Agile Development and Software Architecture: Understanding Scale and Risk

Ipek Ozkaya

Research, Technology and Systems Solutions (RTSS) Program

Ozkaya is a senior member of the SEI technical staff within the Architecture-Centric Engineering (ACE) Initiative in the Research, Technology, and System Solutions (RTSS) Program. Her current interests and projects are in developing empirical methods for improving software development efficiency and system evolution with a focus on software architecture practices, software economics, and requirements management.
The challenge

Tradeoffs and their dependencies must be supported by both Agile software development and architecture practices
The challenge

First, more capabilities

Then, more infrastructure

First, more infrastructure

Then, more capabilities

need to monitor to gain insight into life-cycle efficiency

underestimated re-architecting costs

neglected cost of delay to market

Increased visibility into delivery

Focus on Priority

Focus on Integrated Value

Focus on Cost

Use metrics to monitor & select development tasks
Agenda

Symptoms of failure

Concepts of scale and root-cause analysis

Tactics that can help

• Align feature and system decomposition.
• Create an architectural runway.
• Use matrix teams and architecture.
Symptoms of failure

- Teams (e.g., Scrum teams, product development teams, component teams, feature teams) spend almost all of their time fixing defects, and new capability development is continuously slipping.

- Integration of products built by different teams reveals that incompatibility defects cause many failure conditions and lead to significant out-of-cycle rework in addition to end-to-end fault-tolerance failure.

- Progress toward meeting milestones is unsatisfactory.
Scope drivers

Fundamental project management concerns are essential to keep in mind:

• If the *schedule* needs to be shorter, you may see an increase in *cost* and a decrease in *scope*.

• If *cost* becomes an issue, you may see a decrease in *scope* or an increase in *schedule*.

• If *scope* is increased, you may see an increase in both *cost* and *schedule*.

**Traditional approach:**
- Fixed scope driving cost and schedule

**Agile project management approach:**
- Fixed cost and schedule driving scope
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, or does it simply have more features, elements, and relationships?

- 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?

- Have you considered the end-to-end success of features that may 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?
Polling question

Are you currently doing development in a large-scale context that can be captured by extended scope, team size, or timelines of scale?

1. Large team size
2. Larger than normal scope
3. Longer development roadmap
4. Product expected to be in service for a long time
5. At least two of the above
Root-cause analysis

Investigate both technical and nontechnical areas, looking at both Agile software development and software architecture fundamentals.
Root-cause analysis

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).
Root-cause analysis

Culture

• People are made available (internal and external), including an appropriate number of people who have the right skills and knowledge and clear responsibilities.
• Team members are motivated and empowered by many degrees of freedom.
• Clear communication among teams and team members is established.
• There is high-level management support.
Root-cause analysis

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.
Root-cause analysis

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.
Tactics to consider

Align feature and system decomposition.
Create an architectural runway.
Use matrix teams and architecture.
Align feature and system decomposition

| Dependencies between stories & supporting architectural elements | Understanding the dependencies between stories and architectural elements enables staged implementation of technical infrastructure in support of achieving stakeholder value. |
| Dependencies among architectural elements | Low-dependency architectures are a critical enabler for scaling up Agile development.¹ |
| Dependencies among stories | High-value stories may require the implementation of lower value stories as precursors.² |


Align feature and system decomposition

Tension between high-priority features (vertical decomposition) versus common reusable services (horizontal decomposition)
Align feature and system decomposition
Two examples

Decouple teams and architecture to ensure parallel progress as the number of teams increases.
Create an architectural runway

The architectural runway provides the degree of architectural stability to support the next $n$ iterations of development.

In a Scrum project environment, the architectural runway may be established during Sprint 0.

- Sprint 0 might have a longer duration than the rest of the sprints.
Create an architectural runway

The bigger the system, the longer the runway.
Leffingwell, Martens, Zamora

Use matrix teams and architecture

Establishing the infrastructure

Scrum Team A
Presentation Layer
Common Service

Scrum Team B
Domain Layer
API
Common Service

Scrum Team C
Data Access Layer
API
Common Service
Use matrix teams and architecture

Feature development in parallel

Scrum Team A
Scrum Team B
Scrum Team C

Presentation Layer
Domain Layer
Data Access Layer

Layer
Common Services
API
Feature 1

Team member with feature responsibility
Use matrix teams and architecture

Different teams are assigned to different features, and some team members are assigned to keep layers and framework consistent.
Use matrix teams and architecture

Different teams are assigned to different features, and a temporary team is assigned to prepare layers and frameworks for future feature teams.
Root-cause analysis: Typical problem 1

Symptom

• Scrum teams spend almost all of their time fixing defects, and new feature development is continuously slipping.

Root-cause

Inability to manage scope and time at scale

• Initial focus was “general” rather than “product specific.”
  – Time pressure to deliver became the top priority.
  – The team delivered an immature product.
  – A plethora of variation parameters interact detrimentally.

• There are three different cycles:
  – Customer release (annually, many variants); IV&V Testing (quarterly, 4 variants), and Developmental (monthly, 1 variant)
Solution

Stabilize the architecture.

• Build an architecture for current products.
  – Rules, guidelines
  – Over a few time boxes
• Reduce the number of “variant parameterizations.”
• Make everyone play from the same sheet music.
• Postpone adding new features.

Replan the release cycles/time boxes.

Revisit the testing strategy/team assignments against variants.
Root-cause analysis: Typical problem 2

Symptom

• Integration of products built by different Scrum teams reveals that incompatibility defects cause many failure conditions and lead to significant out-of-cycle rework.

Root-cause

Inability to manage teams at scale

• Cross-team coordination is poor, even though there are many coordination points and much time spent.

• Different teams have different interpretations of interfaces.

• The product owner on each Scrum team does not see the big picture.

• A mismatch exists between the architecture and Scrum development.
Solution

Stabilize to remove failures.
  • Postpone adding new features.

Identify and collapse common services across teams.

Use an architectural runway.
  • A system that has an architectural runway contains existing or planned infrastructure sufficient to allow incorporation of current and near-term anticipated requirements without excessive refactoring.

  • An architectural runway is represented by *infrastructure* initiatives that have the same level of importance as the larger scale requirements epics that drive the company’s vision forward.
Root-cause analysis: Typical problem 3

Symptom
• Progress toward meeting milestones is unsatisfactory.

Root-cause Inability to manage teams and scope at scale
• Mapping of features to software components per Scrum cycle is disorganized.
• Some new features are unused in each cycle, causing wasted effort.
• Developer assignment to teams is inflexible.
Solution

Build more architectural views to align features between teams.

Reorganize teams to better fit iteration and release workloads.

Create matrix teams to clean up unused features.
Final thoughts

No one tactic alone can take any project to success.

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

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

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

- Align feature and system decomposition.
- Create an architectural runway.
- Use matrix teams and architecture.
References


Upcoming

Forthcoming SEI Technical Report on
Managing Agility at Scale:
A Software Architecture Perspective
Bachmann, F., Nord, R., Ozkaya, I., Wojcik, R., Wood, W., and Brown, N.

IEEE Software Special Issue on Technical Debt
Guest Editors: Philippe Kruchten, Robert Nord, and Ipek Ozkaya
http://www.computer.org/portal/web/computingnow/swcfp6
Contact Information

Ipek Ozkaya
Research, Technology, and System Solutions Program
Architecture Practices Initiative
Email: ozkaya@sei.cmu.edu

Software Engineering Institute
Customer Relations
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
USA
As projects continue to grow in scale and complexity, effective collaboration across geographical, cultural, and technical boundaries is increasingly prevalent and essential to system success. SATURN 2012 will explore the theme of “Architecture: Catalyst for Collaboration.”