

SATURN 2019

Scaling Up Incremental Design Reviews: A Tutorial

Felix Bachmann

Stephany Bellomo

Document Markings

Copyright 2019 Carnegie Mellon University. All Rights Reserved.

This material is based upon work funded and supported by the Department of Health and Human Services (HHS) under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

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.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

DM19-0472

Introduction

Design reviews are useful tools for architects to identify early software problems, but there are considerations at scale

Today's design review must ...

- handle multiple concurrent activities generating artifacts that result in information overload
- avoid impeding project progress by doing risk analysis while the train is moving
- understand that small teams are a big thing and collaborate with them to have an impact

Over the past several years, we have refined an approach to incremental design review with characteristics that help architects deal with these things

This is what we will present today

History of the Scaled Incremental Review

We ran the first incremental experiment of our approach in 2009–2011 with a small team and limited context; characteristics included

- financial system, strenuous performance, reliability, extensibility requirements
- replacement of an old system, some new features
- team size: 30 people (developers, testers, architects, requirements)

Since then we have used the approach over a 2-year span; characteristics have included

- number projects in portfolio we oversee: 14
- software category: various IT systems for a large organization
- team sizes: range 5–20
- number of design/code reviews conducted: 22

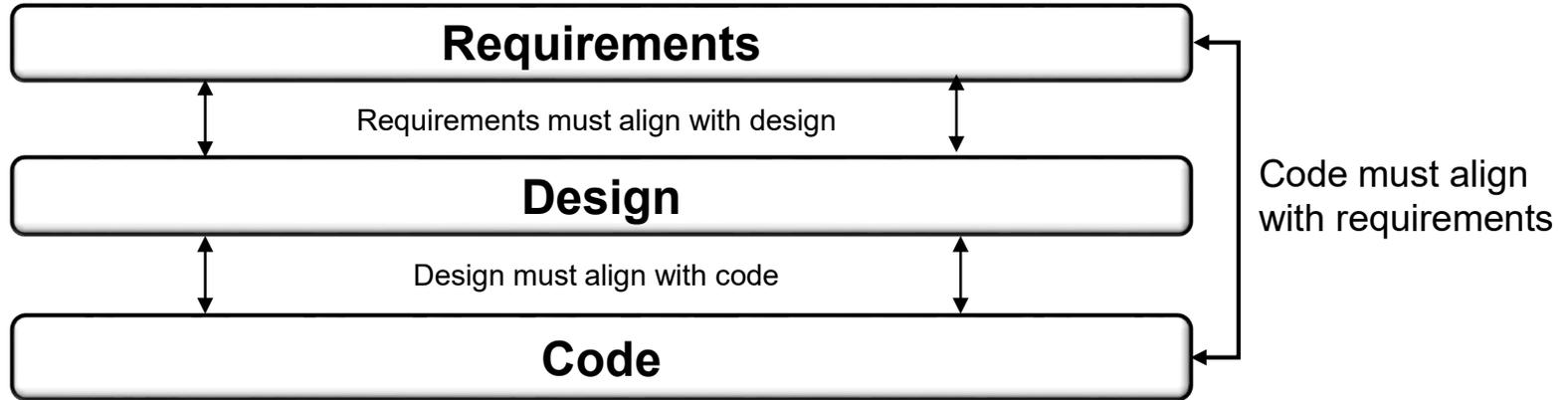
Terms and Assumptions

Below are informal definitions of terms and assumptions in this tutorial:

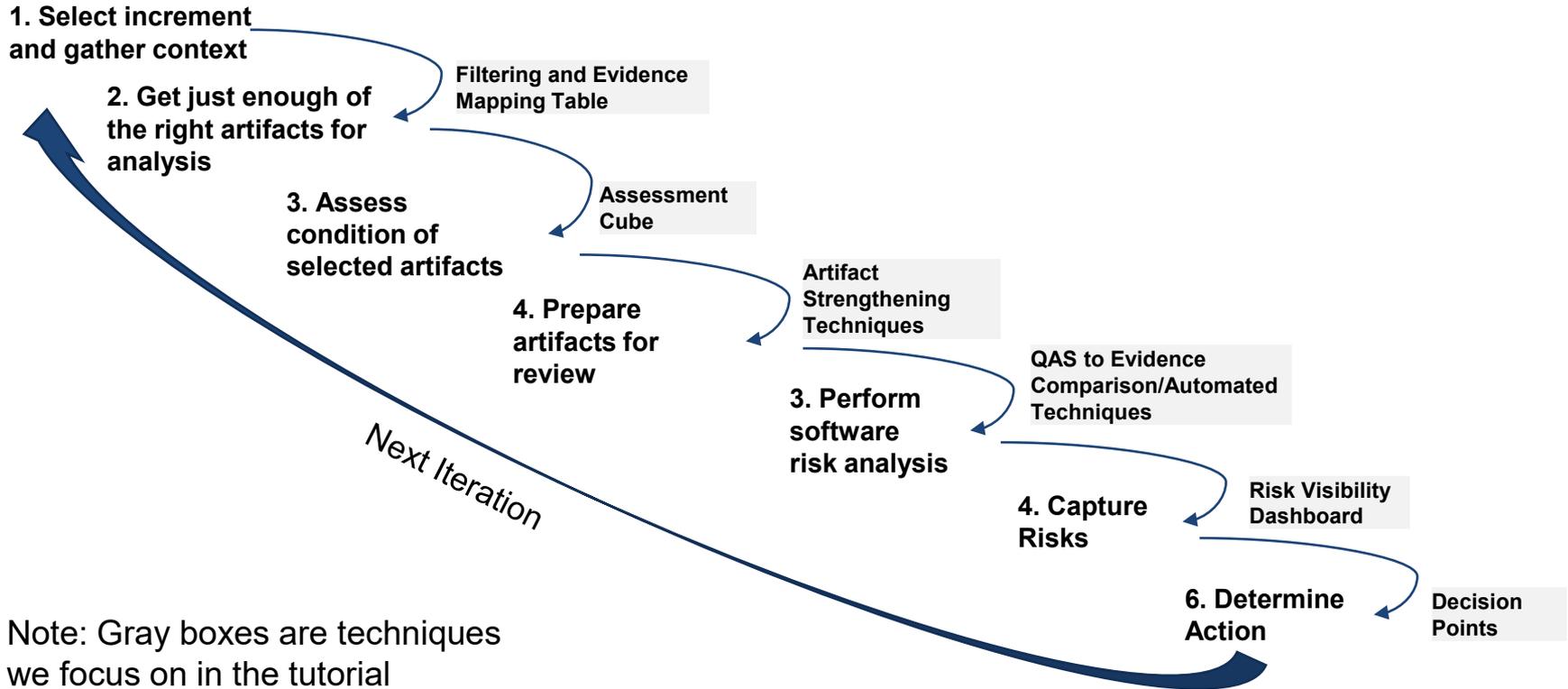
- **Architect:** Person responsible for analyzing software architecture and guiding teams toward decisions that align with stakeholder goals; the information in this tutorial can be applied by architects in small teams or large portfolios
- **Software Increment:** A release of software functionality, a feature, or a bundle of features
- **Design Review:** Opportunity to evaluate design decisions for a specified software increment; it is not a gating function, there is no implied formality, and a design review can be an architect and a team discussing a whiteboard or a formal group process
- **Analysis Artifacts:** Analysis artifacts provide evidence that a requirement is met; the type of artifact needed depends on the requirement; and analysis artifacts can include whiteboards, diagrams, code, and everything in between

Maintaining Alignment

- A key idea is to maintain alignment of requirements, design, and code as we help teams evolve the software
- Our approach to incremental design review encourages alignment but allows for flexibility with how alignment is achieved



Incremental Design Overview



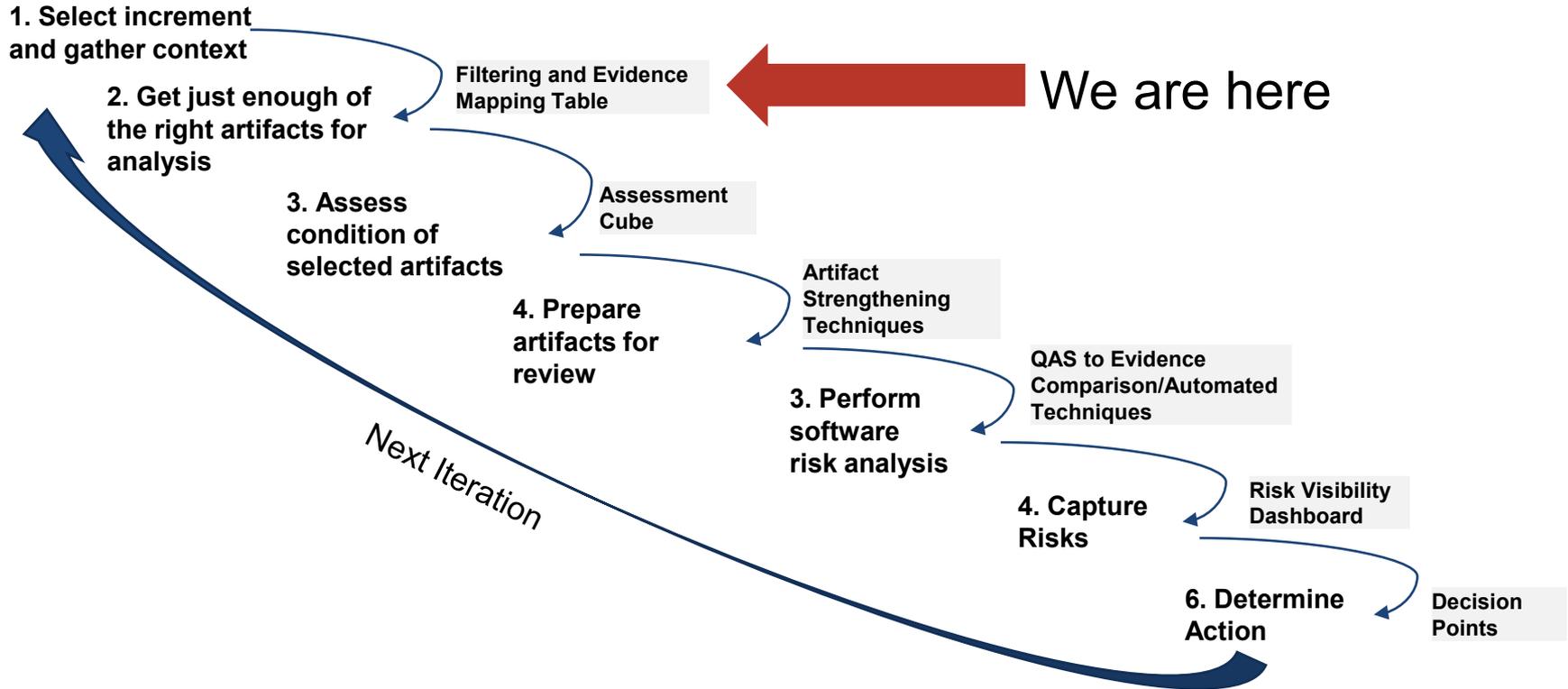
Note: Gray boxes are techniques we focus on in the tutorial

Select Increment and Gather Context

The first step in the incremental design process involves selecting an increment (with stakeholder input) and gathering context

- Increments selected for review should have architectural implications
- Rules of thumb for increment selection:
 - quality attribute consideration (we will discuss quality attributes in detail)
 - opportunity for significant change or refactoring
 - a capability the organization will build on
- Gathering context may require getting up to speed on technologies or frameworks used by the team in the increment

Incremental Review Overview



Filtering and Evidence Mapping

- Next step is to get just enough of the right artifacts for analyzing design risk
- With incremental development, it is not uncommon for increments to involve different technologies and be developed by teams with differing development styles and capabilities
- A variety of stakeholders ask teams to create all kinds of documentation
- The problem is that it is easy for teams to generate a lot of information that is not useful for analysis
- Meanwhile, architects can get buried in information
- Filtering is a technique to reduce information for analysis without losing important details

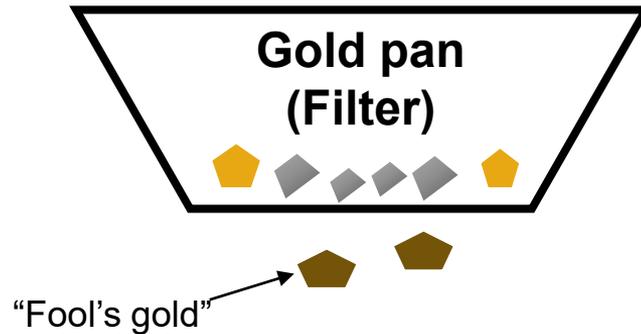
Typical Inputs

For example, typical inputs we get for a design review include

- detailed functional requirements
- UI mockups and/or business flow descriptions
- demos of working code
- detailed context or explanations of why the software is the way it is
- Repackaged, outdated design documentation
- organizational standards
- team objectives
- COTS overviews
- “market-ecture” illustrating the benefits of the design without technical detail

Avoiding the Fool's Gold

- Like miners sifting for gold during the gold rush, we often will get lots of rocks we don't need
- If not careful, we end up wasting time analyzing fool's gold

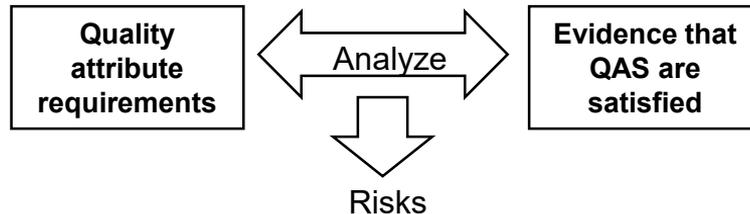


What Is Really Needed?

While many artifacts provide useful context, we mainly need two sets of inputs for an incremental design review:

1. quality attribute requirements
2. evidence that Quality Attribute Scenarios (QAS) are satisfied

During the review, we compare and analyze these artifacts to surface risks



The next few slides provide examples of these inputs

What Is a Quality Attribute?

- Quality attribute requirements describe the qualities of the software, such as performance, security, modifiability, and availability
- These are typically captured as quality attribute scenarios (QAS)
- The next slide shows some example quality attribute requirements

QAS Examples

Security

An authorized user pulls up a record in a system other than SystemABC and changes the notes. That change is recorded in the audit log in 99.9999% of the cases.

Reliability

While not connected to the network, a user adds/edits financial data. Connection is reestablished. All changes and attachments are sent to the server with no data consistency issues in 99.9999% of cases.

Flexibility

A business decides that the rule for who can change business notes should be extended. This change can be done on the production system within 15 minutes.

What Is Evidence?

Evidence is a subset of information

We only need to see a view that convinces the architect that the QAS is fulfilled

This further reduces the amount of information to analyze and reduces waste

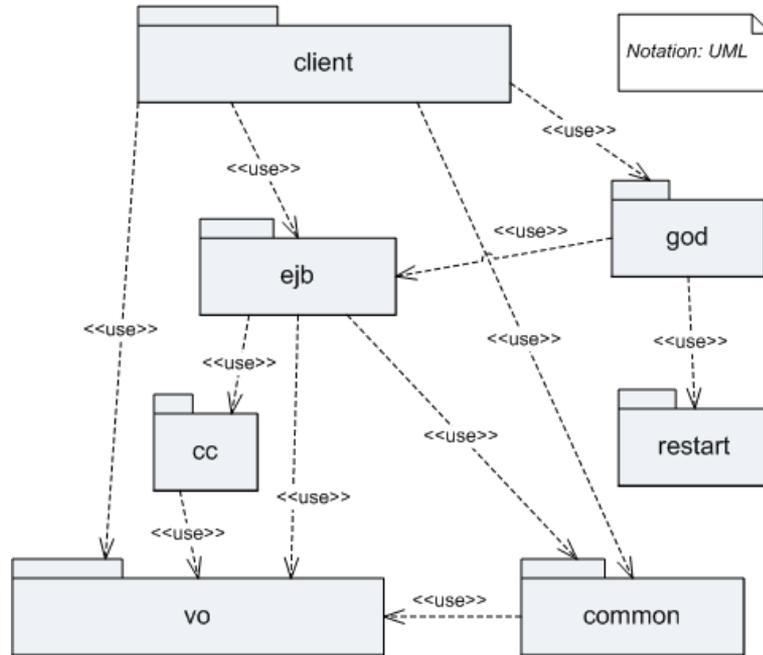
Artifacts for evidence analysis may come in many forms, such as

- design diagrams
- code
- performance test results
- whiteboard snapshots

As long as the evidence supports analysis of the selected QAS, it works!

Component Evidence: Example

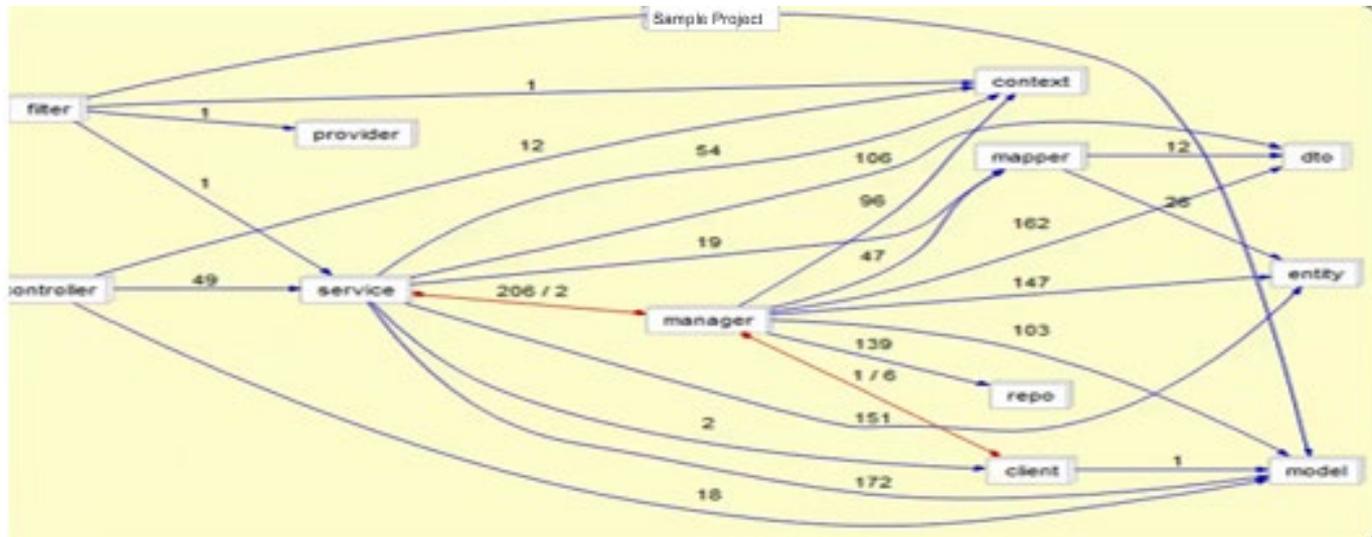
Property revealed by this artifact: Portability



Code Analysis Evidence: Example

Property revealed by this artifact: Maintainability

We can reason about whether changing one component may impact other components

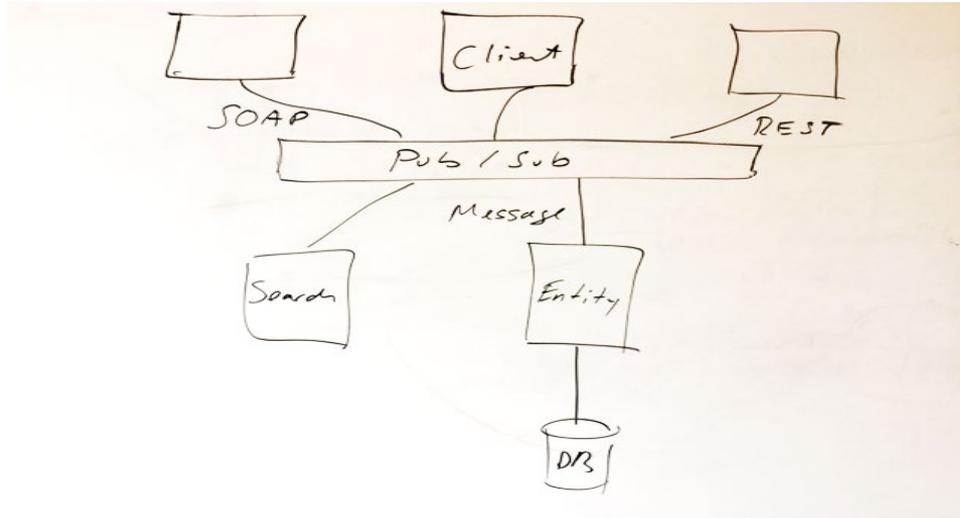


Keep in mind that when analyzing code we always stay at design level

Informal Evidence: Example

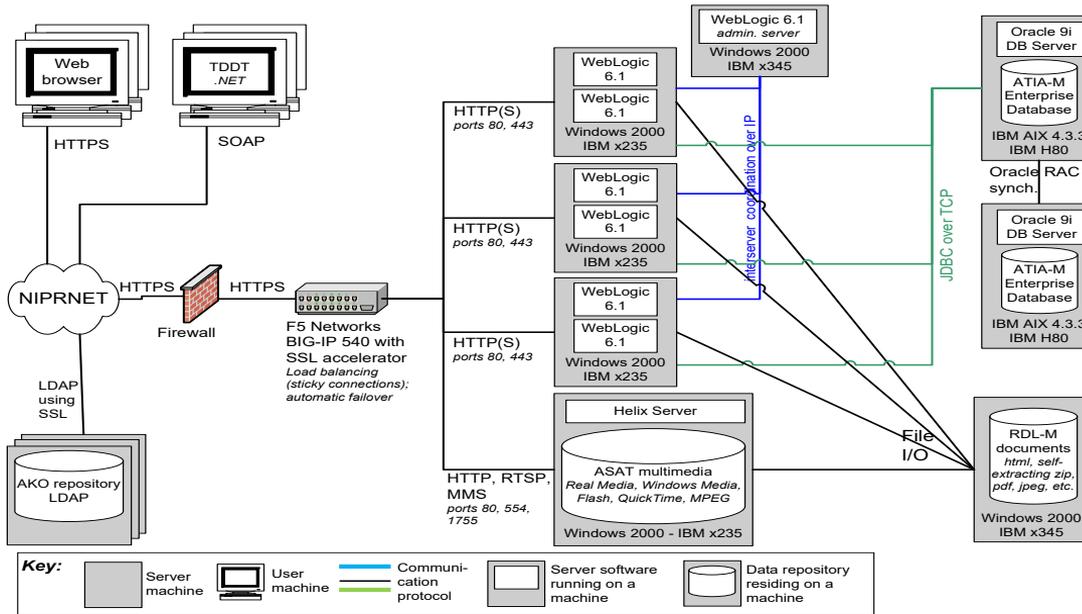
Property revealed by this artifact: Performance

Allows for reasoning about response time from the client to the database and back



Deployment Diagram

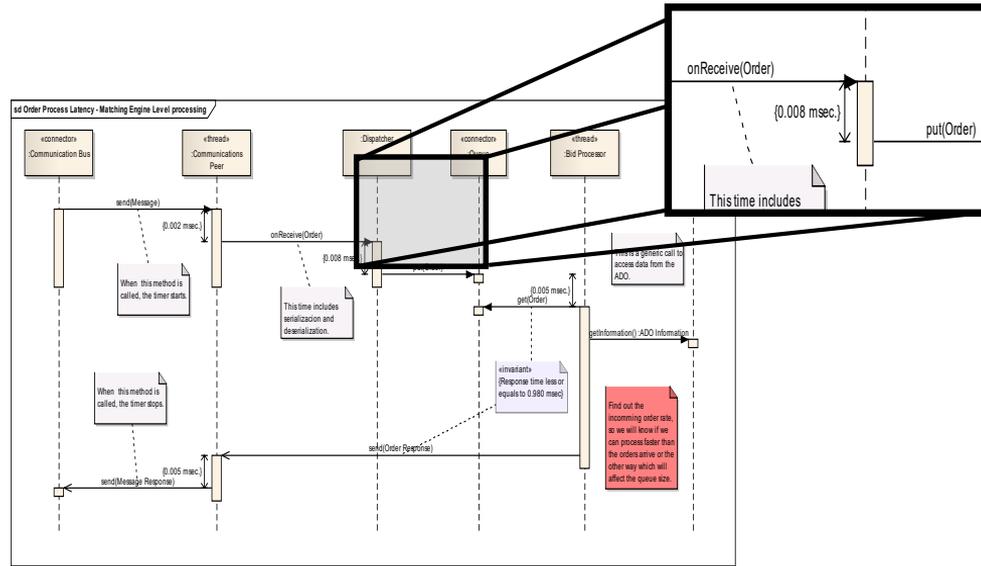
Property revealed by this artifact: Security



Sequence Diagram

Property revealed by this artifact: Performance

Enumerating the sequence diagram with time spent on each call allows for reasoning about performance bottlenecks



Test Output/Logs Evidence

Test output or logs can also be used as evidence for a subset of quality attributes

Property revealed by this artifact: Performance

Time (ms)	Type	Description
161	Java/DB	Tag[processReadXDataMsg in XXProcessor]
1	Java	Tag[processReadOperationDataMsg in XXProcessor]
168	Java/DB	Tag[Transaction ID -9999: readxDat in XXXProcessor]
2	Java	Tag[Transacction ID: convertLogictoXML for read xxresponse in XX processor]
176	Java/DB	Tag[ProcessReadxxDatMsg in XXProcessor]
36	DB	Tag[getNextTransactionID in XXFacadeBean]
68	Java	Tag[generateNextTranIC in XXProcessor]

Static Analysis Design Rules

We have started experimenting with filtering static analysis violations by “design rules”; this helps us focus on which parts of the software to analyze

Violations	Files
48	xServiceImpl.java
29	FindDocDialog.java
26	xDocEnterMetaData Comp.java
57	UpdateDocumentDialog.java

Type	Violations Grouped by Design Paradigm	Original Version (# Violations)	Refactored Version (# Violations)
DR	Complexity	8	0
DR	Exception Handling	8	0
DR	Logger	0	1
DR	Duplicated blocks	0	9
ND	Useless assignment	26	1
ND	Hardcoded constants	6	2
		48	5

For example, xServiceImpl.java files have several design rule violations

QAS to Evidence Mapping Table

Even after we filter down to QAS and evidence, we find there can be mismatches between what teams develop and what is needed for analysis

- A handy tool is the **Evidence Mapping Table**
- List the QAS and expected evidence; team commits to create these artifacts for the review

QA	Quality Attribute Scenario	Evidence
Modularity	Changes to the physical data structures will not impact business logic code	Component diagram
Security	While not connected to the network, a user adds/edits data. Connection is reestablished. All changes and attachments are sent to the server with no data consistency issues in 99.9999% of cases.	Deployment diagram (network/physical diagram)
Performance	System processes a message while processing 100 messages; system stores the data and responds with in 2 ms	Annotated sequence diagram or performance test results

Allocating QAS to Increments

In incremental design review, we don't analyze all the QAS at the same time

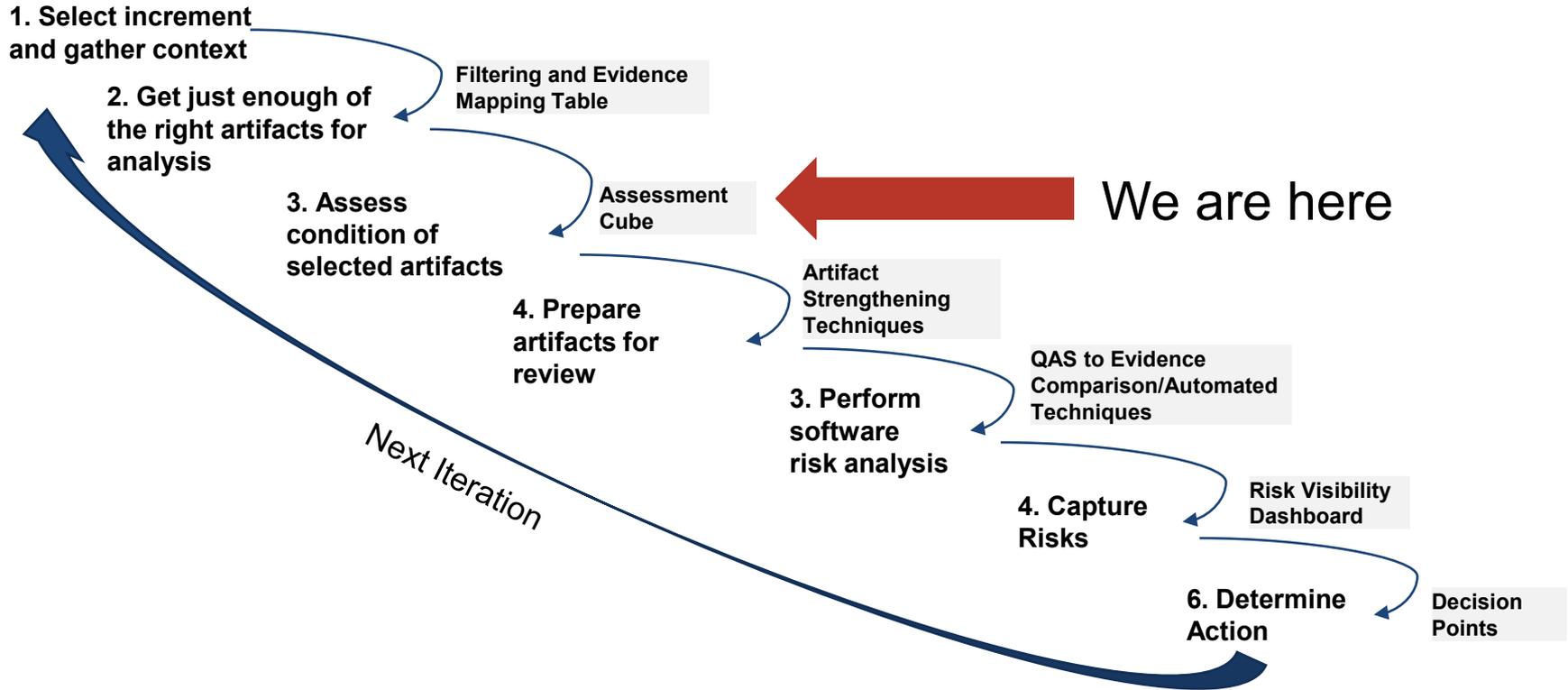
- We typically allocate 3–5 QAS to an increment
- Often teams will sprinkle the QAS increments between feature-only releases
- We try to align the QAS analysis with increments that make sense



Filtering Recap

- We explained that in an incremental development environment there is a lot going on and much information is generated!
- Filtering reduces the amount of information teams need to generate and architects need to analyze to surface design risk
- QAS and evidence are the main inputs needed for analysis; everything else may or may not be useful context
- Reducing to only QAS and relevant evidence is not enough; we need to also make sure we get useful information
- The QAS Evidence Mapping Table helps ensure we get information appropriate for analysis

Incremental Review Overview



Assessing the Condition of Artifacts-1

The next step in the incremental design review is to assess the condition of artifacts submitted by teams for design analysis

Why is this step needed?

We found that teams often show up to reviews with artifacts that are not adequate because

- the QAS are weak
- the design evidence artifacts are weak
- or both

If we (architects) don't recognize this, we waste the team's time and do the wrong things

Assessing the Condition of Artifacts-2

- Assessing the state of the artifacts increases the likelihood of a productive design analysis activity and helps architects determine what to do next
- We know that teams do not follow the same path to develop the analysis artifacts
- Sometimes they have given more thought to requirements than design or vice versa
- In the next few slides, we describe several paths we observe that teams take to get artifacts to the appropriate state
- We also describe techniques that architects can use to help teams strengthen artifacts regardless of the path they take

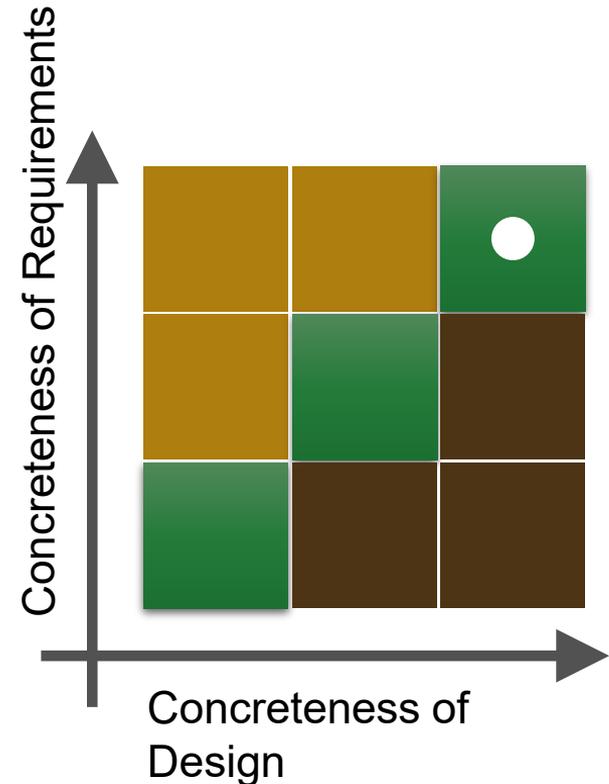
Assessment Cube

The Artifact Assessment Cube is a reasoning tool for determining the state of artifacts prior to analysis

The goal is to move the artifacts toward the top right corner

By evaluating the condition of input artifacts, architects can use the cube to determine what actions they should take next

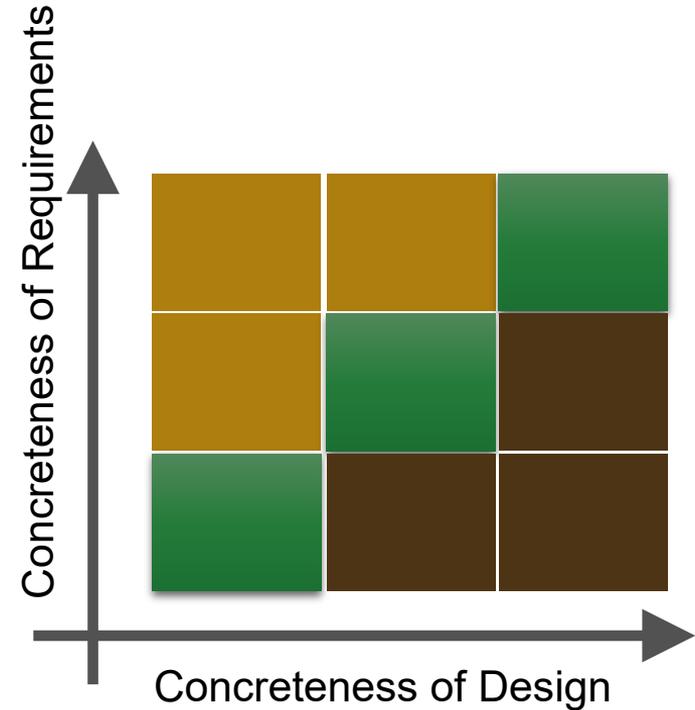
Note: We show the cube as a square here for ease of explanation; later we will explain the cube aspect



Artifact State Overview

There are three states that artifacts can be in:

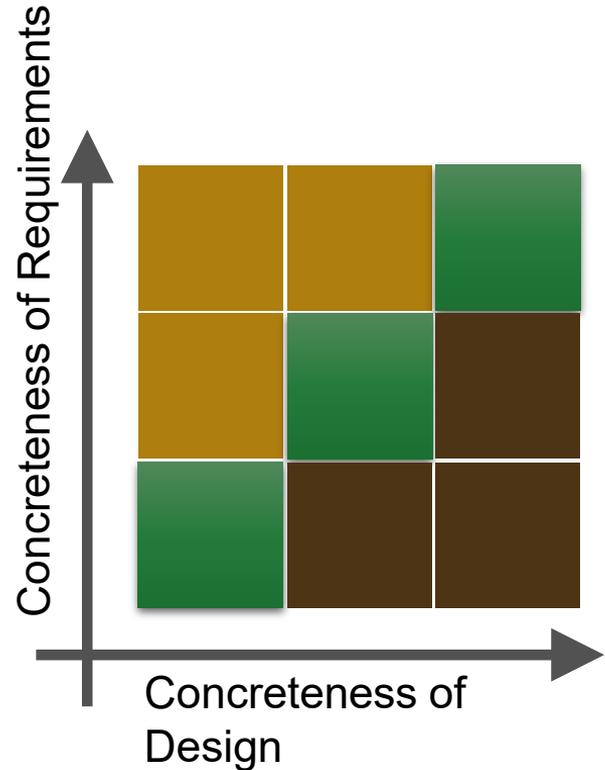
	Insufficient requirements Sufficient design
	Sufficient requirements for analysis Insufficient design for analysis
	Sufficient requirements for analysis Sufficient design for analysis



Factors for Determining What to Do Next

In this example, we assume that the scope of iteration is already chosen; steps the architect needs to take next depend on

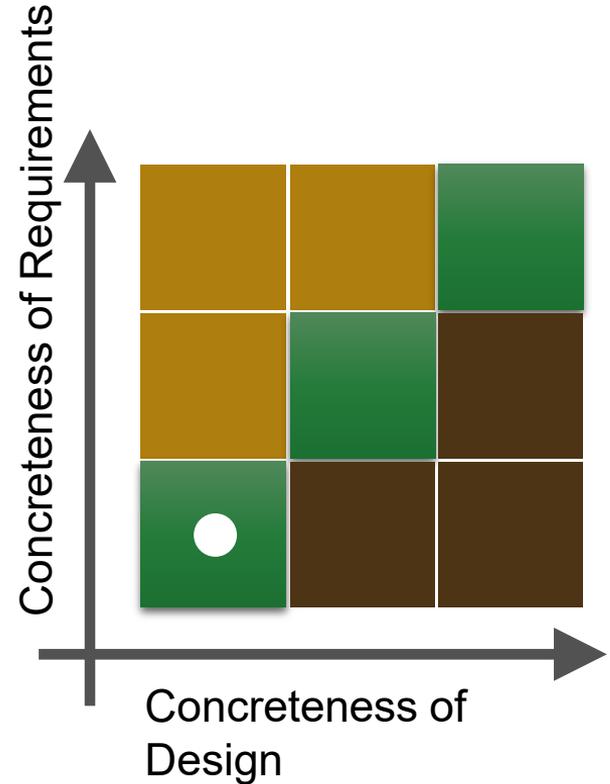
1. the concreteness of the requirements for the features of that iteration/release
2. the concreteness of the design of the solution



Beginning of Increment (Green)

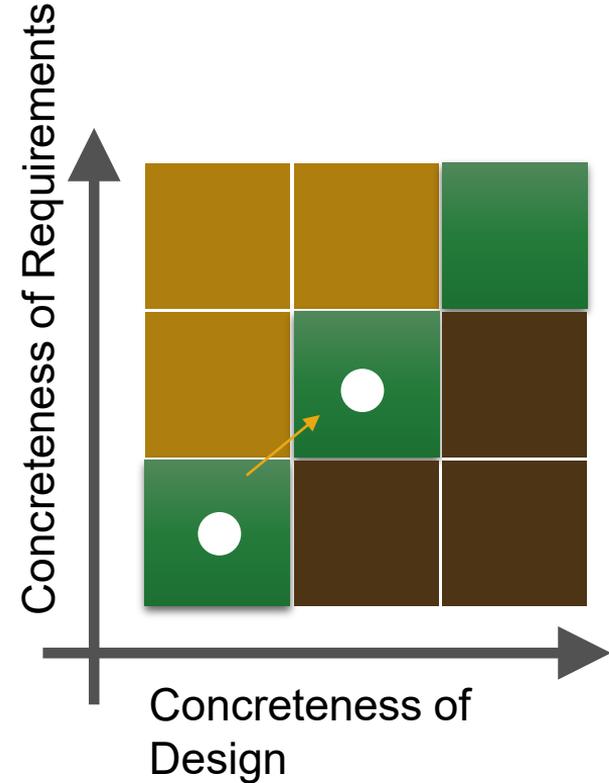
At the beginning of an increment (lower left corner),

1. QAS requirements are vaguely defined (e.g., only business goals)
 2. evidence describes some major decisions (e.g., frameworks), but some decisions have not yet been made
- If we find a risk here, it is likely a show stopper
 - Under certain conditions, it is appropriate to have a review
 - e.g., initial analysis to find major gaps in business goals (e.g., security risk)



Moving Up the Diagonal Green Path

- The further we move up the green diagonal, the greater the detail in the artifacts
- We work with teams in a collaborative way to get them to evolve the architecture
- We refer to this as a Sidecar Model of collaboration
- We have informal reviews at every step, but as we go up the risks become more concrete

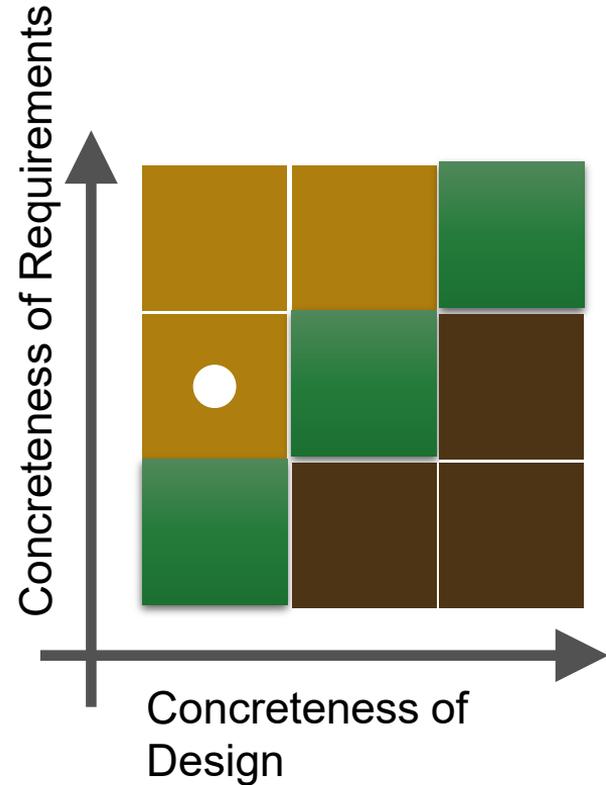


Beginning of Increment (Yellow)

At this time,

1. project goals and functional requirements are defined; quality attribute properties are defined (e.g., security) but not measures
2. design is still vague

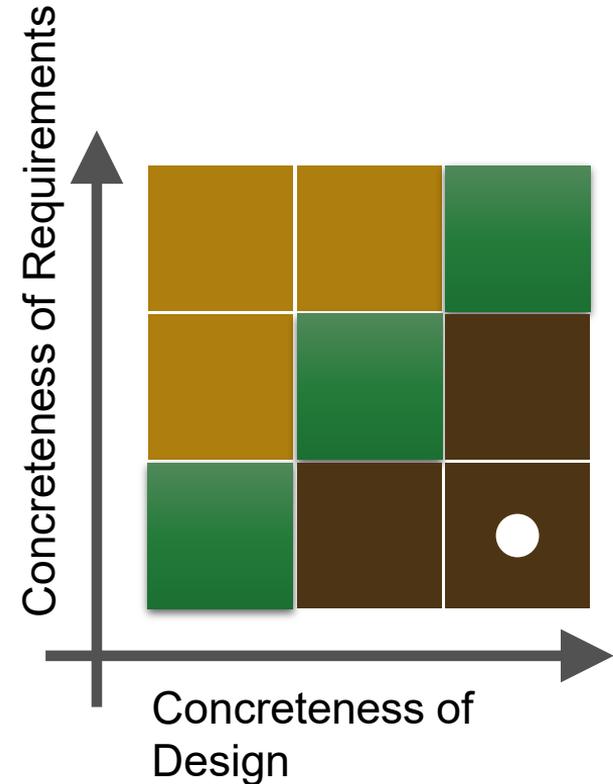
Review at this stage does not seem to be appropriate because of missing design decisions



Beginning of Increment (Red)

At this time,

1. project goals and functional requirements are poorly defined; QAS are not created
 2. there is a fairly specific design for the system/feature available
- If a team is in this state, it has made a lot of assumptions about what the requirements actually mean without validating those assumptions
 - This is a risky position to be in



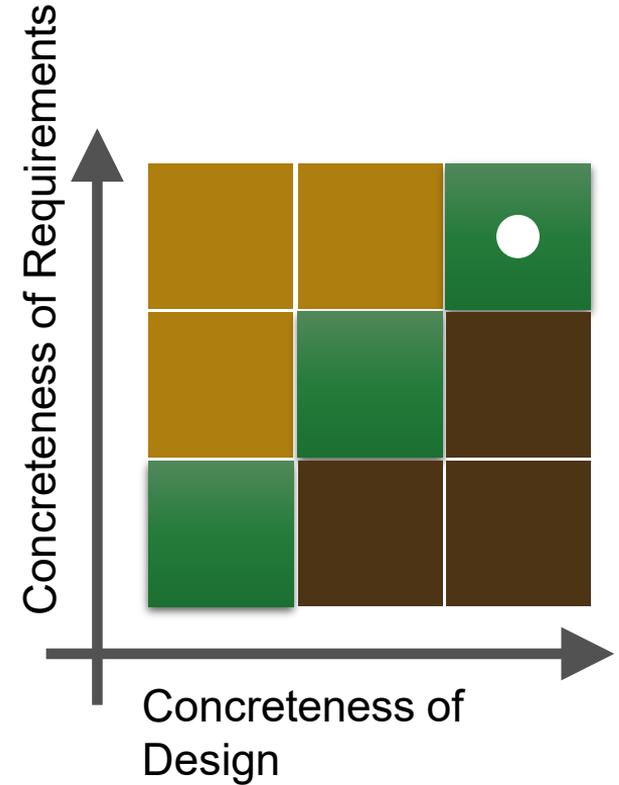
End of Increment (Green)

The “Happy State”

At this time,

1. measurable quality attribute requirements are defined
2. there is a specific design for the system/feature available

The design can be reviewed and will lead to concrete risks (if there are some)



Multiple Paths

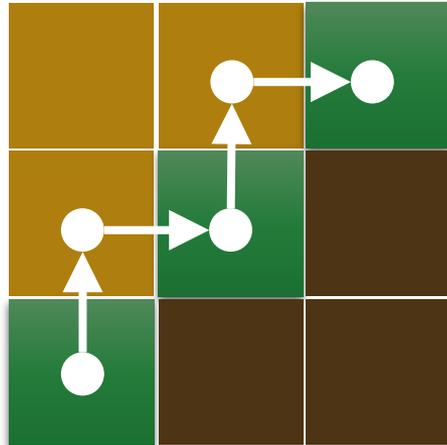
Teams may use multiple paths to reach the Happy State, which means they are ready for design review

The paths have trade-offs

Depending on the project context and goals, it may make sense to go with one path vs. another

The Iterative Path

This path makes sense when using an incremental lifecycle



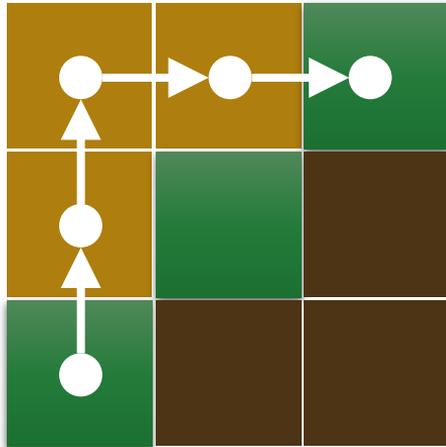
In this example, we have a new feature for which the requirements are not clear yet and it requires a new design

The team

1. decides to work toward a better understanding of the requirements
2. designs the solution while understanding the requirements
3. uses the gained knowledge to refine the requirements
4. adds the necessary details to the design

The Waterfall-ish Path

This may be appropriate in a safety-critical system where understanding and analysis of requirements are critical or the organization prefers this path

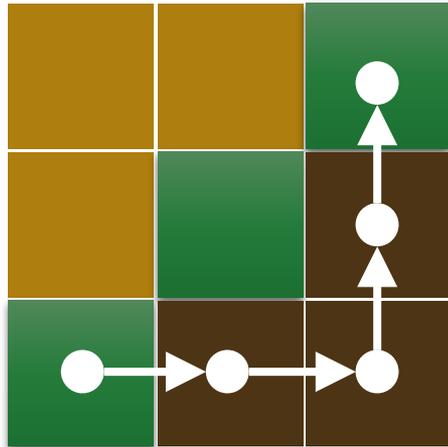


It is a new feature for which the requirements are not clear yet and it requires a new design

1. Team decides to work on the requirements first and get them to a well-understood state
2. After that, the team designs how the new feature should be implemented

The Prototype Path

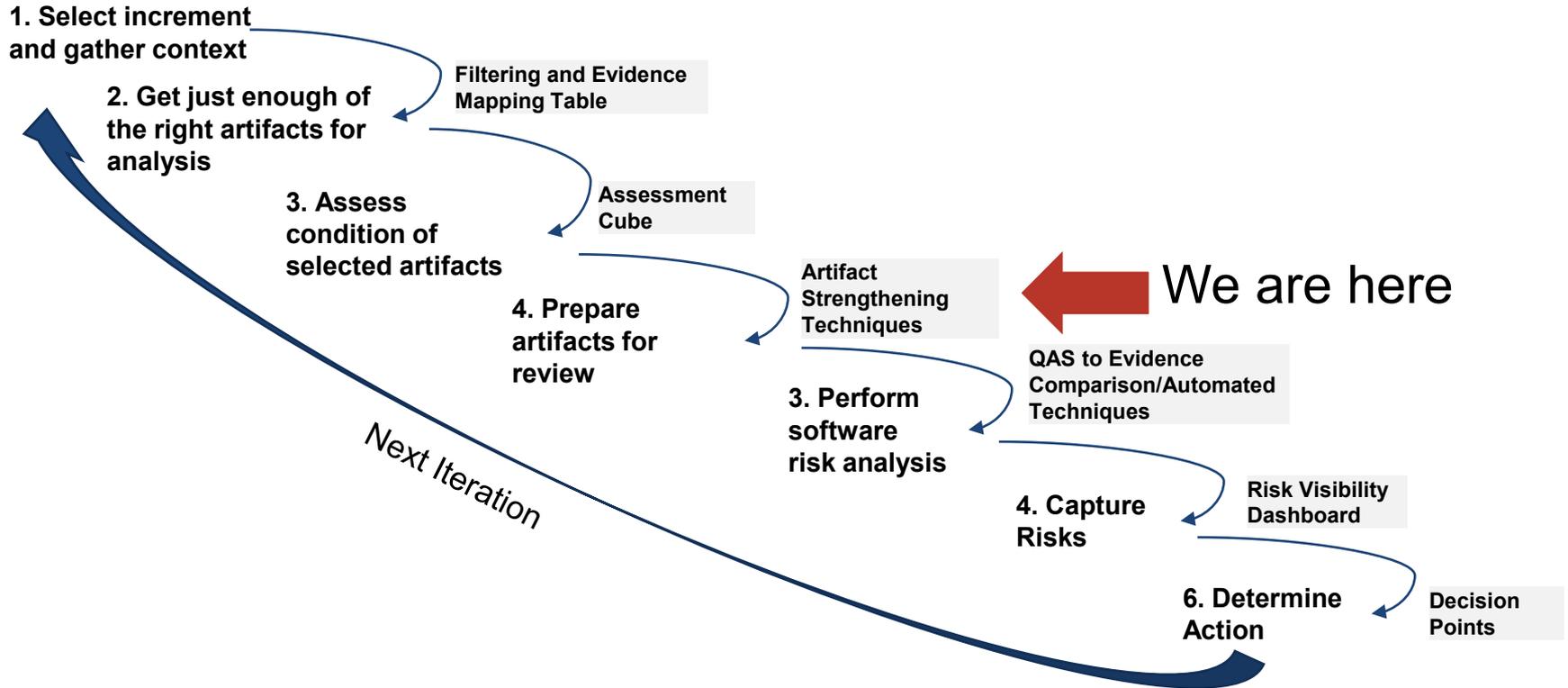
This path may be more appropriate for trying out a new technology (e.g., Agile spike)



It is a new feature for which the requirements are not clear yet and it requires a new design

1. Without understanding the requirements, the team goes ahead to design the solution
2. After the design is finished, the team starts understanding the requirements
3. Requirements might be refined to fit the design

Incremental Review Overview

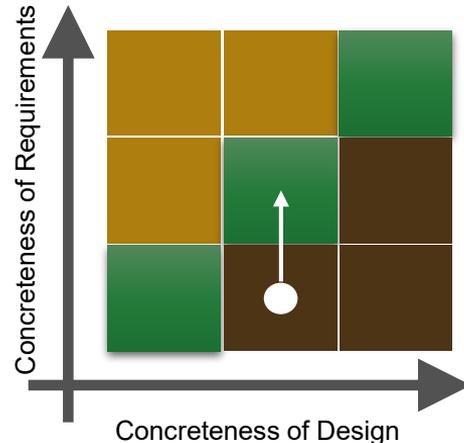


Artifact-Strengthening Techniques

Based on the current state, architects can use a variety of practices to help teams move artifacts toward the green, such as...

Artifact State:

- Insufficient requirements for analysis
- Sufficient design for QAS analysis



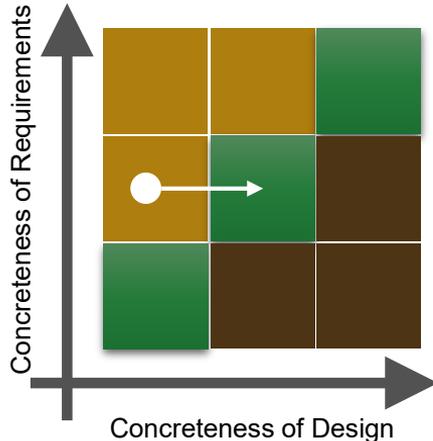
Artifact-Strengthening Techniques

- Extend functional requirements (e.g., user stories) as a basis for QAS
- Create “seed” QAS via doc review or active listening
- Conduct stakeholder workshop to develop QAS and measures
- Share metric examples from projects
- Query the organization for established measures

Architect Practices for Navigating the Cube

Artifact State:

- Sufficient requirements for analysis
- Insufficient design for QAS analysis



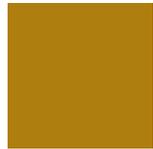
Artifact-Strengthening Techniques

- Create example evidence artifact with the team (e.g., architecture drawing tool, whiteboard)
- Conduct Options Workshop if decisions need to be made
- Provide iterative artifact feedback
- Conduct design review dry run
- Skill coaching in design analysis and tool usage (e.g., UML, dependency analysis)
- Share examples from other projects

Class Exercise-1



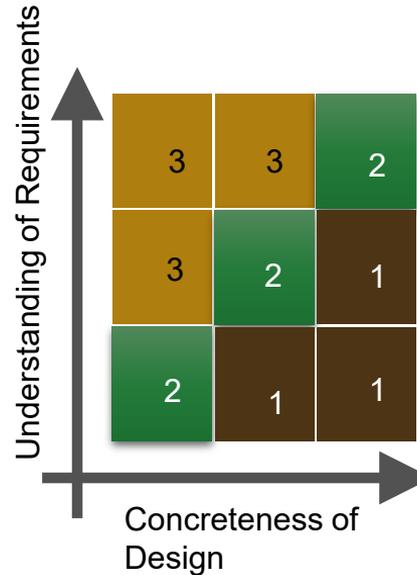
Insufficient requirements
Sufficient design



Sufficient requirements
Insufficient design



Sufficient requirements
Sufficient design



Exercise1:

Raise your hand if the teams you work on have artifacts mostly in

- State 1
- State 2
- State 3
- N/A

Class Exercise-2

Artifact State	Problem	Architect Activities
Insufficient requirements	Missing QAS	1. Use functional requirements to derive QAS
		2. Create “seed” QAS via doc review or active listening
		3. Conduct workshop to develop QAS
	Lacking QAS Measures	4. Facilitate sessions with team and business
		5. Share past project metric examples
		6. Query organization for established measures

Exercise 2 (1)

1. Do you use any of these practices?
2. How well do they work?
3. Do you use others not listed here?

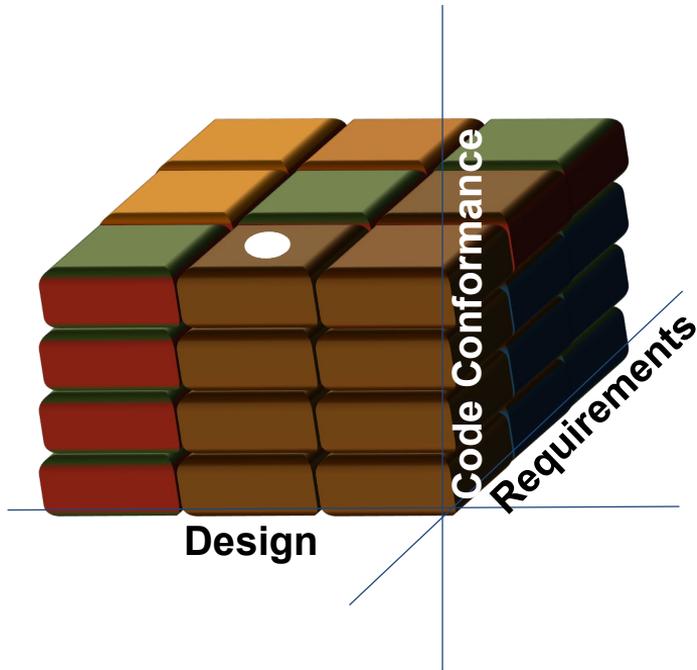
Class Exercise-2

Artifact State	Problem	Architect Activities
Insufficient design	Wrong evidence	1. Create example evidence artifact with the team (e.g., architecture drawing tool, whiteboard)
	Options	2. Conduct Options Workshop to make a design decision
	Incomplete design	3. Provide iterative evidence feedback
		4. Give bite-sized skill builder sessions (e.g., UML, dependency analysis)
		5. Provide examples from other projects

Exercise 2 (2)

1. Do you use any of these practices?
2. How well do they work?
3. Do you use others not listed here?

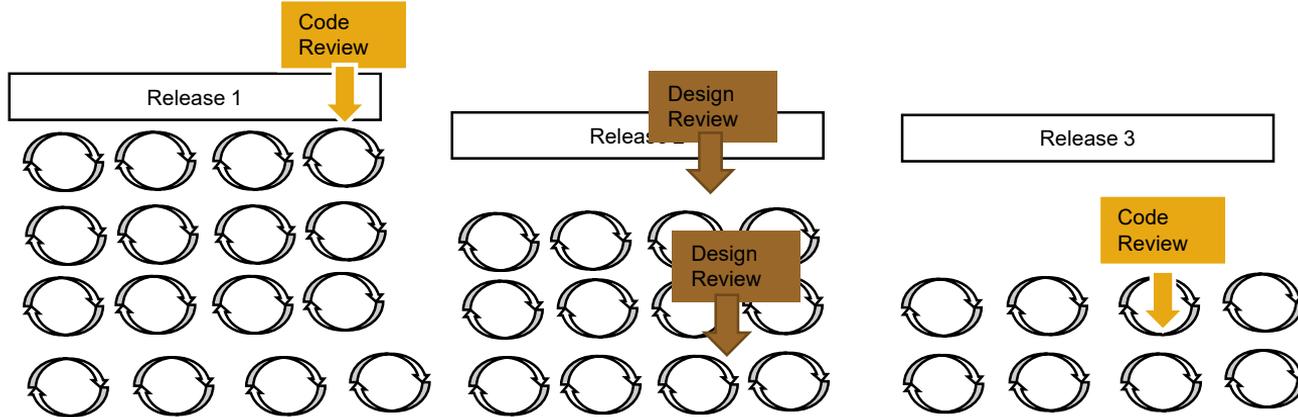
The “Cube” Actually Is a Cube



- The third dimension is code conformance
- It is important that code aligns with design and requirements
- In early stages, we might have minimal code
- As we move up the cube, the amount of code increases

Multiple Projects

- Even within one project, multiple developments can go on in parallel where the teams are in different stages
- You might have to coordinate several projects to create a new product
- You might be responsible for coordinating applications for an enterprise
- Regardless, we can quickly figure out where teams are and what to do next



Enabling Scalability with the “Cube”

Characteristics of the “cube concepts” that scale during this phase include

- inputs at an adequate level of specificity for the analysis activity
- helps architects do the right things
- non-linear process: no dependencies on previous steps/gates; you can apply on your desk tomorrow

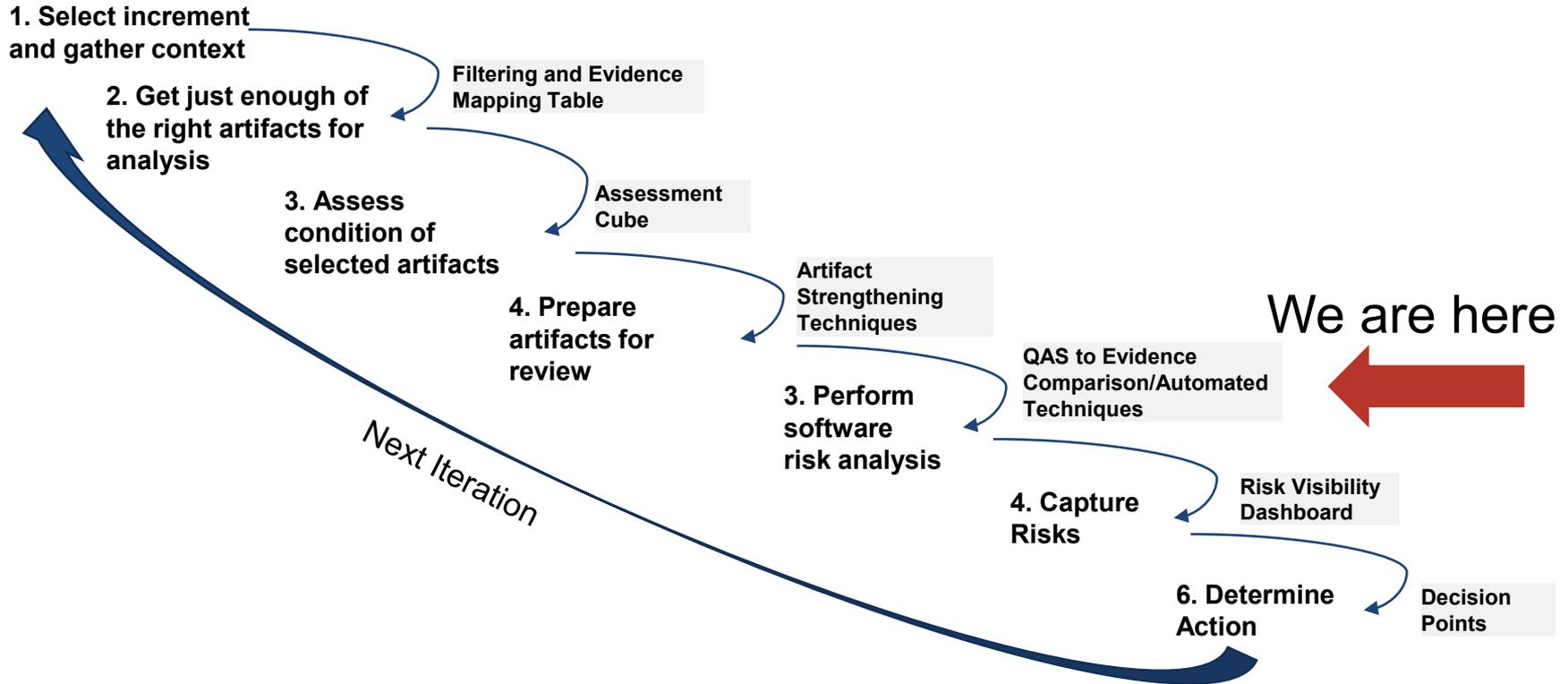
Continuous collaboration also helps

- We refer to this as the Sidecar Model
- **Benefits:** Reduces wasted time due to context switching, builds trust, allows early detection of problems, enables easier sprint planning, and has greater flexibility

Artifact Assessment Recap

- We shared the Artifact Assessment Cube concept, which is a reasoning tool for assessing the state of a design review artifact quickly and efficiently
- We explained that this is useful for three things:
 1. helping teams along the way
 2. helping architects know what to do next
 3. serving as a pre-design review checkpoint
- We practiced artifact assessment in an exercise
- We talked about activities that architects can do to close artifact gaps

Incremental Review Overview

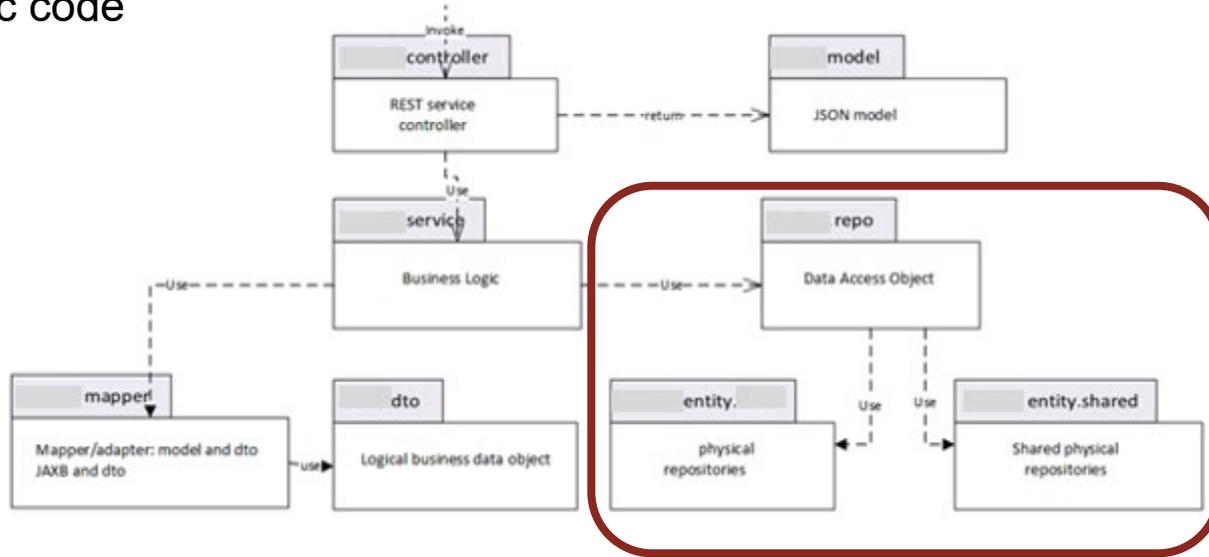


Design Analysis

- The main activity during the design analysis step is comparing evidence artifacts to QAS for analysis
- The main output is a list of software risks or TODOs
- The next slide shows an example comparison that happens during design review

Design Risk: Example

Non-functional Requirement: Changes to the physical data structures will **not impact** business logic code



Evidence Description: Changes to the physical tables require the repository (repo) component to change; the business logic components are likely not impacted

Design Risks: Examples

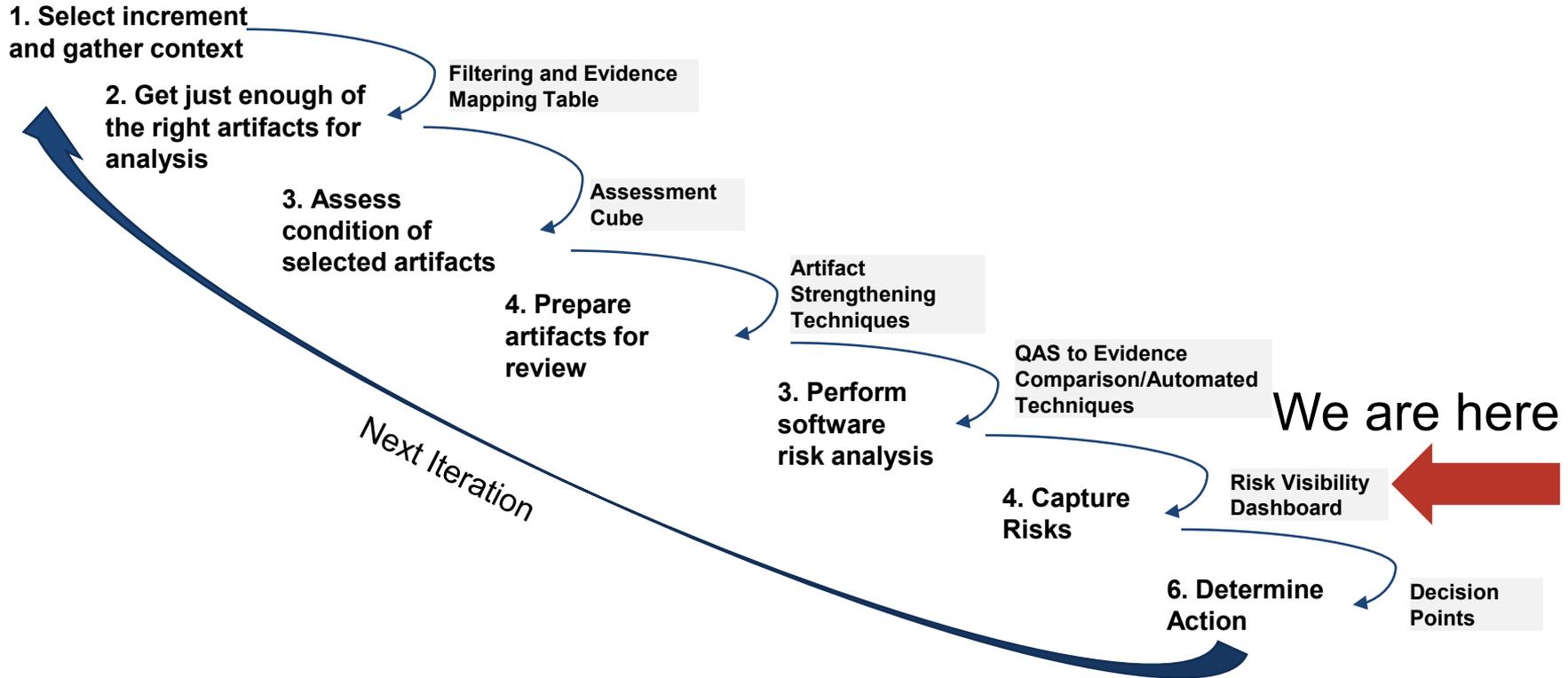
Risks

- Lack of clarity around responsibilities for how developers should use modules in Spring framework that implement the Model–View–Controller pattern may result in maintainability issues and developer confusion.
- Intranet Service X assumes that access control checking is done prior to calling XYZ Service; however, this leaves a security gap because an unauthorized user could call the service
- The enterprise service bus is a single point of failure; therefore, the team should consider adding a redundancy or failover tactic

TODOs:

- Investigate other options for mobile that support non-Windows platforms
- Based on the evidence provided, there is not sufficient evidence that QAS performance measures are met; provide performance testing metrics

Incremental Review Overview



Visibility Promotes Action

- The goal of this step is to get action on important risks
- The best way to motivate action is not to tell people what to do; it is to make the data visible and let it speak for itself
- If people can see it, they are more motivated to make changes
- We don't want an information stream that overwhelms
- During the design review, we bring up only what is most important
- These are classified as risks or TODOs

Visibility and the Architect's Role

It is the architect's responsibility to make sure something happens

Architects typically focus on fixing things

Rather than having architects take on fixing all the design risks, the work becomes making it visible to encourage teams to fix it

By making risks visible, we mean the following:

- Risks are written down
- Risks are stored in an accessible repository (we don't care what kind)
- Risks are communicated to developers
- Risk activity and status are monitored
- Risk data is viewable in such a way as to encourage action (e.g., dashboard)

Where to Put Design Risks

We suggest putting risks in the project backlog

However, for important risks we also recommend a software quality design risk repository and dashboard where open risks are visible

The reason for this is that at the project level feature development is typically prioritized higher than design risks and technical debt

Consequently, the design risks and technical debt items fall to the bottom of the project backlog and no action is taken

Design Risks and Technical Debt

In practice, teams frequently choose (often for good reasons) to delay fixing design risks

This can result in technical debt, which is risk that accumulates effort, money, or time

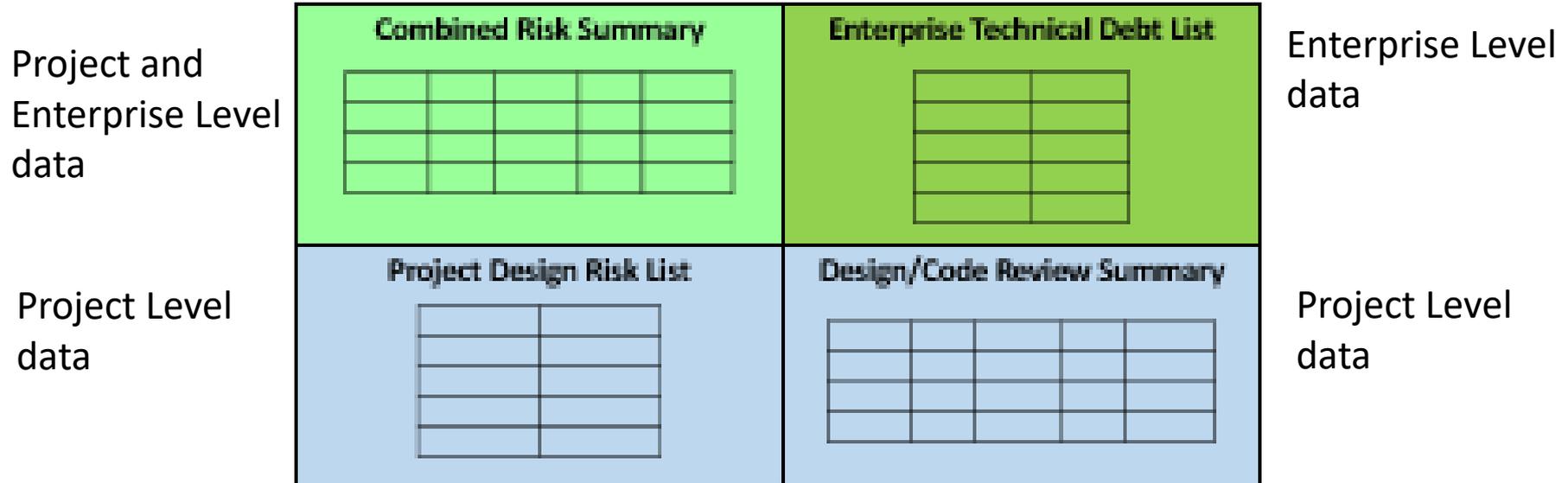
In our experience, these debts are very likely to be forgotten and the debt never paid down because there is no practice in place for doing so

Therefore, it is particularly important to make technical debt visible at the project and enterprise levels

Software Architecture Dashboard

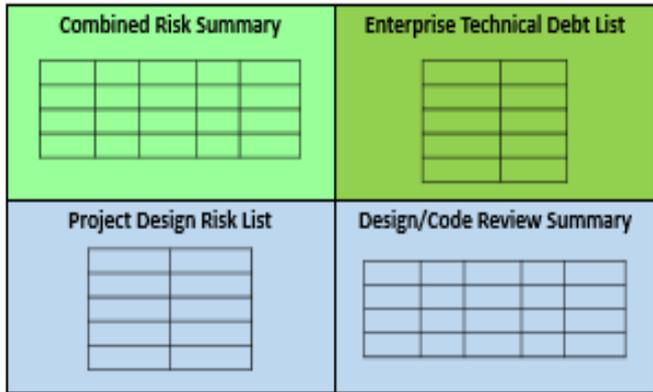
Our Software Architecture Dashboard has 4 parts; The bottom two quadrants are project level and the top two quadrants are enterprise level

We only focus on the project level (blue) for this talk



Dashboard – Project Design Risk List

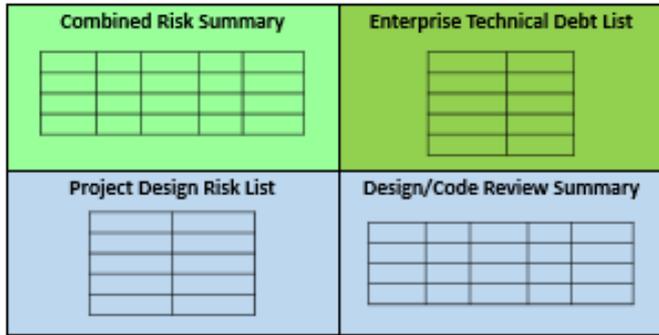
The Project Design Risk List shows the project design risks



ID	Project Risk Description	Project
SWQ-114	Data object definitions in Service belong in Utility	Project A
SWQ-108	Role/permission architecture is not flexible	Project B
SWQ-106	Use inheritance to get rid of unnecessary classes	Project B
SWQ-99	Design missing integration with Secure gateway	Project C
SWQ-77	Reduce branching complexity in xxDetailMgr.java	Project B
SWQ-71	Performance issue due to calls to database	Project B
SWQ-62	Duplication of data requires synchronization	Project D
SWQ-35	Add a notification component	Project E
SWQ-105	Manager and Model dependency	Project B
SWQ-76	Investigate JPA for complex SQL stmts in validation	Project B
SWQ-72	Move business rule logic to service layer	Project B
SWQ-36	Clarify data transformation responsibility	Project E
SWQ-27	Define IDs to map extranet records to intranet	Project E
SWQ-16	Use repo component via a manager	Project B

Dashboard – Design/Code Review Summary

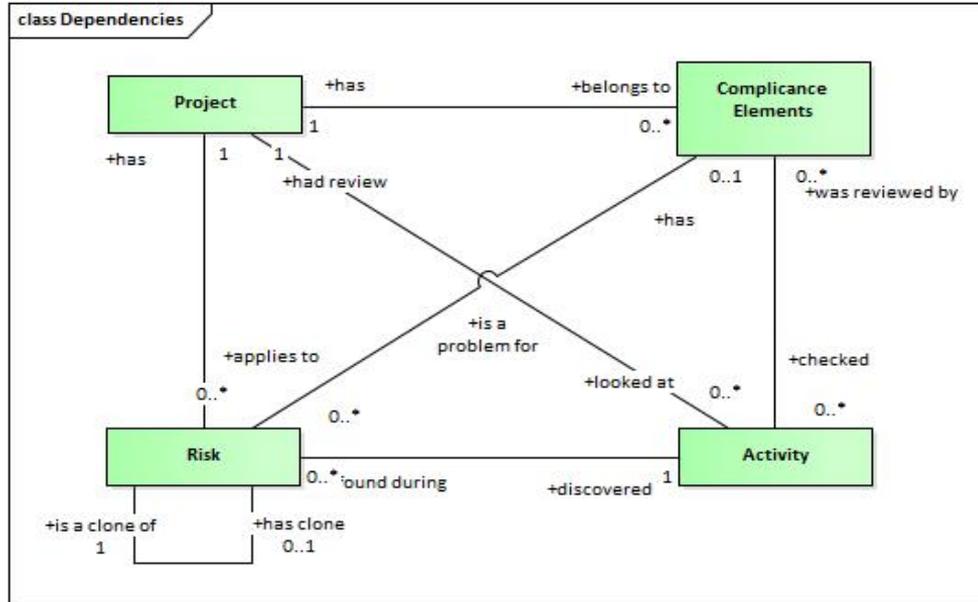
The Design/Code Review a project-level roll up of open issues by design or code review activity



Review Name	Design Risks	Code Risk	Technical Debt Item	TODO
Design Review 09-16-18	0	4	2	2
Code Analysis 10-13-18	7	2	4	1
Design Review 11-20-18	2	0	3	3
Design Review 11-26-18	9	3	0	2
Code Analysis 01-12-19	5	9	1	5
Design Review 02-09-19	0	3	0	4
Code Analysis 03-24-19	6	7	0	2
Design Review 04-03-19	7	0	1	0
Design Review 06-18-19	7	1	2	1

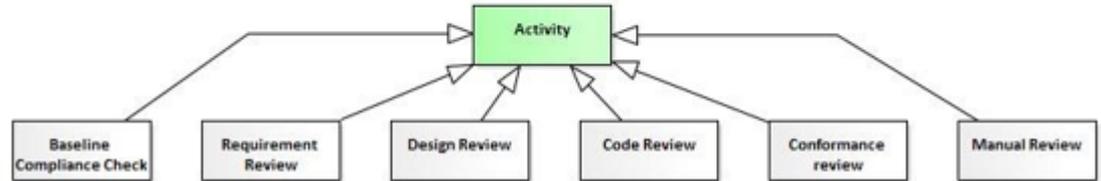
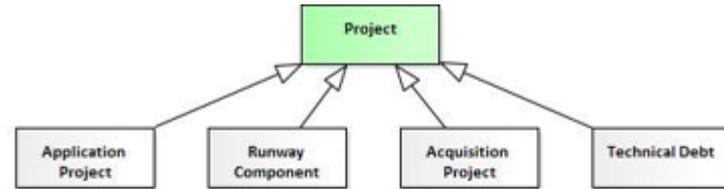
Dashboard Metadata Overview

The purpose of this structure is to track review activities and risk across multiple projects
Summarized data risk and activity can be used to monitor system/application quality



Projects and Activity Relationships Expanded

- The model is flexible and easily expanded
- There may be different types of projects
 - These are some that we commonly have
- We (architects) also support many types of activities



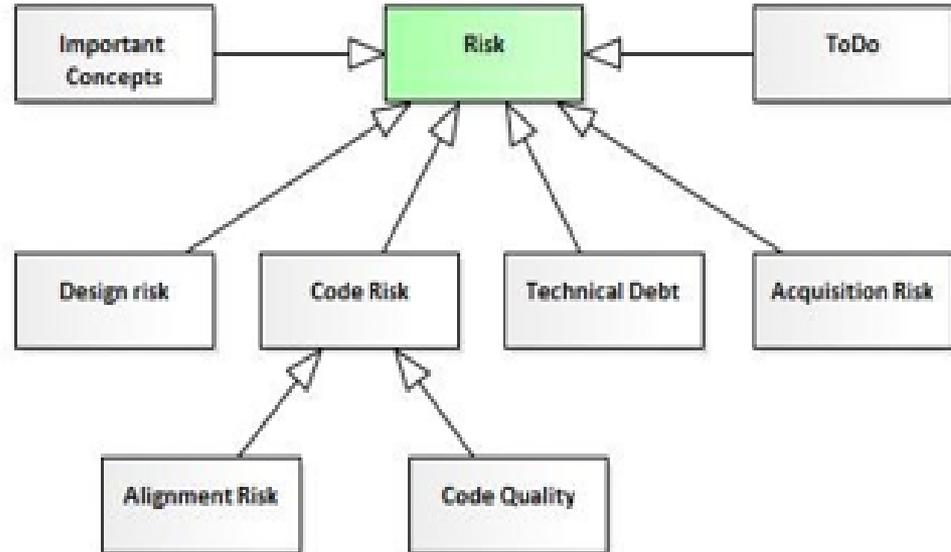
Risk Types

The risk metamodel is also flexible, allowing for handling many types of risks

The most common risk types we use are

- design risk
- code quality
- ToDo
- technical debt*

*We discuss this in a separate session



Format for Design Risk

Try to use a consistent format for capturing design risk, such as

- problem statement
- consequence
- proposed solution

Consequence allows us to

- argue for high priority if appropriate
- reason about whether the risk is a technical debt item (if effort or cost will accumulate, it might be)

Design Risk: Example

Problem: Lack of clarity around responsibilities for how developers should use modules in Spring framework that implement the Model–View–Controller pattern.

Consequence: Developers put business logic in Spring framework components inconsistently. When it is time to change the business logic, developers have to spend additional time analyzing the code, and there is a high risk of making a change that has unintended consequences. It is likely that effort and cost to maintain the software will go up over time, particularly because this pattern is intended to be reused across multiple projects.

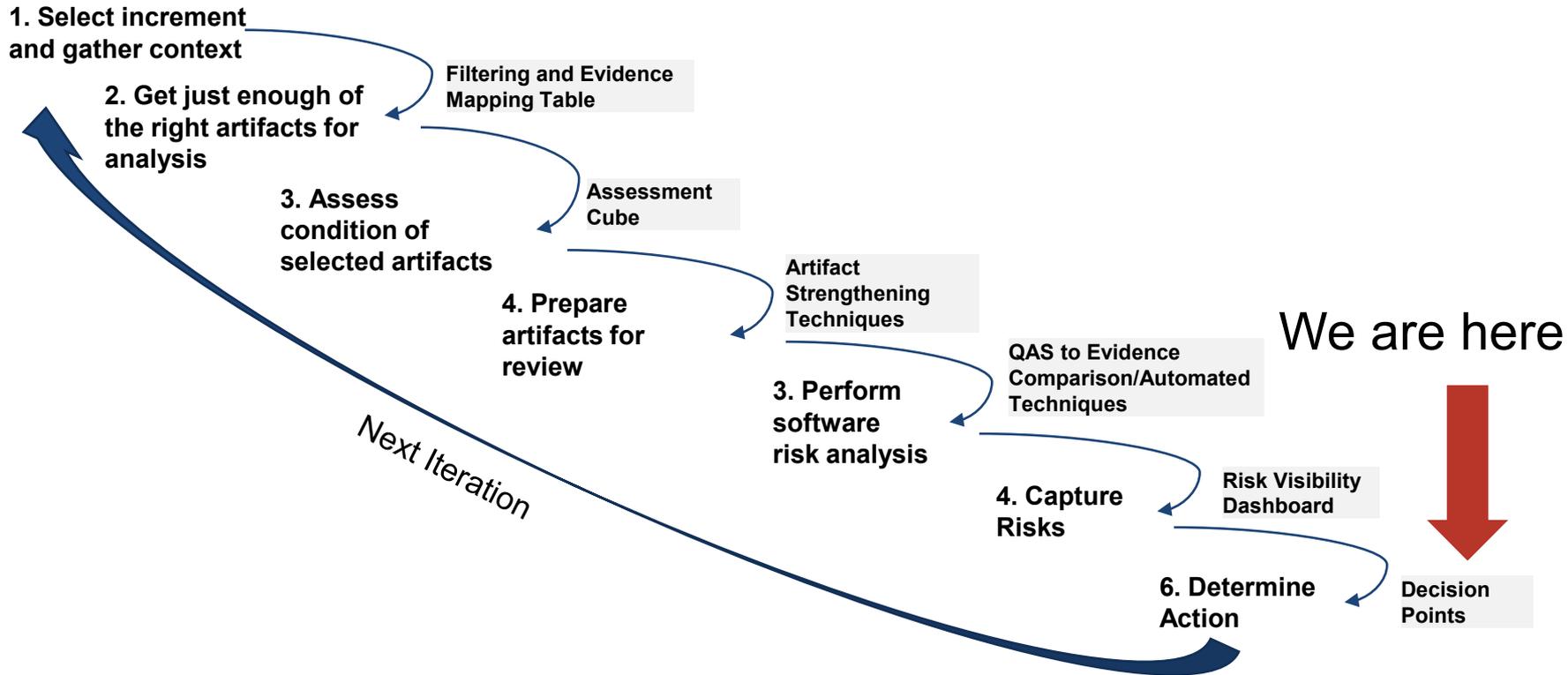
Proposed Solution: Include component responsibility descriptions in the software design document that clearly specify where business logic goes. Check for conformance to these responsibility descriptions during code analysis.

Tracking Progress in Incremental Design

- It is not enough to track only risk status (e.g., open, closed)
- It is also helpful to track steps in the incremental review process to see if teams are getting stuck
- Each row in the table maps to an increment
- A project may have multiple increments

1. Establish ideal software quality activity plan for year	2. Conduct Education and Verify Understanding	3. Collect materials (e.g., problem statement, constraints, design diagrams)	4. Agree upon QAS and evidence criteria with teams	5. Collect Principles conformance	6. Initial design/code materials review	7. Conduct design review prep session	8. Conduct Architecture Evaluation team meeting	9. Conduct Design or Code Review	10. Log risks and TDRs
1	1	3	2	3	2	2	3	3	3
2	1	1	1	1	1	1	1	1	1
1	1	2	2	2	3	3	3	3	3
1	1	3	1	3	1	1	1	1	1
1	1	2	2	2	2	2	2	3	3
2	2	2	2	2	2	2	3	3	3
2	3	3	3	3	3	3	3	3	3
1	3	3	3	3	3	3	3	3	3
1	4	4	4	4	4	4	4	4	4
2	3	3	3	3	3	3	3	3	3
2	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4
4	4	4	4	4	4	4	4	4	4

Incremental Review Overview



Motivating Action Through Decision Points-1

Making design risk data visible is good, but sometimes this alone is not enough

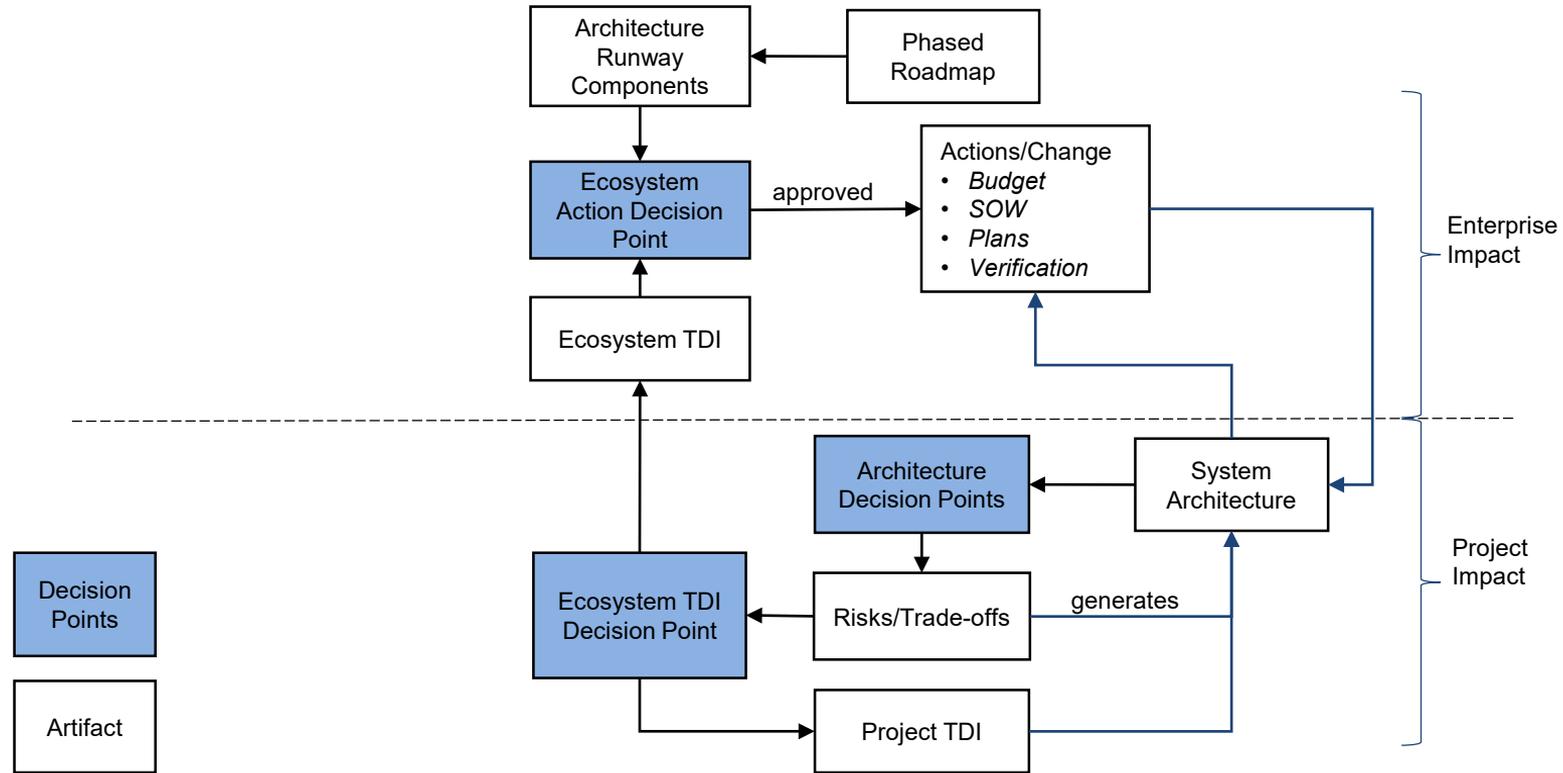
For this reason, we found it necessary to identify decision points

Decision points are a forcing function for stakeholders to come together to make a decision

A good example is a stakeholder meeting or call to discuss project design risk

The next slide explains some of the decision points in our process and why they matter...

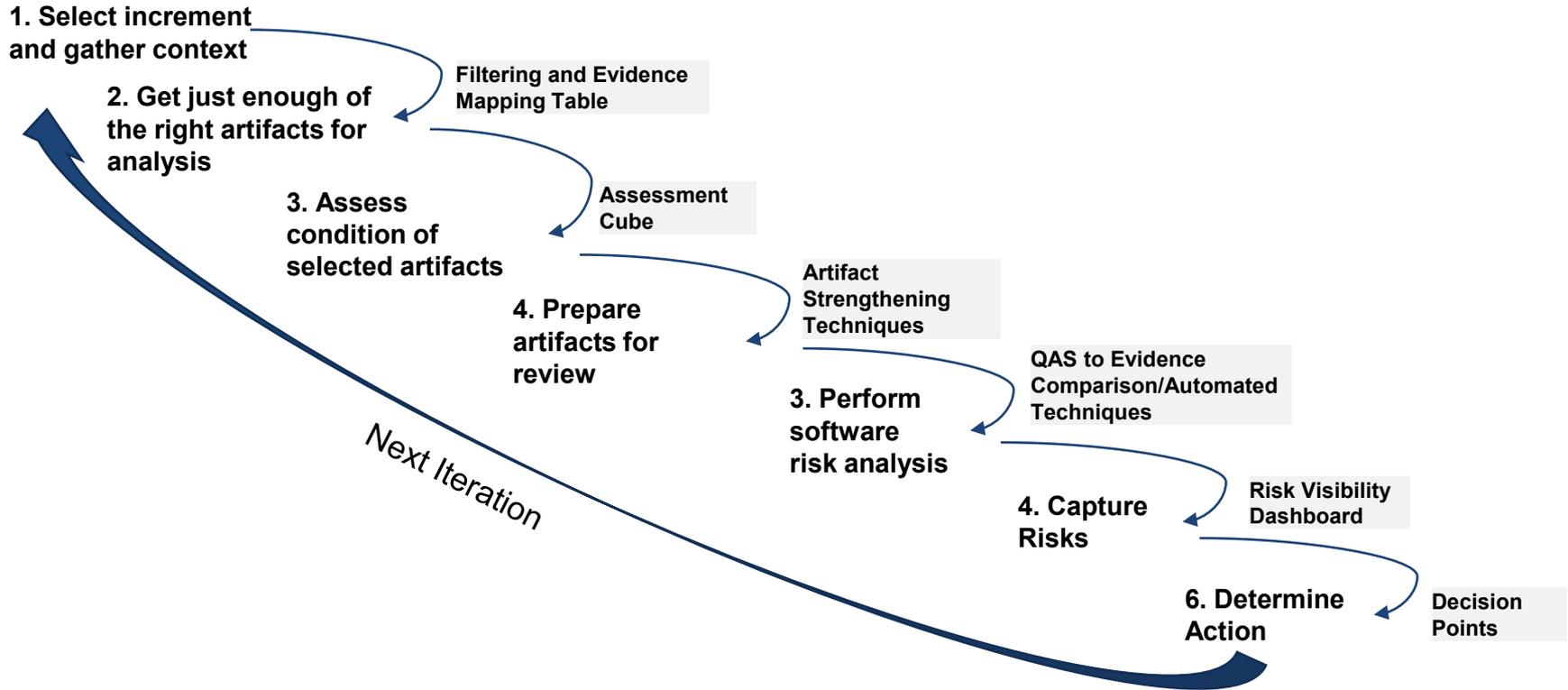
Motivating Action Through Decision Points-2



Summary

- The goal of a design review is to produce quality software
- What is quality software? Software without major design risks!
- To do this, we need to keep requirements, design, and code in alignment
- Our incremental design process is designed to maintain alignment for analysis while keeping up with a fast pace and project increments in different stages of evolution
- We introduced techniques we use to support the incremental design process
- Techniques we shared include filtering, evidence mapping table, assessment cube, risk dashboard, and decision points
- Finally, we find that the incremental design process provides the greatest impact if architects work collaboratively with teams in a continuous, ongoing manner rather than use design reviews as a gating function

We Made It!



Contact Information

Stephany Bellomo

sbellomo@sei.cmu.edu

Felix Bachmann

fb@sei.cmu.edu