product Backlog (PBK) and Release Backlog (RBK) for a Sprint
- Sprint Review & Demo (SRD) Letter
- System Design Document (SDD)
- Security Artifacts (SR, SAR, POA&M)
- Burnup and Burndown Charts [to Fulfill Similar Function to Earned Value Management (EVM)]
- Updates to POP, TEMP

Activities

- Iteration Planning
- Requirements Management
- Mock-ups/Use Cases/Simulations
- Develop Test Cases and Scripts
- Compliance Bodies Participate as Defined in the TEMP, TPA
- Update Progress Reports for Transparency
- Sprint Planning Review (SPR)
- Sprint Review & Demo (SRD)
- Final Review - Acceptance Test Readiness Review (ATRR)

Nth Iteration

- Lightweight and Limited Scope
- Evaluate Architecture Against Scenarios
- Specify New Quality Attribute Scenarios
- IF Architectural Risks are Discovered:
  - Analyze Architectural Risks
  - Mitigate Architectural Risks
  - Document Architectural Changes

Baseline System Architecture Description

Initial Software Architecture Specification

Baseline Software Architecture Description
Agenda

Dealing with scale
Example legacy system
Architecture-centric risk factors while adopting agile software development
Iterative-incremental evaluations using mission threads within Agile development
Take-aways
Take-aways

There are different aspects of scale to be managed with different approaches, such as scope, team, and time.

Systematic root-cause analysis is essential for understanding risks arising in large-scale software development.

Embracing principles of Agile software development and software architecture provide improved visibility of project status and better tactics for risk management.

Incorporating short architecture review meetings into the planning and sprint activities can provide a mechanism to identify and mitigate architectural risks.
References


Contact Information

Michael Gagliardi, Robert Nord, Ipek Ozkaya
Research, Technology, and System Solutions Program
Architecture Practices Initiative
Email: {mjg, rn, ozkaya} @sei.cmu.edu

Software Engineering Institute
Customer Relations
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
USA