Achieving Agility and Stability in Large-Scale Software Development

Ipek Ozkaya
Senior Researcher,
Research, Technology, and System Solutions Program

Ipek Ozkaya is a senior member of the technical staff in the Research, Technology, and System Solutions Program at the SEI. She is currently engaged in activities focusing on large scale agile and architecture and works to develop, apply, and communicate effective methods to improve software development efficiency. Ozkaya serves as the chair of the advisory board of the *IEEE Software* magazine and is also a member of the technical faculty for the Master of Software Engineering Program at Carnegie Mellon University.

Agenda

Concepts of scale and root-cause analysis

Tactics that can help

Examples

Are we asking the right question?

How much should we architect?

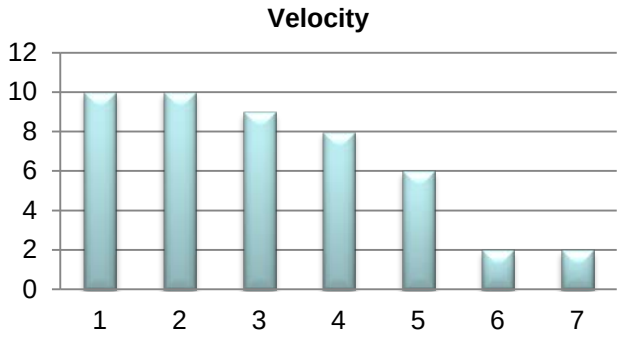
- This process, or that one…

When should we architect?

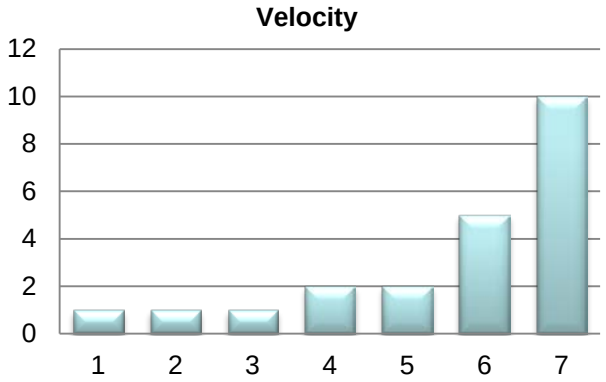
- Balancing anticipation versus adaptation…

Increased visibility into delivery

Focus on Priority

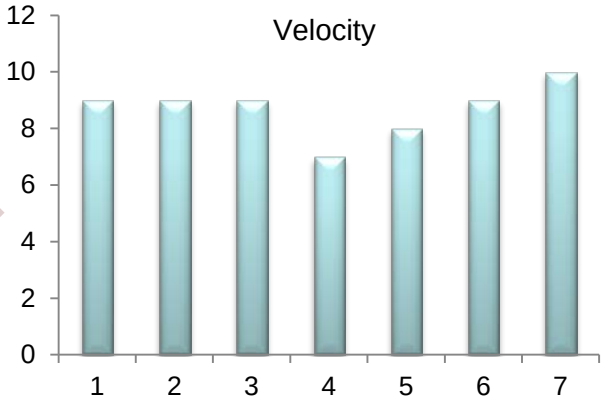


Focus on Cost



?

Focus on Integrated Value



Polling question

Which software development process are you currently using?

1. Agile software development (e.g., using Scrum, XP practices, test-driven development)
2. Waterfall/phased-based development
3. Rational unified process
4. Organization-specific iterative and incremental development
5. None of the above

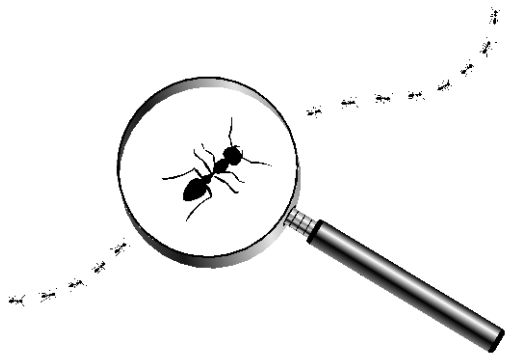
Symptoms of failure

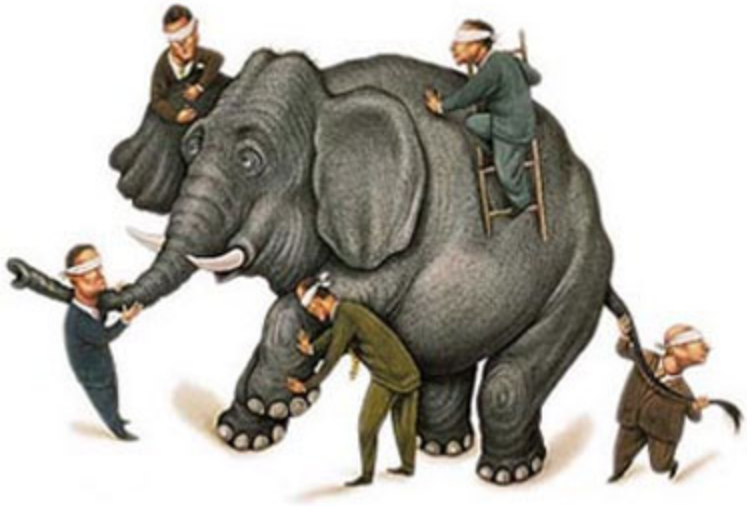
- Teams (e.g., Scrum teams, product development teams, component teams, feature teams) spend almost all of their time fixing defects, and new capability development is continuously slipping.

- 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.

- High testing costs result in some testing activities to be eliminated.

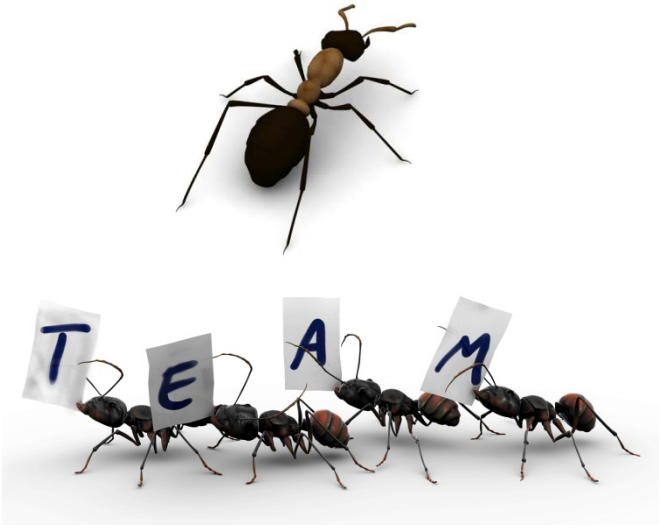
A closer look at scale: Scope

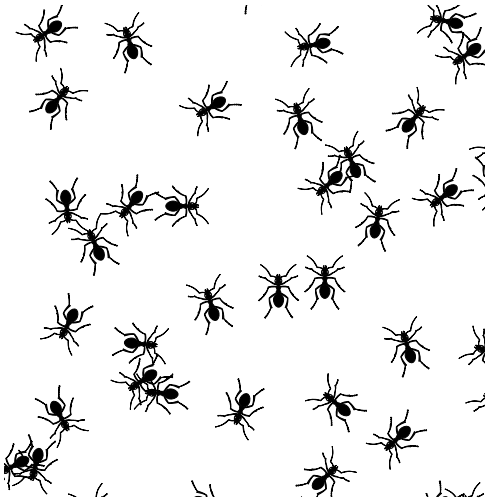




- Is the project in a new domain or technology?
- Does the project have new requirements—such as standards compliance, system testing, and integration lab environments—or does it simply have more features, elements, and relationships?
- Is there a need to align systems engineering and software development 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

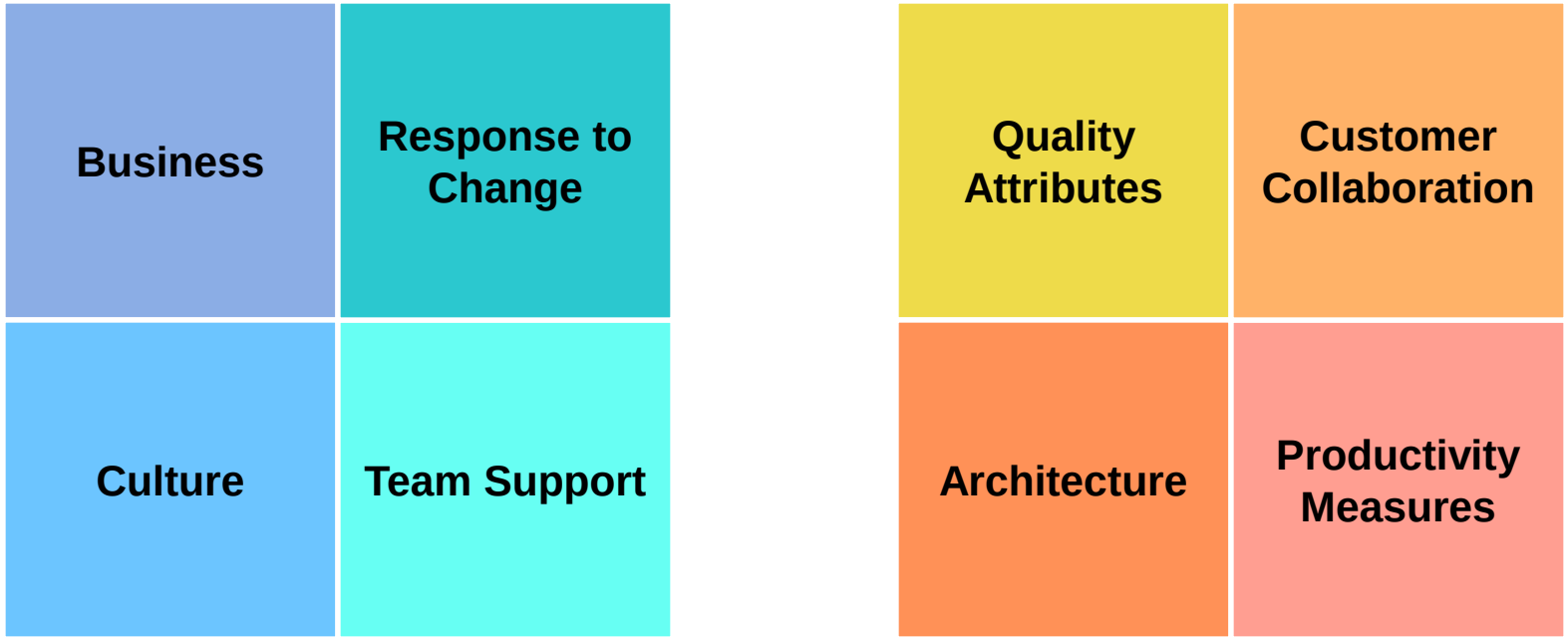
- Does the work require different schedule constraints for releases?
- How long is the work product expected to be in service?
- How important are sustainability and evolution?

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

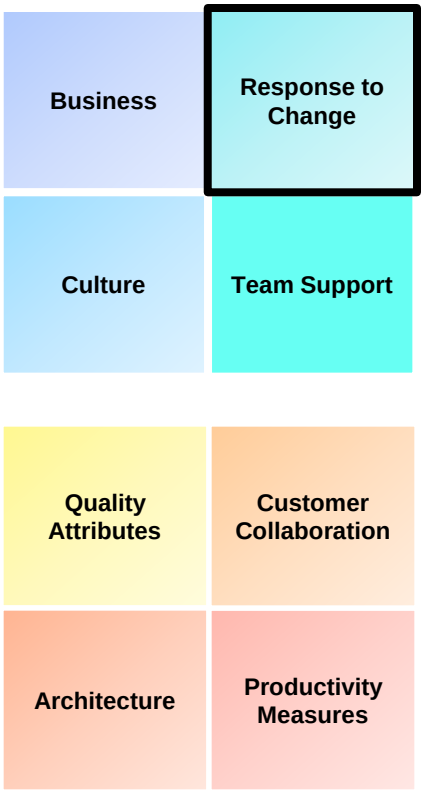


Investigate both technical and nontechnical areas, looking at both Agile software development and software architecture fundamentals.

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

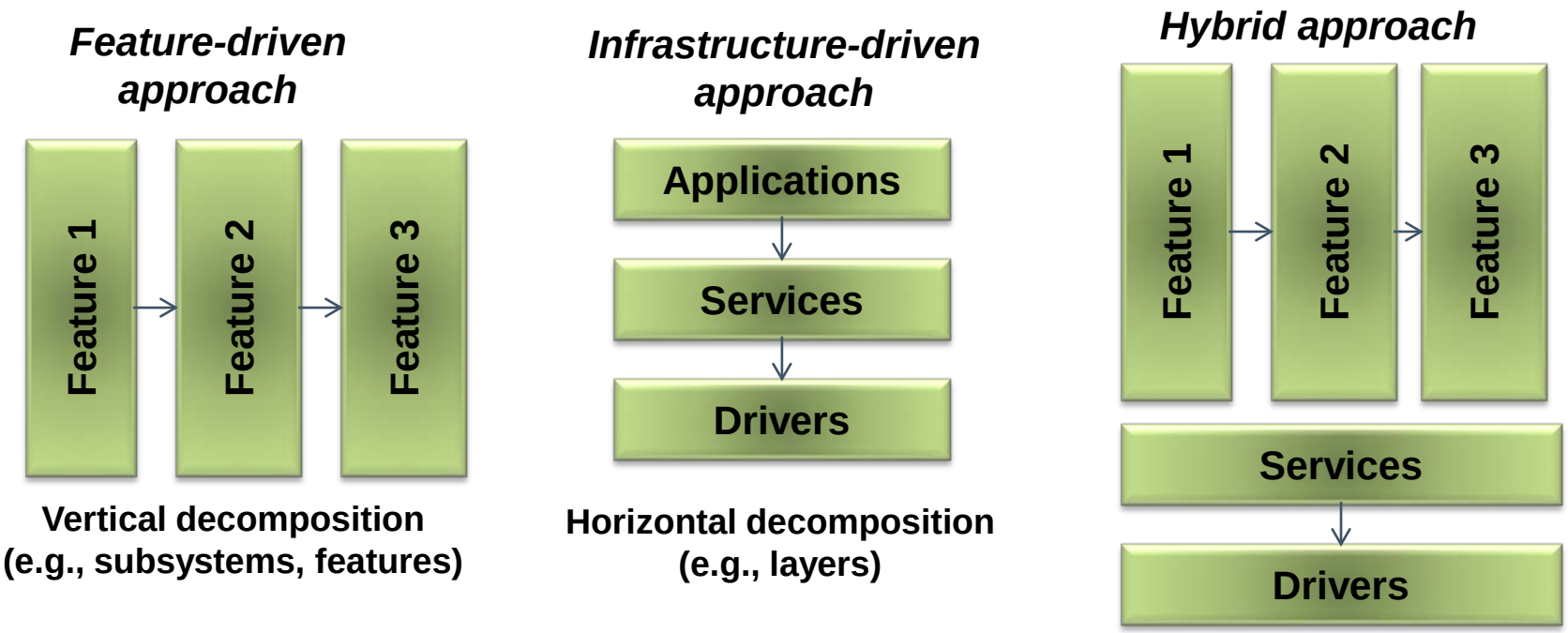
Dependencies between stories & supporting architectural elements	Understanding the dependencies between stories and architectural elements enables staged implementation of technical infrastructure in support of achieving stakeholder value.
Dependencies among architectural elements	Low-dependency architectures 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]

1. Poppendieck, M., and Poppendieck, T. *Leading Lean Software Development.* Addison-Wesley Professional, 2009.

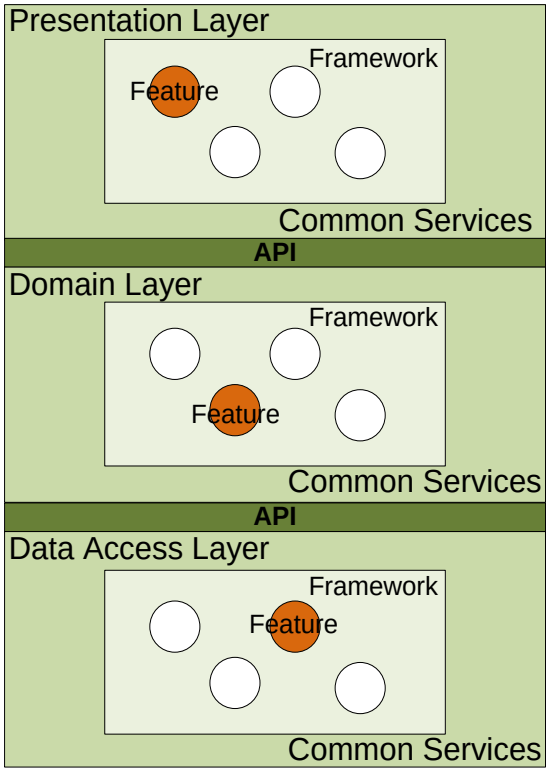
2. Denne, M., and Cleland-Huang, J. *Software by Numbers*. Prentice Hall, 2003.

Align feature and system decomposition

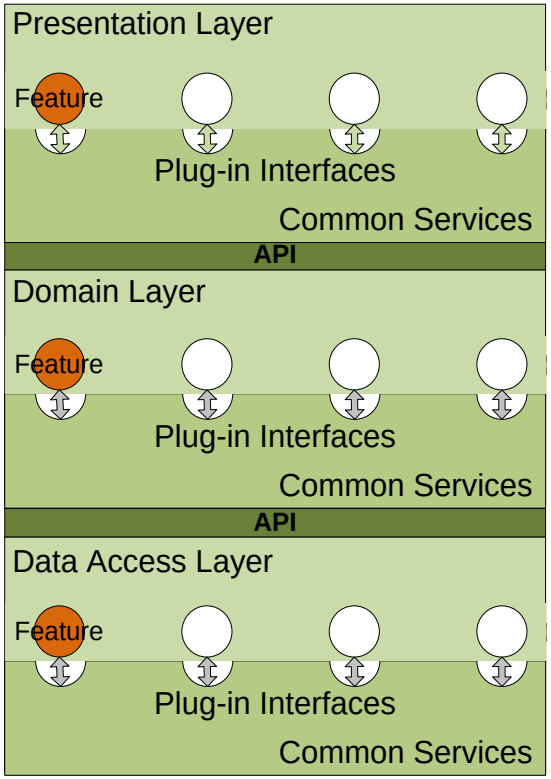
Tension between high-priority features (vertical decomposition) and common reusable services (horizontal decomposition)



Align feature and system decomposition
Two examples



Layered architecture with frameworks



Layered architecture with plug-ins

Decouple teams and architecture to ensure parallel progress as the number of teams increases.

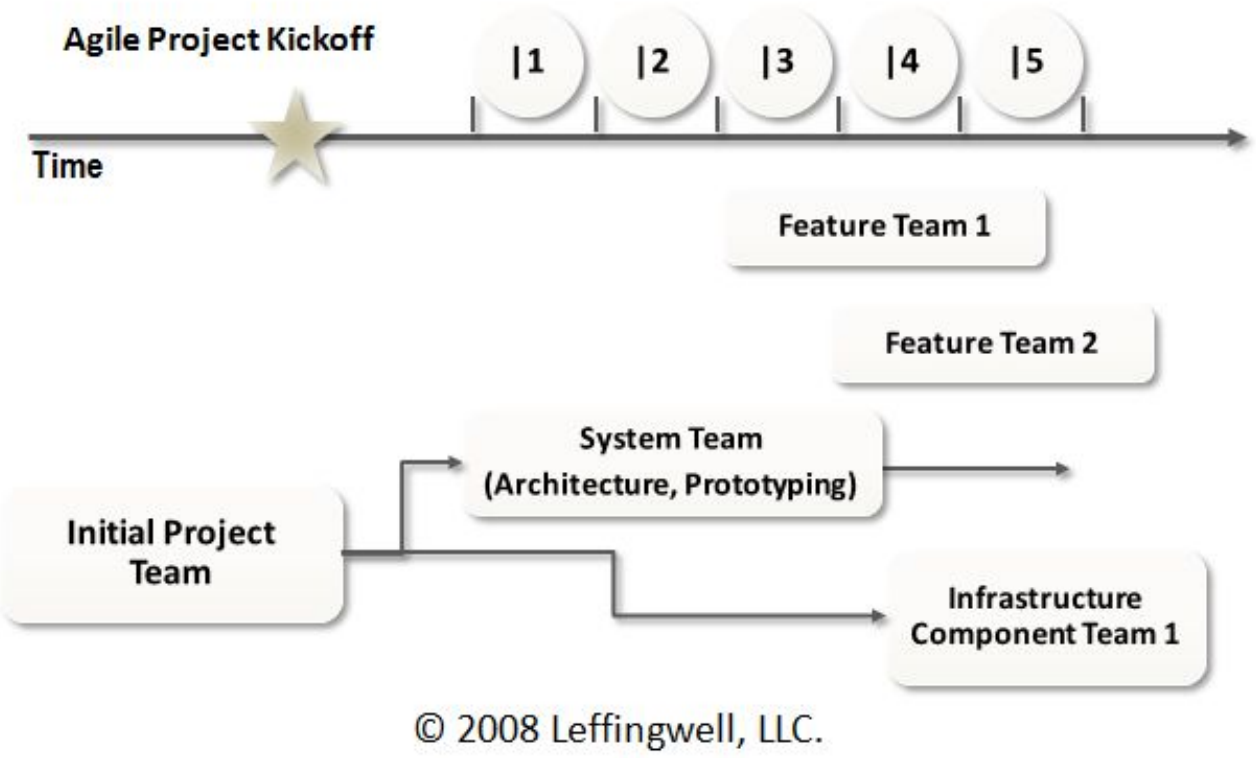
Create an architectural runway

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

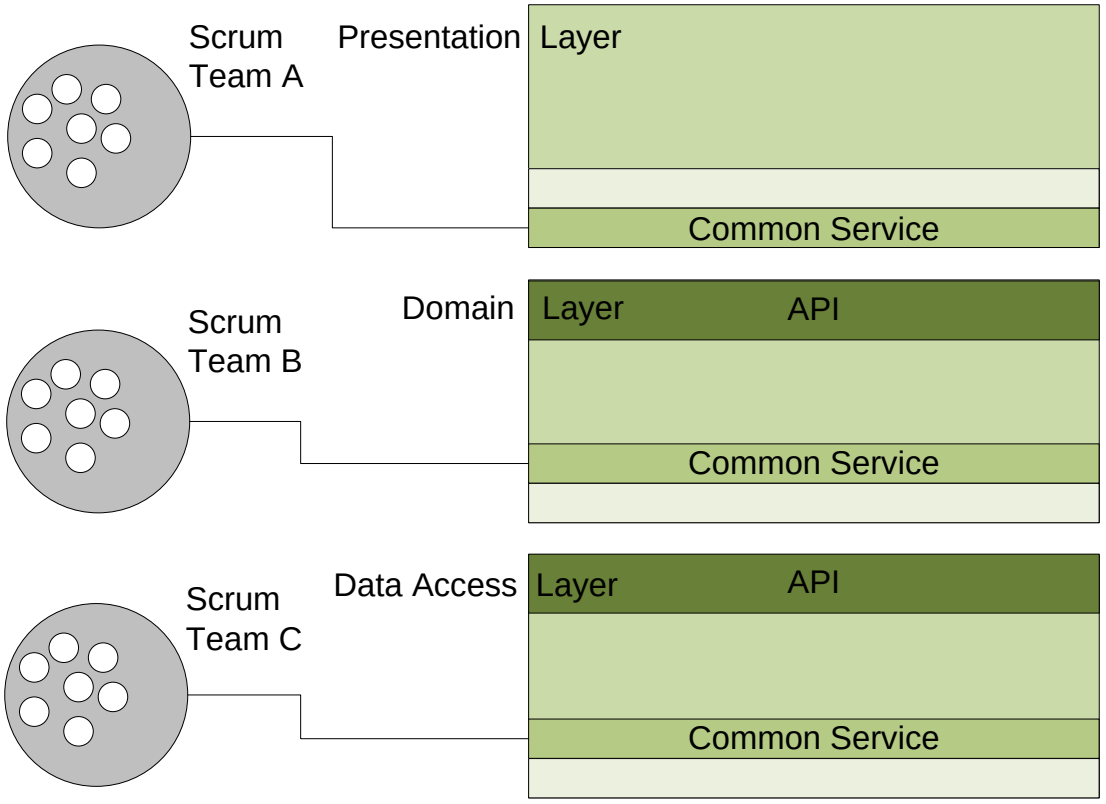


The bigger the system, the longer the runway.
Leffingwell, Martens, Zamora

Leffingwell, D. *Scaling Software Agility*. Addison-Wesley, 2007.
http://scalingsoftwareagility.wordpress.com/2008/09/09/enterprise-agility-the-big-picture-10-the-system-team

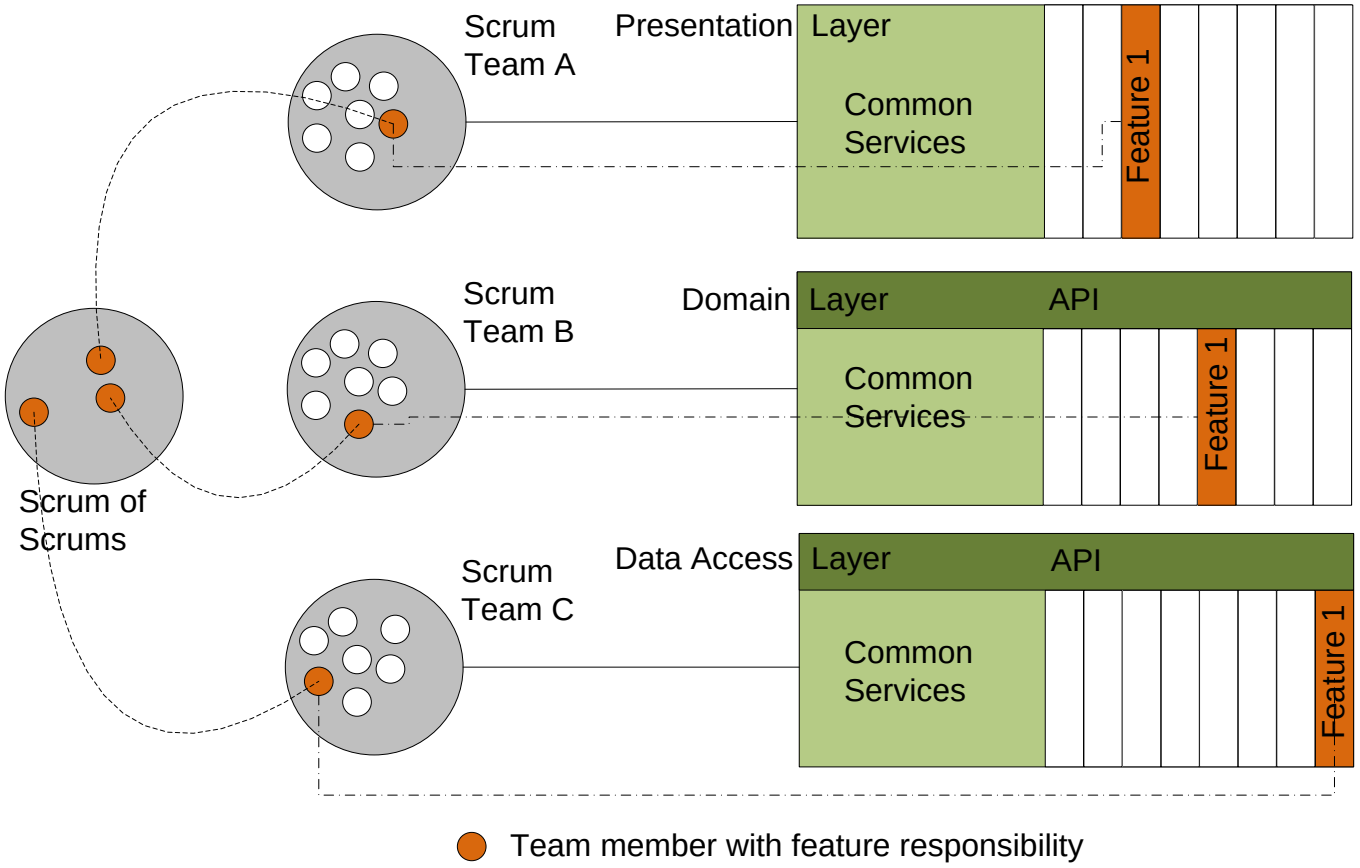
Use matrix teams and architecture

Establishing the infrastructure



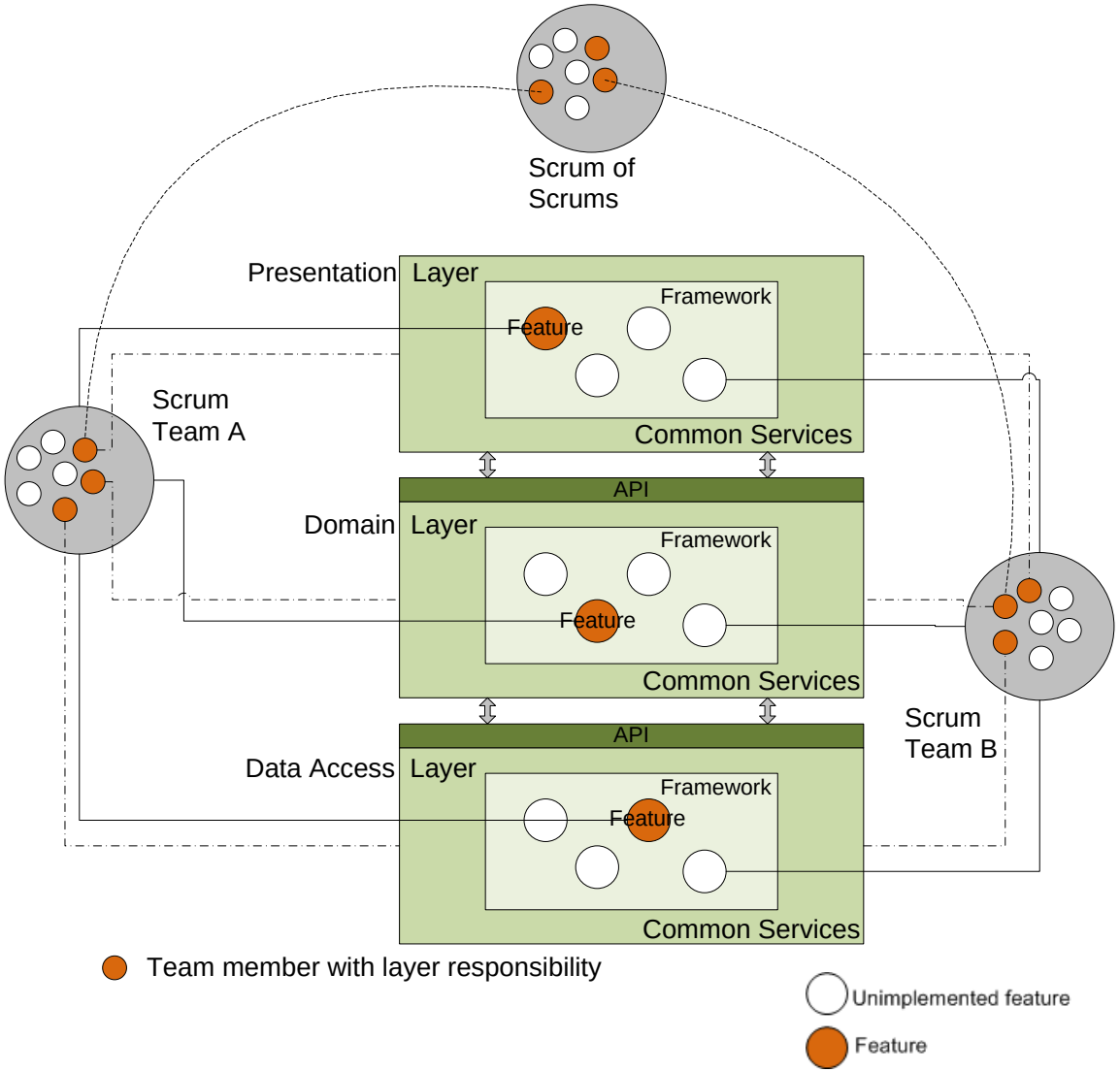
Use matrix teams and architecture

Feature development in parallel



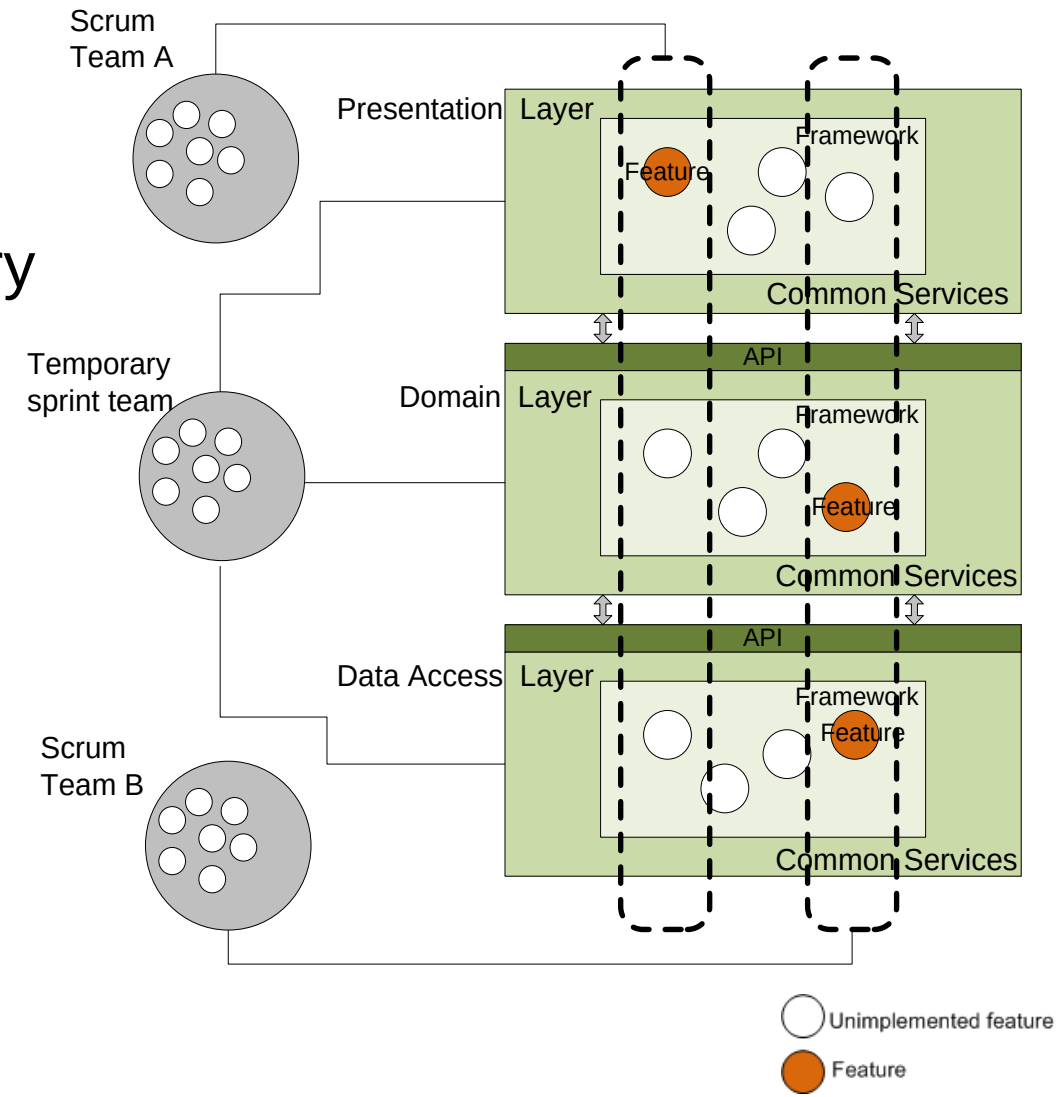
Use matrix teams and architecture

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.



Example 1: Inability to manage scope and time

Symptom

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

Root-cause

- 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:
  1. Customer release (annually, many variants)
  2. IV&V Testing (quarterly, 4 variants)
  3. 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.

Re-plan the release cycles/time boxes.

Revisit the testing strategy/team assignments against variants.

Example 2: Inability to manage teams and scope

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

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.

Final thoughts

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

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.

Architecting is an ongoing activity throughout the software development life cycle regardless of choice of process.

References

Ambler, S. The Agile Scaling Model (ASM): Adapting Agile Methods for Complex Environments. IBM developerWorks, 2009.

Bachmann, F., Nord, R. L., Ozkaya, I. Architectural Tactics to Support Rapid and Agile Stability. *Crosstalk* May/June 2012:

Brown, N., Nord, R., and Ozkaya, I. Enabling Agility Through Architecture. Crosstalk Nov./Dec. 2010: 12–17.

Denne, M., and Cleland-Huang, J. Software by Numbers, Prentice Hall, 2003.

Kruchten, P., Nord, R. L., Ozkaya, I.: Technical Debt: From Metaphor to Theory and Practice. IEEE Software 29(6): 18-21 (2012)

Kruchten, P. “What Color Is Your Backlog?” Agile Vancouver talk, 2009. http://files.me.com/philippe.kruchten/vuldw4

Larman, C., and Voddle, B. Scaling Lean & Agile Development. Addison-Wesley, 2009.

Leffingwell, D. Scaling Software Agility. Addison-Wesley, 2007.

Nord, R.L., Ozkaya, I., Sangwan, R. S. Making Architecture Visible to Improve Flow Management in Lean Software Development. IEEE Software 29(5): 33-39 (2012)

Poppendieck, M., and Poppendieck, T. Leading Lean Software Development. Addison-Wesley Professional, 2009.

Contact Information

Ipek Ozkaya
Senior Member of the Technical Staff
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

This material is based upon work 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.