Architectural Implications of DevOps

Stephany Bellomo
Senior Member of Technical Staff

Stephany Bellomo a senior member of the technical staff at Carnegie Mellon's Software Engineering Institute (SEI). Stephany received her Master's degree in Software Engineering in 1997 and spent several years as a software developer/technical lead for companies such as Intuit, Verisign and Lockheed Martin before joining the SEI. While at the SEI has had the privilege of working with a wide variety of government and DoD organizations such as Army, DHS, Veterans Affairs and various Intelligence Community Agencies. Current interests include research in Incremental Software Development. She also has interest in architectural implications of DevOps and Continuous Integration/Delivery. Stephany is a member of the organizing committee for the International Workshop on Release Engineering 2014 hosted by Google. She is also guest editor of IEEE Software magazine 2015 Special Issue on Release Engineering. She has been a Software Architecture Conference (SATURN) program committee member since 2010 and served as SATURN tutorial chair in 2014. Stephany also teaches courses in Service-Oriented Architecture and Software Architecture at the SEI.

Copyright

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.

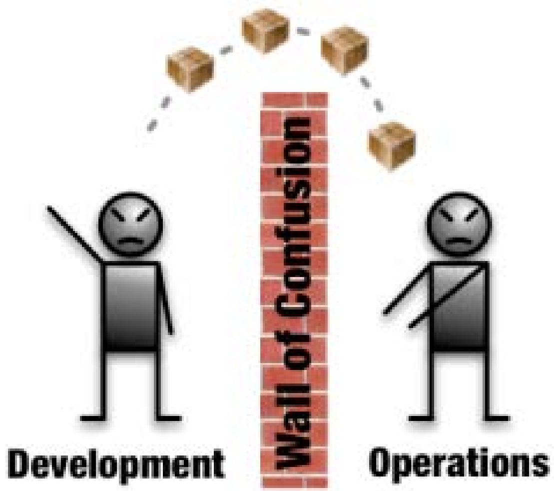
DM-0001132

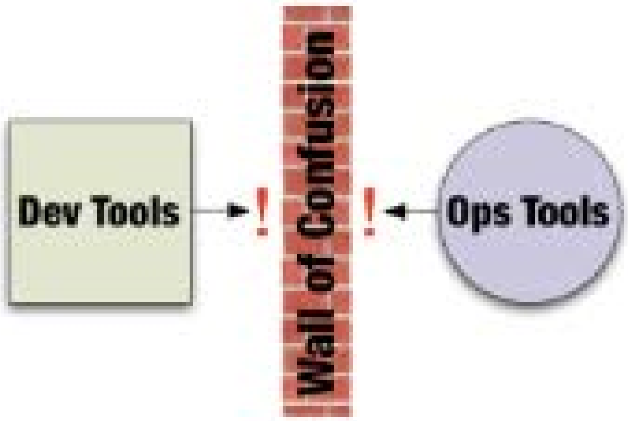
The DevOps Movement Began as a Reaction …

To years of disconnect between Dev and Ops which began to manifest itself as conflict and inefficiency

Familiar DevOps Problems

- Disconnect between Dev and Ops teams leads to a wall of confusion between stove-piped teams
- Disconnects between Dev and Ops tools, as well as processes, cause inefficiency and rework





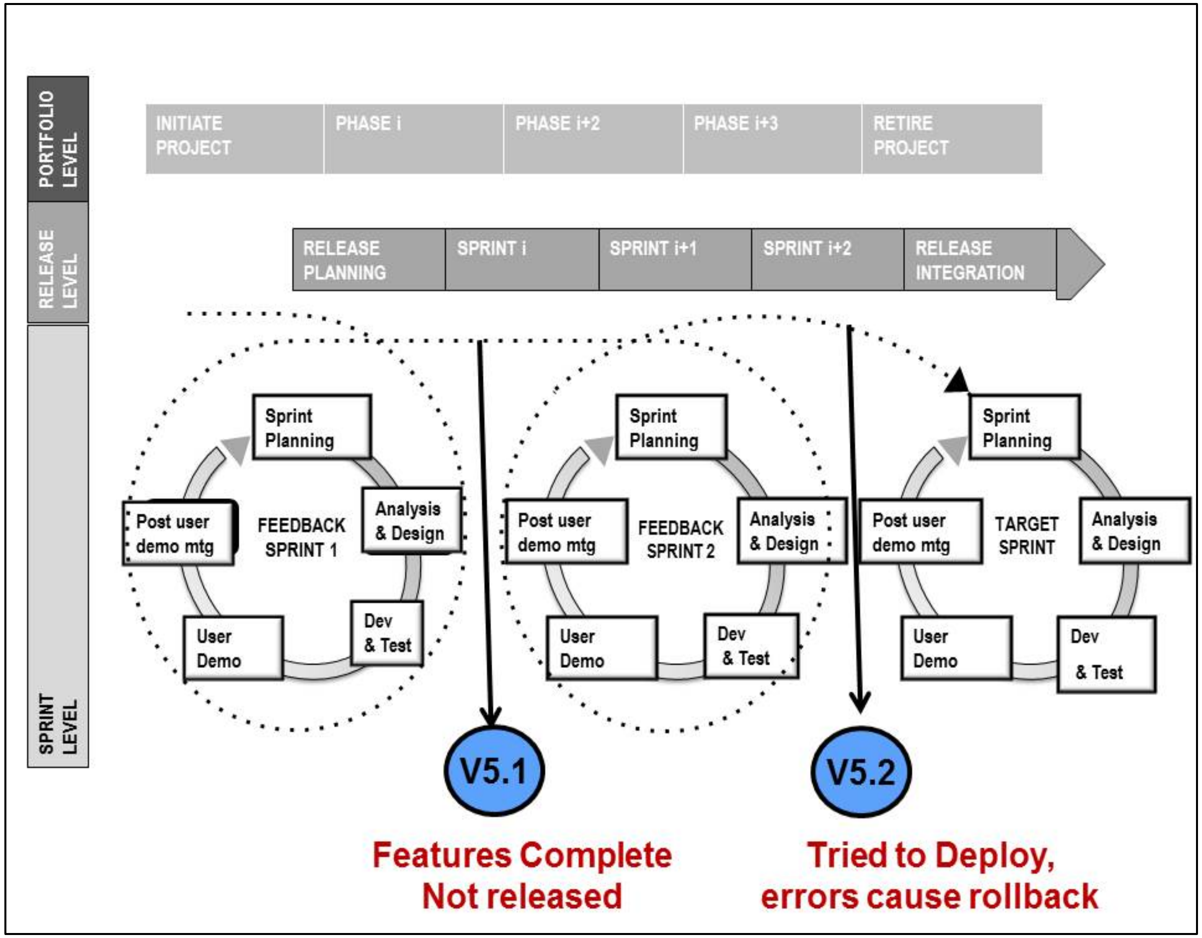
Source: Lee Thompson and Andrew Shaffer

DevOps is helping to finish what Agile started

We saw reduced development cycle time with Agile, but due to issues such as:

- Lack of confidence in deployment/ rollback
- Inefficient test approaches, etc.
- Unreliable software

Deployment cycle time is often weeks or months



No Value gained when Software is not Delivered

Informal DevOps Definitions

"DevOps is a software development method that stresses communication, collaboration and integration between software developers and information technology (IT) professionals"

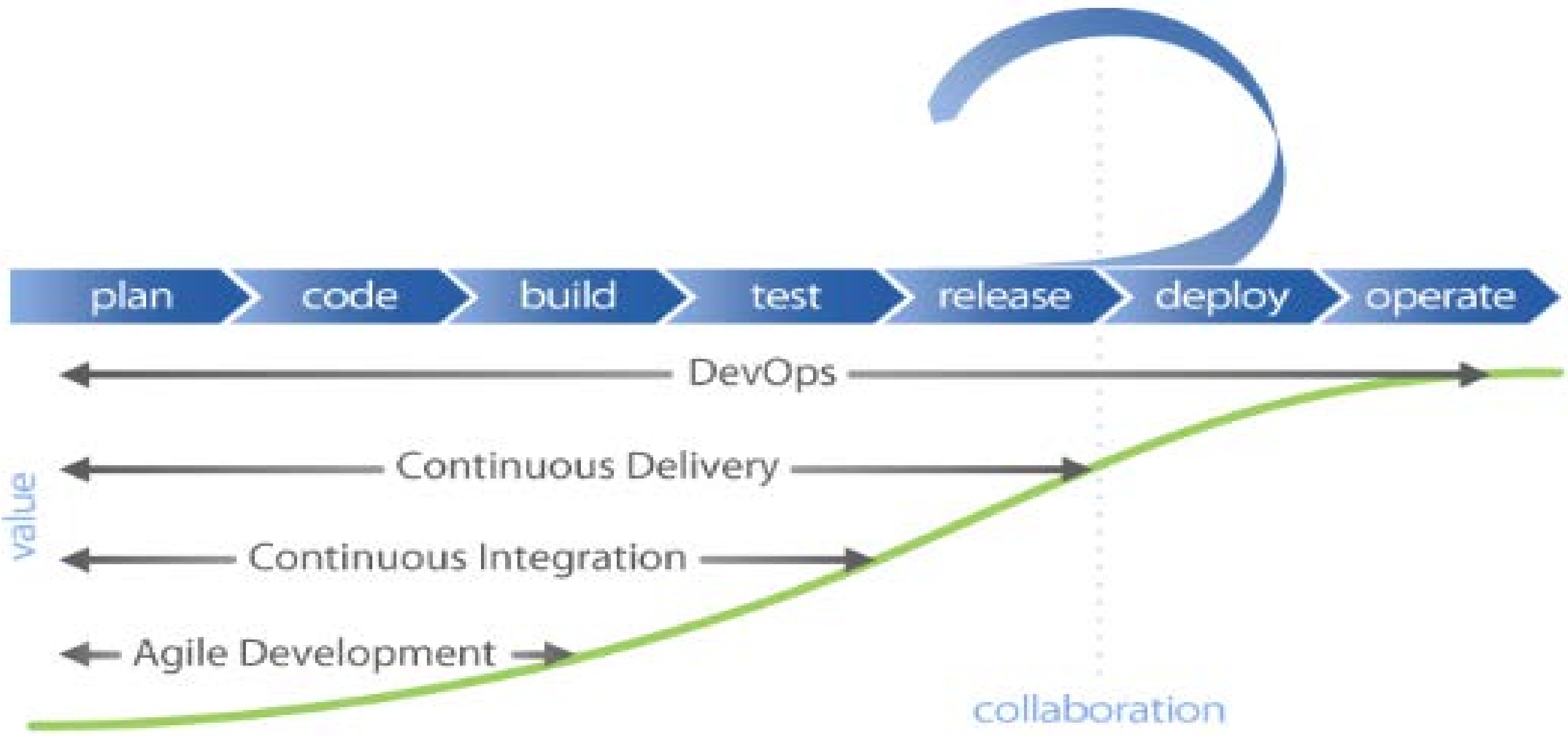
Pant, Rajiv

"DevOps is an ***umbrella*** concept for anything that smooth's out the interaction between development and operations"

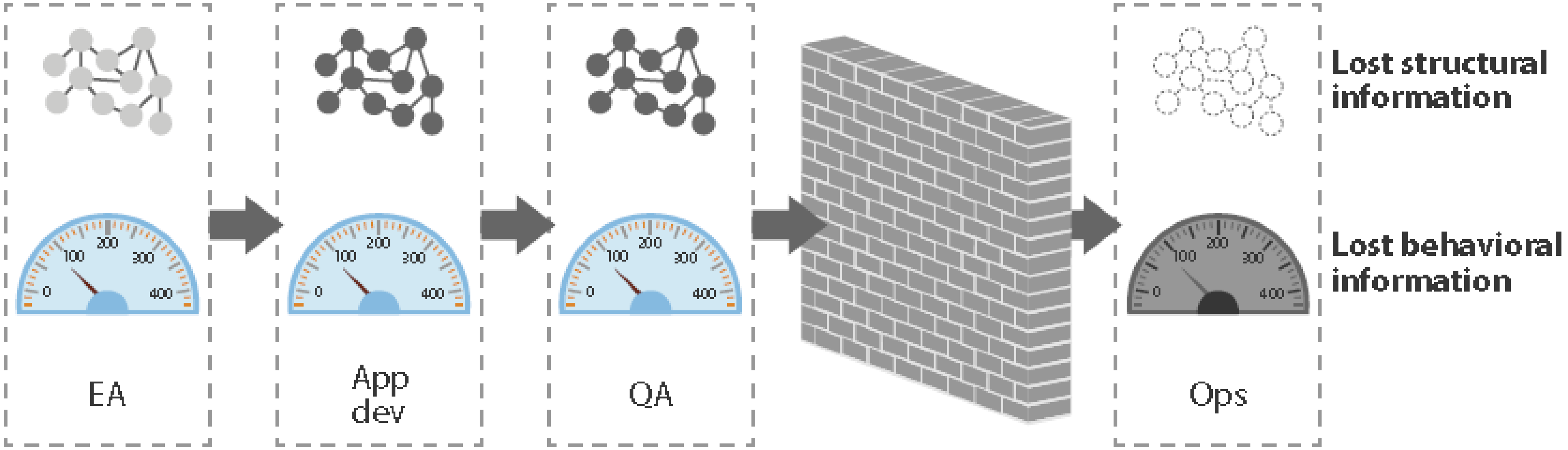
Damon Edwards

Scope

The scope for DevOps looks at reducing deployment cycle time and enabling feedback cycles across the end-to-end Deployment Pipeline …



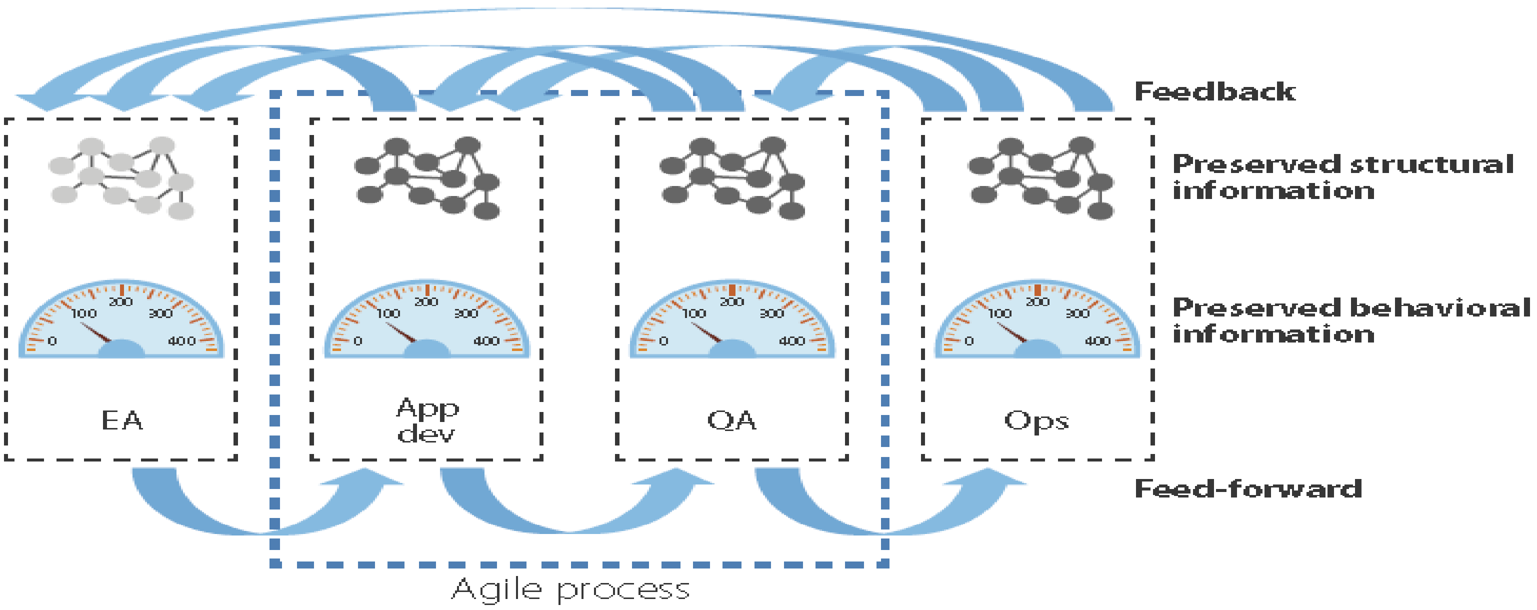
Challenges DevOps is trying to Solve



- Non-collaborative stove-piped Dev and Ops teams
- Limited improvement within stove-piped areas (e.g., process, tools, metrics) but not end-to-end
- Broken feedback cycles; process flows only one way

Forrester, The Seven Habits Of Highly Effective DevOps

DevOps Community Future Vision

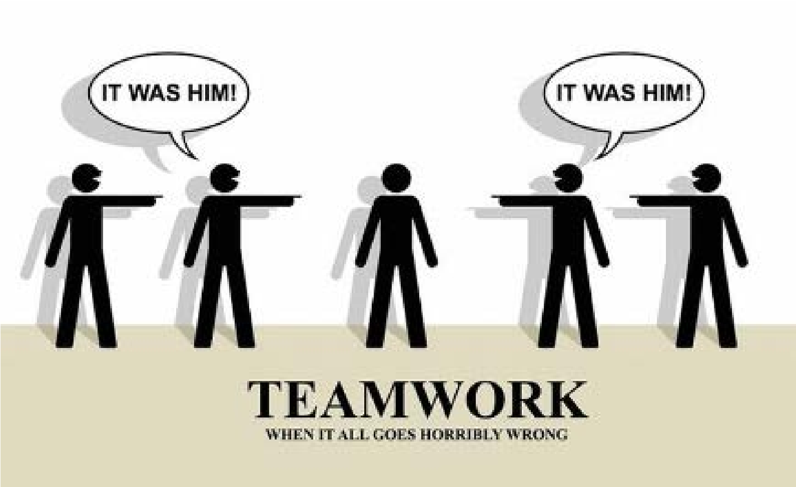


- Collaborative, Dev and Ops teams combine or working closely together
- Continuous improvement across the deployment pipeline targeted at producing something of value to a user or organization (inception to dev to release/sustain)
- Effective feedback cycles within each stage

Adapted from Forrester, The Seven Habits Of Highly Effective DevOps

More than Dev and Ops Working Together

Those are some of the overarching goals of DevOps, but is easy to think of DevOps as just a collaborative movement because people get that



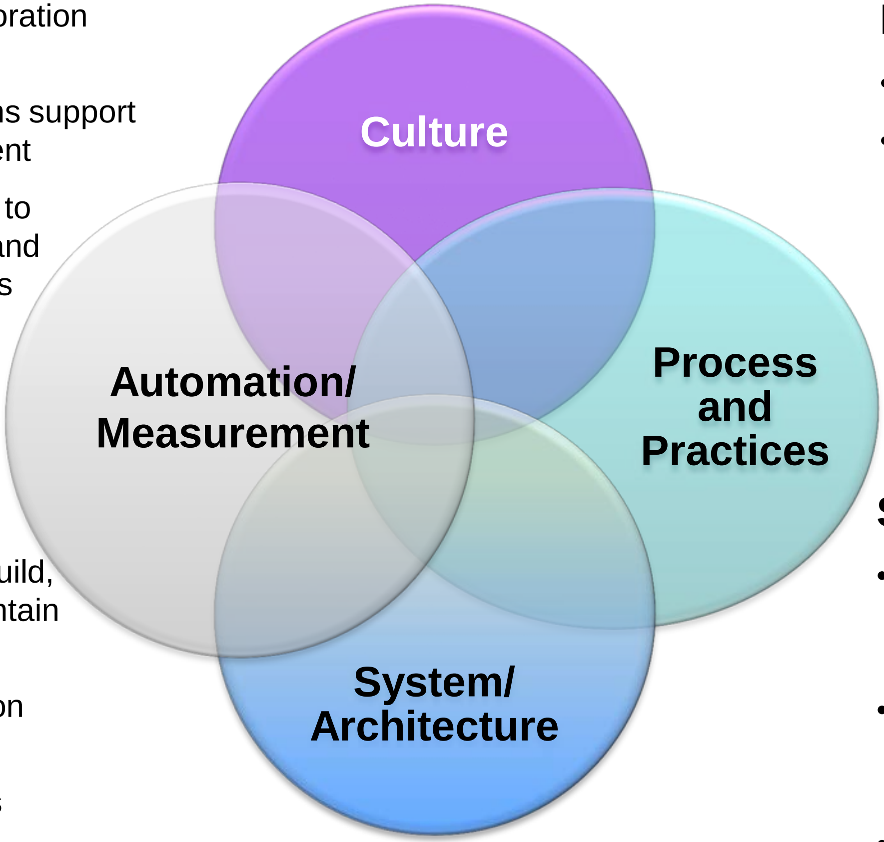
But it is really more than that

- There are multiple dimensions to the movement…

Multiple Dimensions of DevOps

Culture

- Developer and Ops collaboration (Ops includes Security)
- Developers and Operations support releases beyond deployment
- Dev and Ops have access to stakeholders who understand business and mission goals



Process and Practices

- Pipeline streamlining
- Continuous Delivery practices (e.g, Continuous Integration, Test Automation, Script-driven, automated deployment, Virtualized, self-service environments)

Automation/ Measurement

- Automate repetitive and error-prone tasks (e.g., build, testing, deployment, maintain consistent environments)
- Static analysis automation (architecture health)
- Performance dashboards

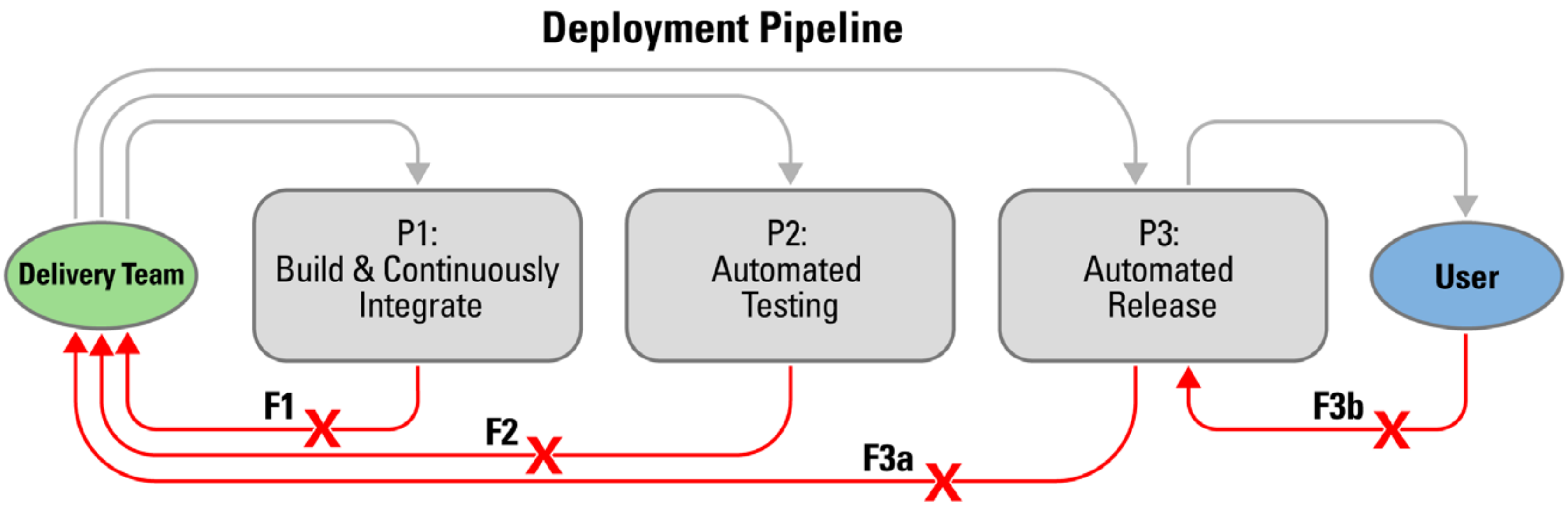
System/Architecture

- Architected to support test automation and continuous integration goals
- Applications that support changes without release (e.g., late binding)
- Scalable, secure, reliable, etc.

Ignoring any of these dimensions can cause problems

Feedback Cycle Breakdown Examples

Architecture can enable or imped short feedback cycle time



Examples of Feedback Cycle breakdown due to Architecture Issues:

- **F1:** Builds take too long due to poorly managed component dependencies; integration builds are slow and become infrequent
- **F2:** System doesn't have architectural interfaces for test automation and manual tests are slow; tests are skipped
- **F3a&b:** Architecture creates deployment complexity and error prone manual steps prevent release; weeks/months without release

Challenge Questions

We just gave several examples of how architecture can enable or impede feedback cycles, and consequently, end-to-end deployment cycle time (we refer to as Deployability)

However, this raises several questions such as:

- How do we specify *Deployability requirements* clearly and concisely?
- How do we design systems for Deployability?
  - What kinds of design decisions really matter?
  - Are there architectural tactics and/or patterns we might want to leverage to promote Deployability?
- When planning work, what Deployability-related requirements and design decisions should be considered early to avoid rework?

Requirements for Deployability

Lack clear specification for Deployability requirements leads to feedback cycle breakdowns

Example Vague Requirements:

"Our system, and delivery environment, shall support continuous delivery and multiple deploys a day like Amazon, Google, etc."

"When it comes to deployment, everything possible should be automated"

In next few slides, we give examples of Deployability requirements that enable better feedback across the deployment pipeline

Specifying Deployability Requirements

Well specified requirements enable Feedback Cycles; Several example Deployablity Requirements are shown below:

P1: Build and Continuously Integrate

- Complete full software build in **< 5 minutes** under peak load

P2: Automated Testing

- Complete execution of Unit tests suite within **10 minutes**
- Complete execution of increment tests suite (e.g., NFR) within **5 hours**
- Create/build a new system-level test case, avg time to build/test is **1 day**

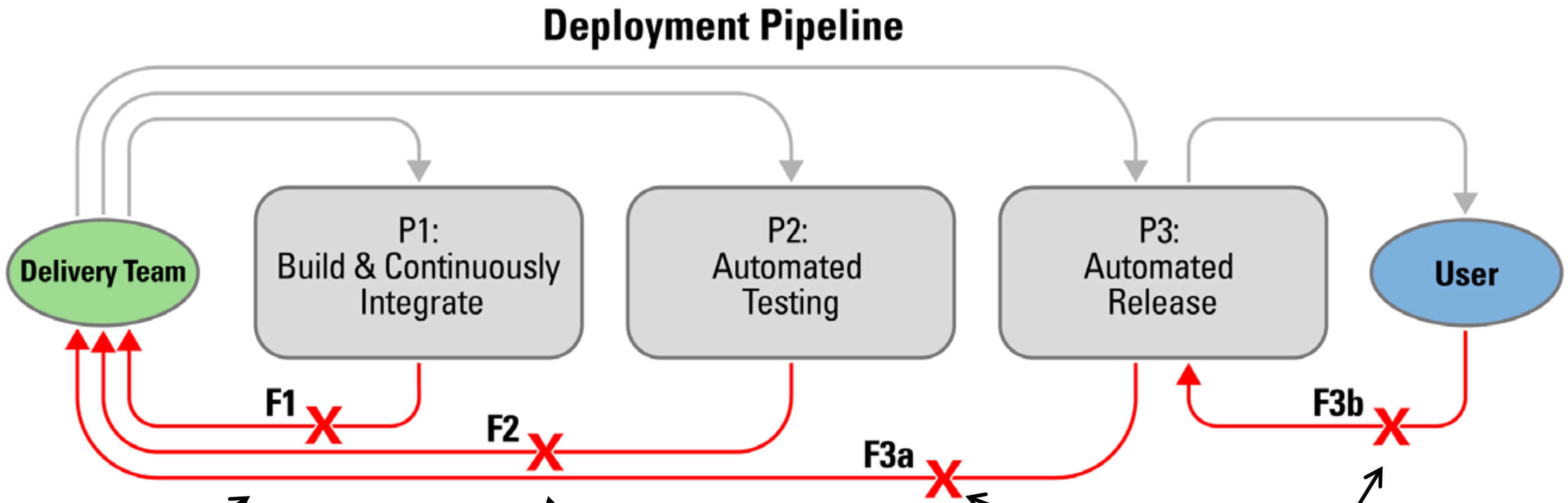
P3: Automated Release

- There is an upgrade being pushed out, **99% of release** is automated and 1% is handled manually
- The team makes a change to feature X (UI and business logic change) and deploy is pushed out within **2 hours** of code/test completion

Source: ATAM Analysis Data 2006-2013

Requirements Mapped to Feedback Cycles

Deployablity requirements specified as quality attributes can provide concrete measures for designing systems to achieve feedback cycle time



After a change is committed, complete build in < 5 minutes

Complete execution of Unit tests within 10 minutes; increment tests complete in 5 hours

The team makes a change to feature X and deploy is pushed out within 2 hours of code/test completion

Source: Towards Design Decisions to Enable Deployability, DSSO workshop paper submission (in review)

Design Decisions to promote Deployability

- We just gave examples of Deployability requirements; next we investigate design decisions. We draw upon interviews with projects practicing continuous delivery (sampling below)…

Project	Management Approach	Size Metrics	Years In Use	Release Cadence	CI Cadence
A	Agile/Scrum (last 2 years and traditional before that)	1M SLOC	17	Client release available every 2 months (not all accept it)	Daily CI build
B	Water/Scrum/Fall	3M SLOC, team size 6–8, 90,000 users	3+	Internal release every 2–3 weeks, external release as needed	Daily CI build
C	Agile/Scrum	Team size 30	2+	Internal release every 2–3 weeks, customer release every 2–3 months	Daily CI build

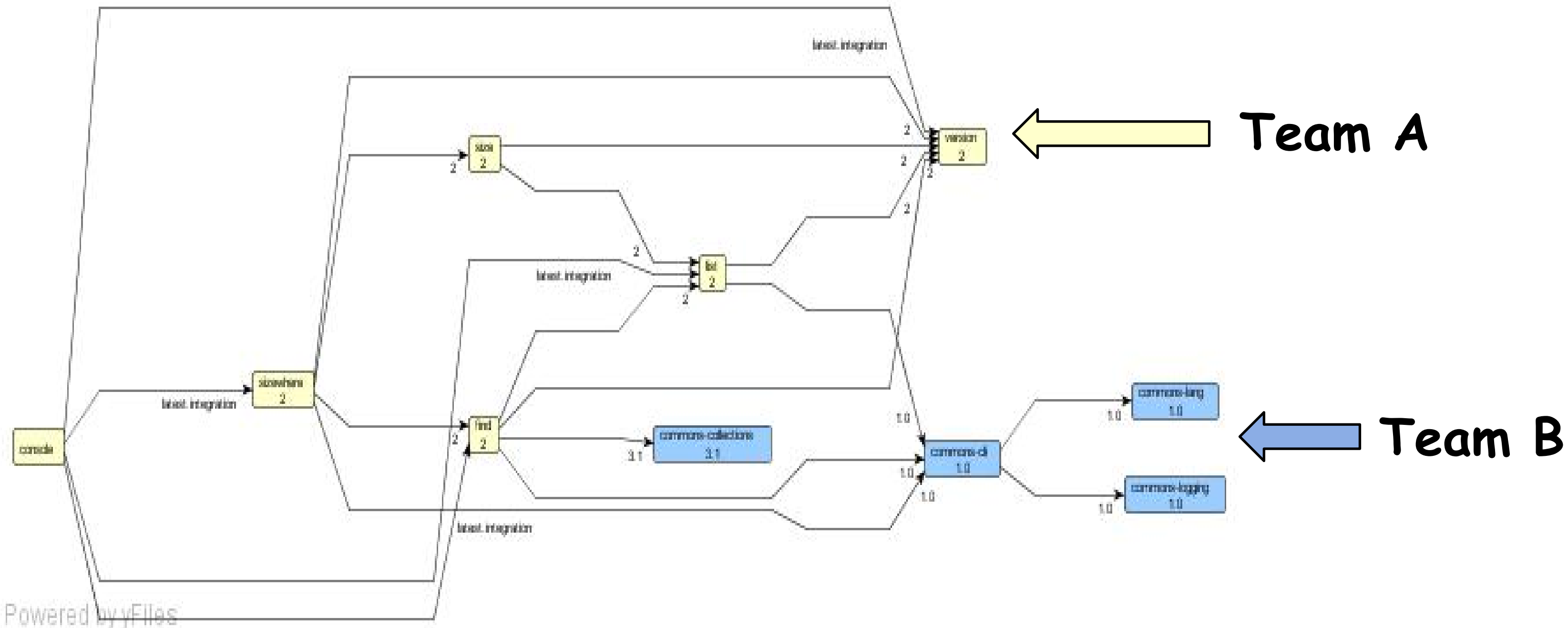
Source: Towards Design Decisions to Enable Deployability, submitted Dependability and Security Workshop, Bellomo, Kazman, Ernst

Architecture Partitioning Decision

Trade-offs
+Modifiability
+Testability
+Reduced Build Time
-Reuse

Decision: Divide components and allocation teams separately to promote rapid builds and tests

- Changes to blue components (Team B) do not require rebuild of yellow components (Team A) which shortens build time



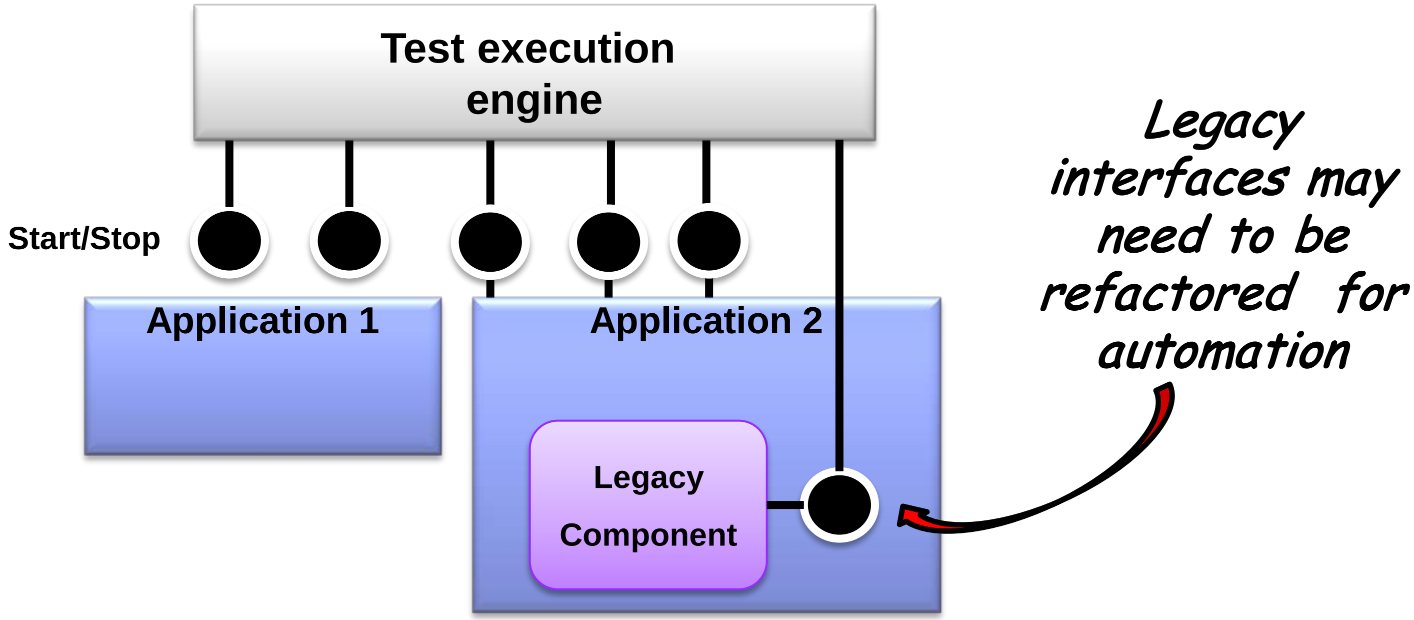
Source: Ant.patch.org

Integrated Test Harness Decision

Trade-offs
+Testability
+Modifiability
-Complexity

Decision: Integrate test harness hooks to architecture to start and stop application (start in clean state, end test with clean environment)

- Shortened Test Duration



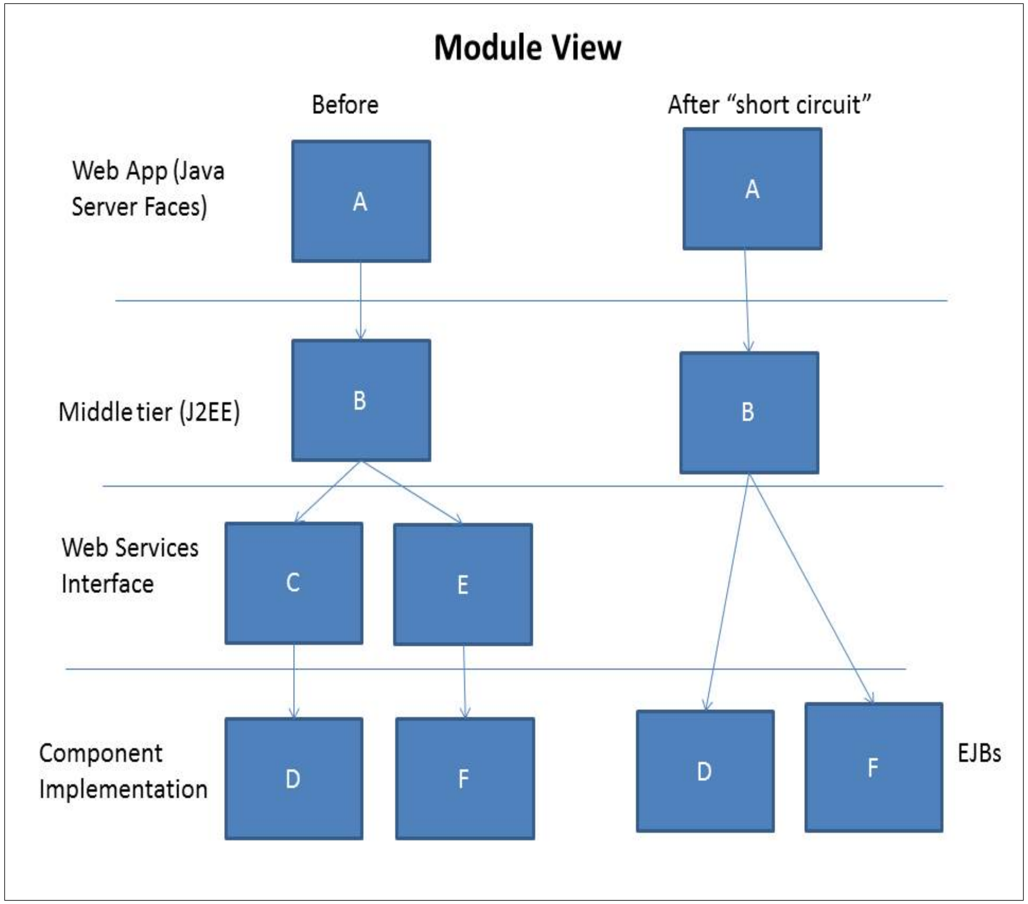
Web Services Layer Removal Decision-1a

Decision: Remove web services layer; replace with Enterprise Java Bean implementation

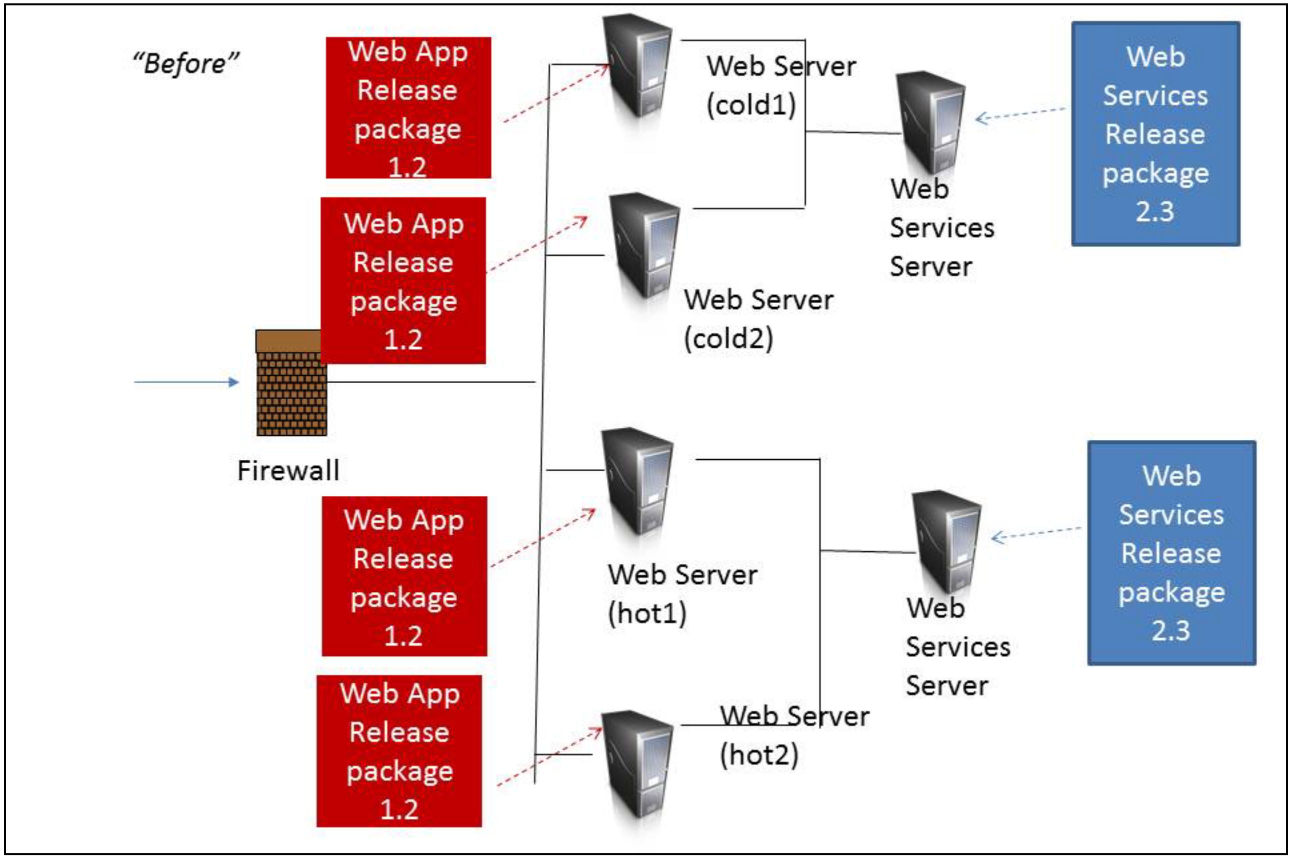
- Minimized Deployment complexity

Trade-offs

+Releasability
+Reduced Complexity
+Performance
-Testability
-Modifiability

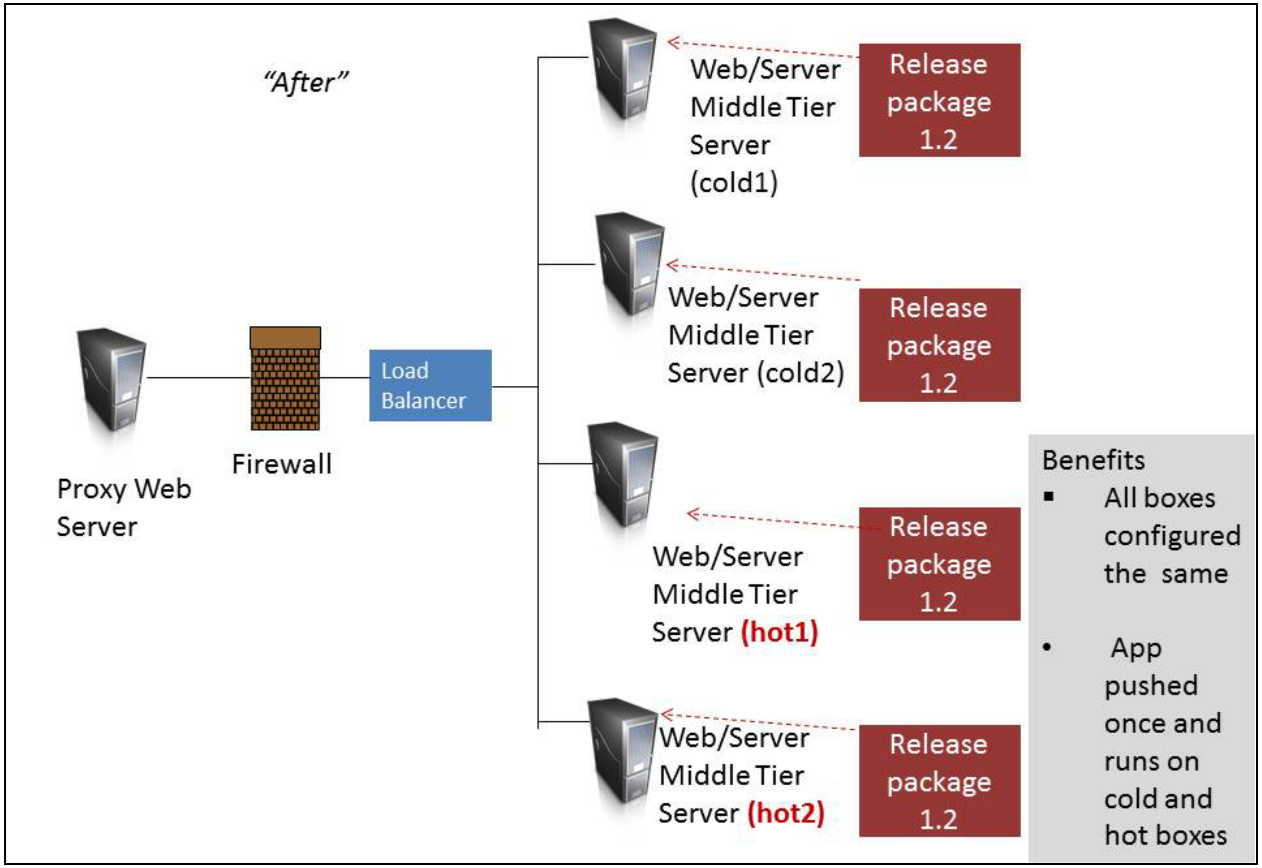


Web Services Layer Removal Decision-1b (Before redesign)



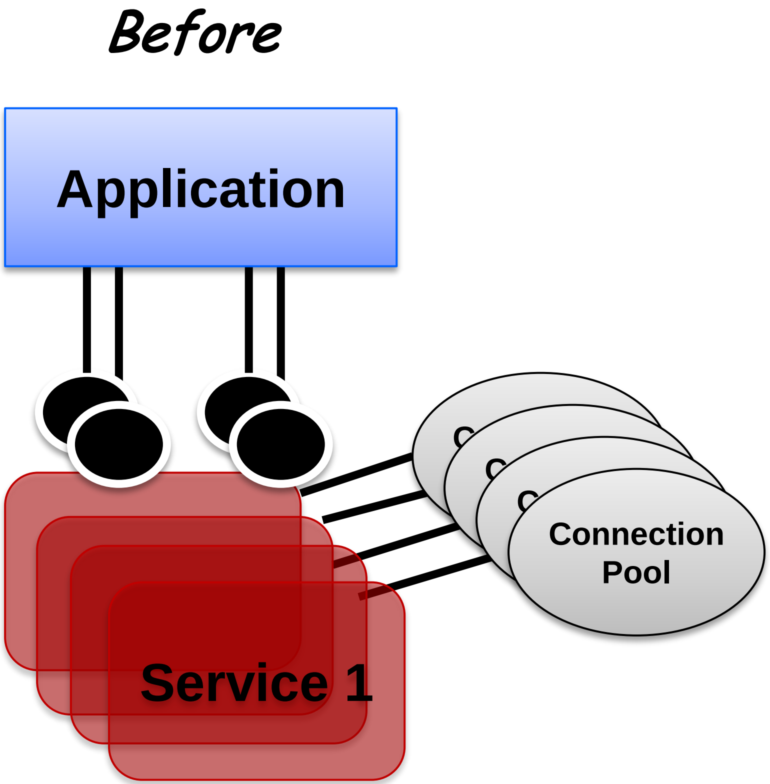
- Before, had to update multiple application servers and web services to be sure that application and services versions were in synch

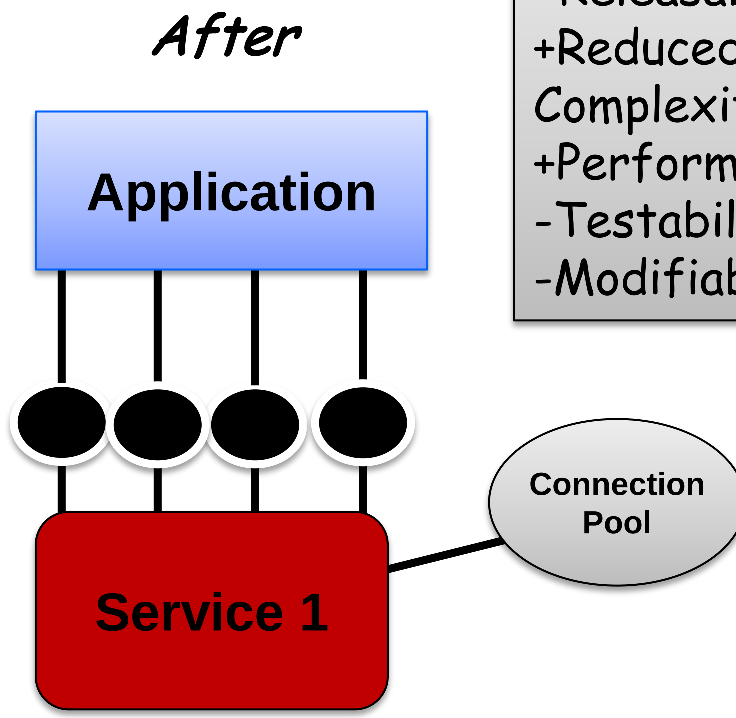
Web Services Layer Removal Decision-1c (After redesign)



Web Service Consolidation Decision

Decision Example: Consolidate Web Services for easier release, increased performance and reduced complexity



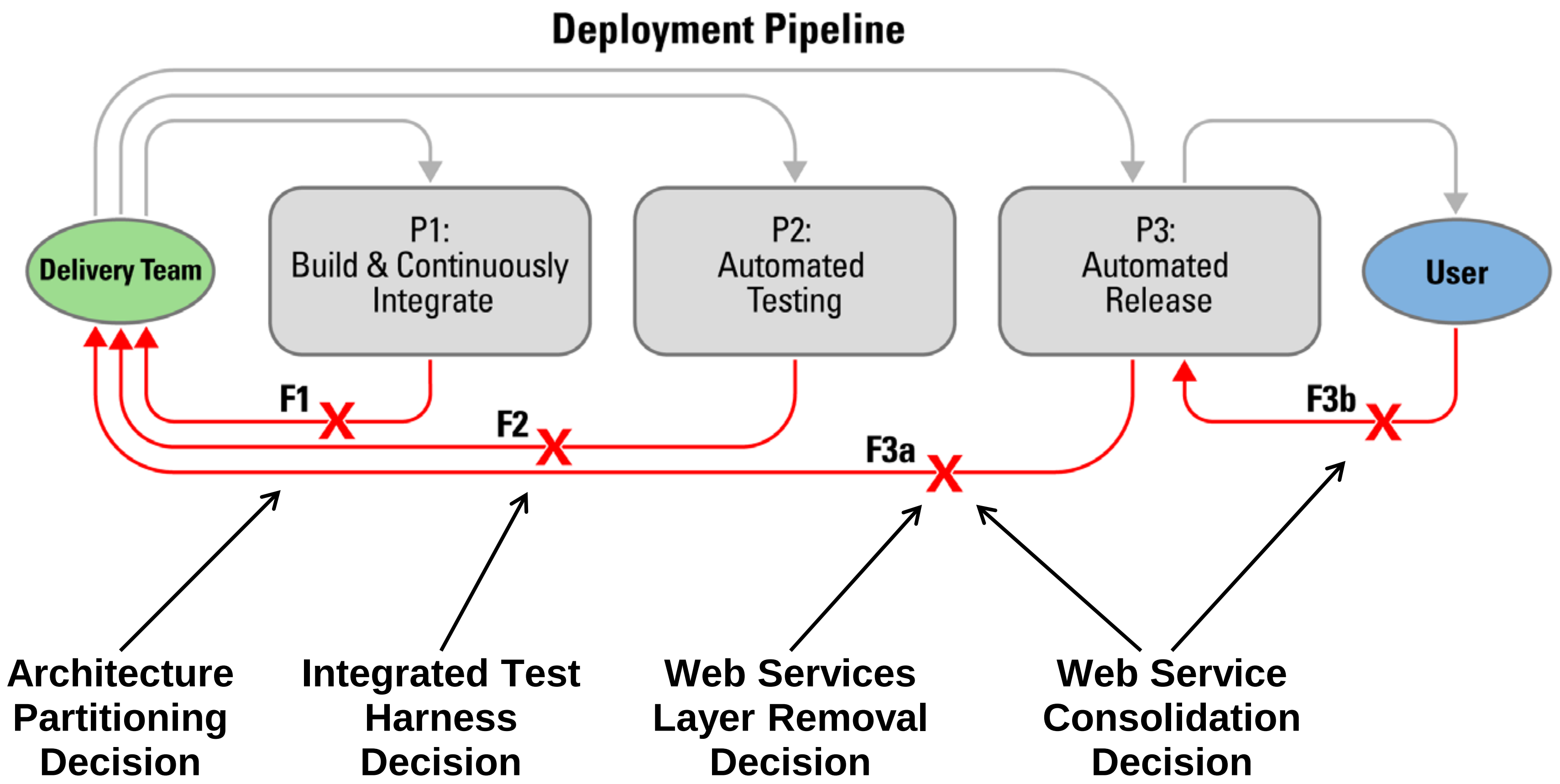


Trade-offs

+Releasability
+Reduced Deploy Complexity
+Performance
-Testability
-Modifiability

Mapping Design Decisions to Pipeline

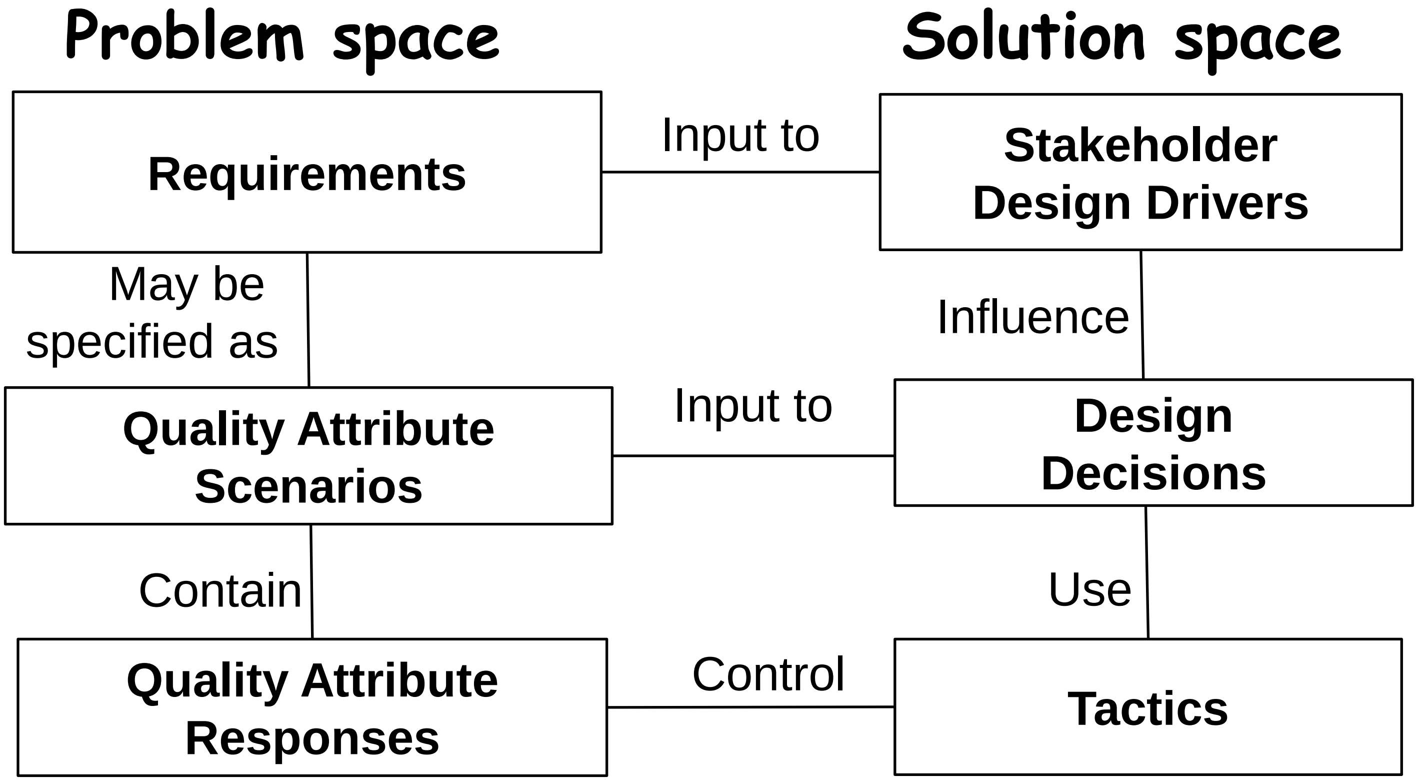
Each design decision also supports the pipeline feedback loops



Source: Towards Design Decisions to Enable Deployability, DSSO workshop paper submission (in review)

Relating Terms and Concepts

In the next few slides, we give a few examples that connect from requirements to design decisions to tactics; The ER diagram below provides an overview of concepts we are discussing



Integrated Test Harness Example

Problem: Long testing duration due to problems with establishing clean test start state and difficulty executing tests in automated fashion (manual steps required)

Broken Feedback loop:
Long Automated Testing Cycle

Requirement Scenario:

"Complete execution of increment tests suite (e.g., NFR) within 5 hours"

Design Decision:
Integrated test harness

Tactics Used:

- Specialized Access Routines
- Record/playback
- Maintain Interfaces,
- State Synchronization & resychronization

Fixed Feedback loop:
Shortened Test Duration

Modular and Distributed Architecture Example

Problem: Long deployment duration due to problems with architectural dependencies

Broken Feedback loop:
Infrequent deployments

Requirement Scenario:
"The team makes a change to feature X (UI and business logic change) and deploy is pushed out within 2 hours of code/test completion"

Design Decision:
Distribute & modularize architecture

Tactics Used:
- Increase Semantic Coherence
- Encapsulation
- Maintain Existing Interfaces

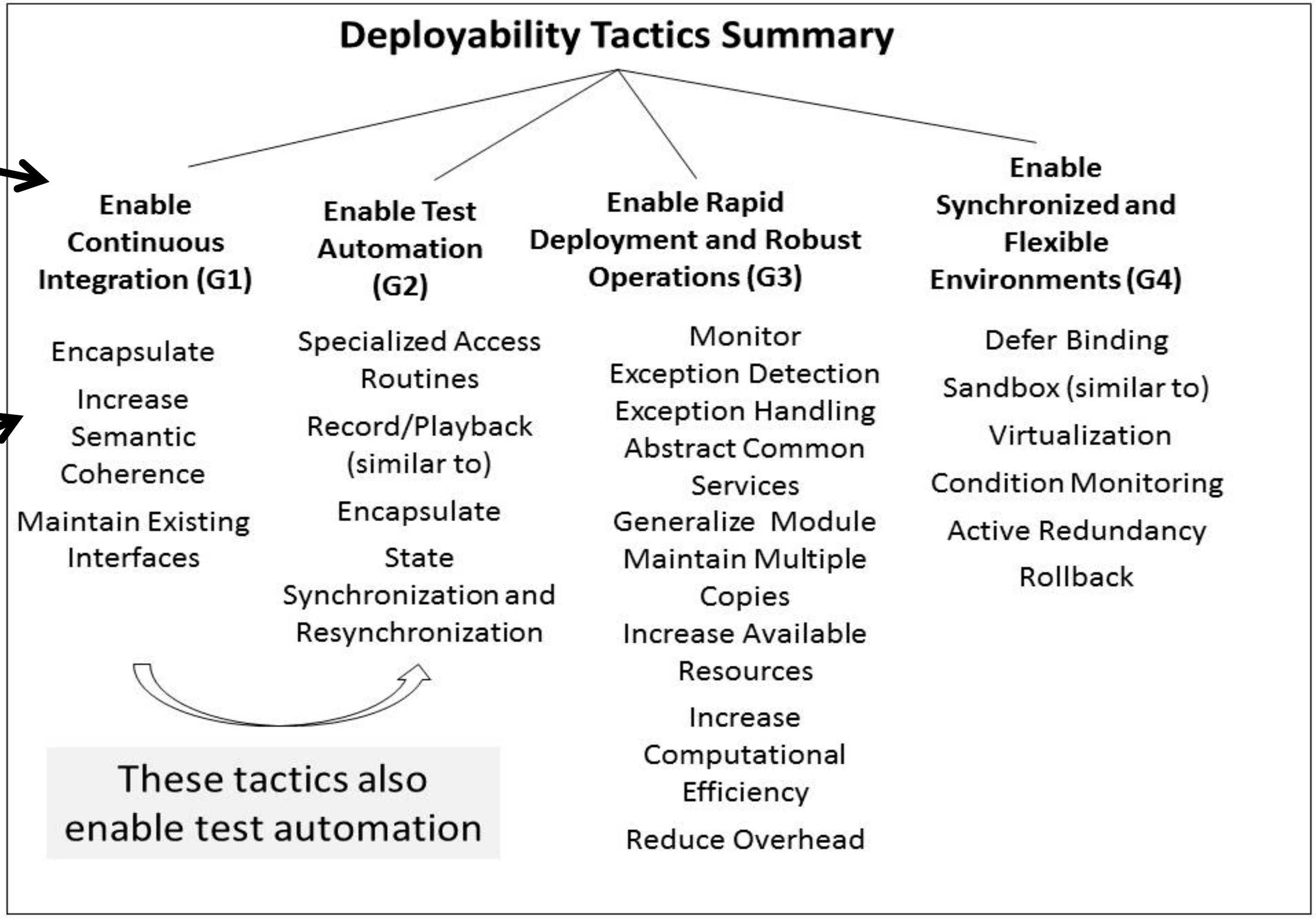
Fixed Feedback loop:
Reduced Deployment time

"If you push the whole three million line application every time a change is made you are in a world of hurt" Project C

Deployability Architecture Tactics Tree

Top Row Represents Stakeholder Design Drivers

Deployability Tactics



Source: Towards Design Decisions to Enable Deployability,
submitted Dependability and Security Workshop, Bellomo, Kazman, Ernst

Modular and Distributed Architecture Example

Deployability Tactics Summary

Enable Continuous Integration (G1)
- Encapsulate
- Increase Semantic Coherence
- Maintain Existing Interfaces

Enable Test Automation (G2)
- Specialized Access Routines
- Record/Playback (similar to)
- Encapsulate
- State Synchronization and Resynchronization

Enable Rapid Deployment and Robust Operations (G3)
- Monitor
- Exception Detection
- Exception Handling
- Abstract Common Services
- Generalize Module
- Maintain Multiple Copies
- Increase Available Resources
- Increase Computational Efficiency
- Reduce Overhead

Enable Synchronized and Flexible Environments (G4)
- Defer Binding
- Sandbox (similar to)
- Virtualization
- Condition Monitoring
- Active Redundancy
- Rollback

Integrated Test Harness Example

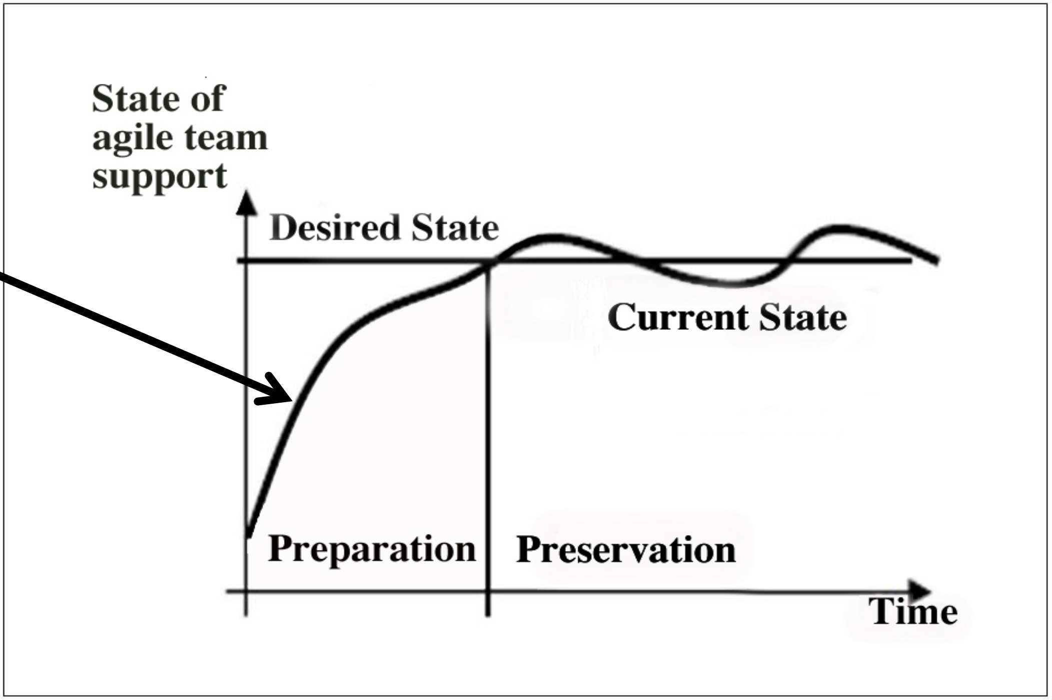
These tactics also enable test automation

"Need Speed and Rigor"

Allocating Deployability

- Our examples suggest some Deployablity-related design decisions/trade-offs can have significant impact

- In cases where the structure of the architecture is impacted by a decision, it may make sense to consider them early to avoid rework



Designing for Deployability, like any quality attribute, requires well informed architectural trade-off analysis

Wrap Up

In this talk, we have shared an approach for:

- Describing Deployability concerns as architecturally significant scenarios
- Applying trade-off analysis to make Deployment-focused design decisions
- Leveraging tactics to control Deployability-related response measures

Work to be done

- Collect more examples of scenarios, design decisions and tactics
- Expand and further validate the Deployability tactics tree
- Apply Deployabliltiy tactics to help teams reduce deployment cycle time and enable feedback cycles across the deployment pipeline (e.g., tactic checklist)

Want to get involved?

Upcoming activities

- IEEE Software Magazine Special Issue on Release Engineering, April/May 2015
- SATURN SEI Software Architecture Conference, 2014, May 5-9 Portland Oregon, *Tutorial on Architecture Tactics to Reduce Deployment Cycle Time*

Contact Information:

Stephany Bellomo,
sbellomo@sei.cmu.edu

Rick Kazman
kazman@sei.cmu.edu

Neil Ernst
nernst@sei.cmu.edu

Rod Nord
rn@sei.cmu.edu