

**SATURN 2019**

# Detecting and Tracking Enterprise Technical Debt

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 Defense 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.

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](mailto:permission@sei.cmu.edu).

DM19-0470

# Bottom Line Up Front

We have been working for a large organization for 2 years, conducting design reviews of a large portfolio of projects

We also work in the area of technical debt research

During project design reviews, we routinely surface technical debt

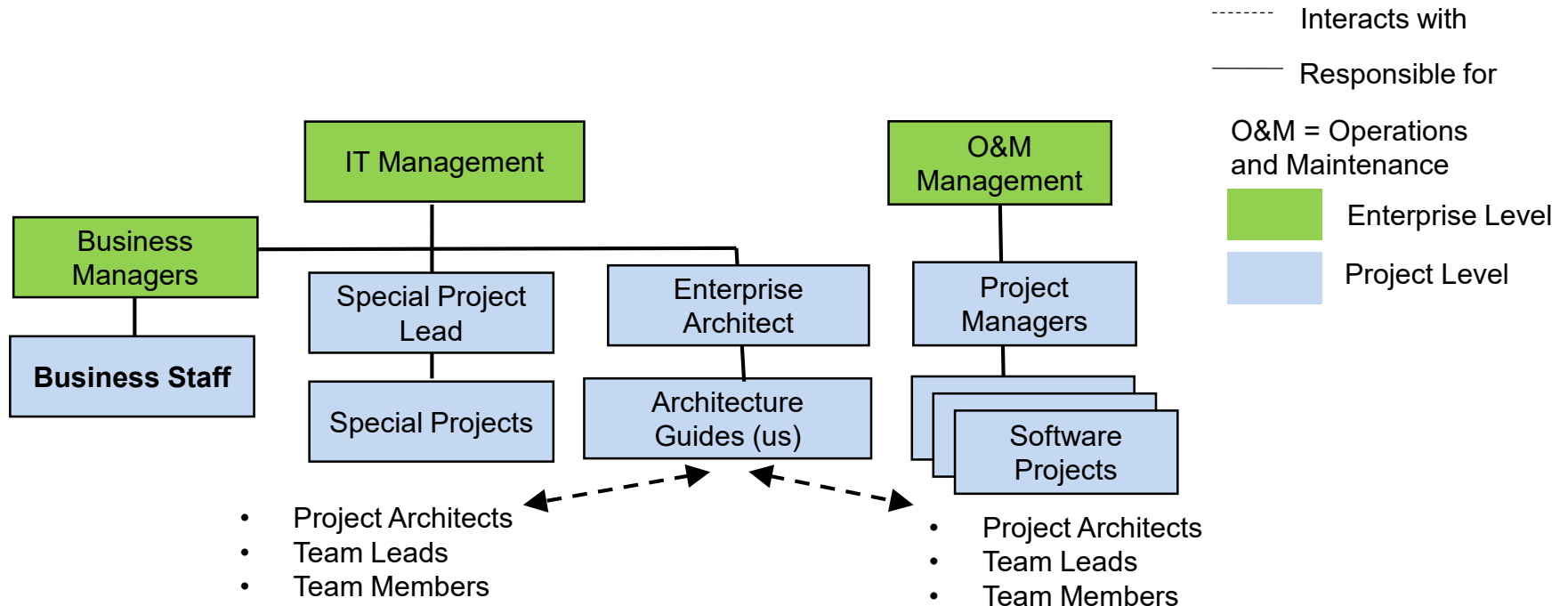
We noticed that some technical debt has unique characteristics

We call this type of technical debt *enterprise technical debt* (ETD)

In this talk, we

- share characteristics and examples of ETD
- explain how to motivate action by making ETD items, along with supporting evidence, visible to stakeholders

# Conceptual Organization Chart



# Our Role

We work for IT management, and we serve as architecture guides for a portfolio of projects; each project has at least one architect

Our achievements in this role include

- establishing an incremental design review process
- establishing a dashboard for tracking design risks
- facilitating and participating in design reviews and code analyses
- educating architects and teams on software architecture practices
- mentoring architects to conduct design review and code analyses

# What Is Technical Debt?

In software-intensive systems, technical debt consists of design or implementation constructs that are expedient in the short term, but set up a technical context that can make a future change more costly or impossible

## Technical debt

- exists in a **system artifact**, such as code, build scripts, automated test suites, or data
- traces to **several locations** in the system, implying ripple effects of impact of change
- has a **quantifiable** effect on system attributes of interest to developers, such as increasing number of defects, negative change in maintainability, and code quality indicators

# Characteristics of Enterprise Technical Debt

Comparison to technical debt:

- ✓ Exists in a **system artifact**
- Δ Is traced to **several systems or projects**
- Δ Has a **quantifiable** effect at the **enterprise management level**

Often, these characteristics also exist:

- ✓ Requires enterprise-level involvement and investment to resolve
- ✓ Requires project resources to fix
- ✓ Conflicts with an enterprise goal or standard
- ✓ Impact may have ripple effects across multiple projects in the enterprise

# Examples of Enterprise Technical Debt

In this section, we will walk through three examples:

1. Shared schema integration (versus service API)
2. Business logic confusion in SpringMVC framework
3. Decentralized access control



# Examples of Enterprise Technical Debt

1. Shared schema integration (versus service API)
2. Business logic confusion in SpringMVC framework
3. Decentralized access control

 **We are here**

# Shared Schema Example: Background

In this example, we have a project that forces two teams from different parts of the organization to exchange data

- One team provides an external portal
- The other provides internal functionality and data used by the portal

The integration approach they used was common at the time it was built

However, now they are feeling the limitations of the approach

Neither team has control or motivation to fix the problem

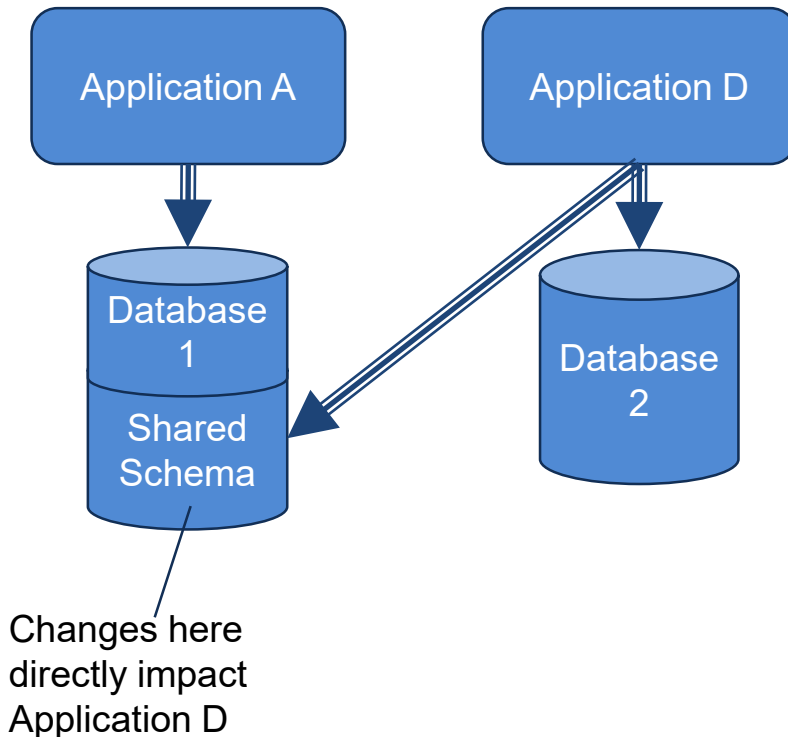
This is because the enterprise has different goals from the project level; therefore, risks may impact the enterprise but not the project (and vice versa)

# Example: Shared Schema Integration

## System Artifact: **Shared Schema**

**Problem: Applications A and D** are integrated using a shared database schema. Changes to the schema by Application A impact D and vice versa.

- ✓ Exists in a **system artifact**
- ✓ Is traced to **several systems or projects**



# Shared Schema Impact and Quantification

Impact	Quantifiable Effect	Who Impacted
Changes made to the shared schema by one team break the other team's application <b>Conflicts with organizational modernization goal to move toward service APIs</b>	Maintenance <b>cost</b> , feature <b>delays</b>	<b>IT Management, Business Managers</b>
Does not scale Reusing the shared schema for more than two projects is very difficult	<b>Delays</b> making UI changes, <b>cost</b> of change	<b>O&amp;M Management</b>
Functionality and data duplication are workarounds to avoid impact of changes	<b>Cost</b> of feature implementation, feature delivery <b>delays</b>	<b>Business Management, O&amp;M Management</b>

- ✓ Has a **quantifiable** effect at **enterprise level**
- ✓ Conflicts with **enterprise goal** or standard

# Solution Recommendations


**Problem:** Product Owner of Application A states, “There is no money to make the life of team Application D easier.”

Therefore, the recommended solution is a two-pronged approach:

- **Enterprise Management** must allocate funds and/or reorder priorities to fix this
- **Projects J and K** must do the work to replace shared schema integration with service API

- ✓ Requires **enterprise-level involvement** and investment
- ✓ Requires **project resources** to fix

# Examples of Enterprise Technical Debt

1. Shared schema integration (versus service API)
2. Business logic confusion in SpringMVC framework  **We are here**
3. Decentralized access control

# SpringMVC Example: Background

For many years, the organization has been suffering from slow feature delivery

A contributing factor is that business rules are PL-SQL stored procedures that have become complex and hard to change

The organization

- set a goal to extract business logic out of the database stored procedures
- decided to use SpringMVC pattern to separate business logic from physical storage layer
- started with two projects using the SpringMVC pattern

# Example: MVC Business Logic Inconsistency

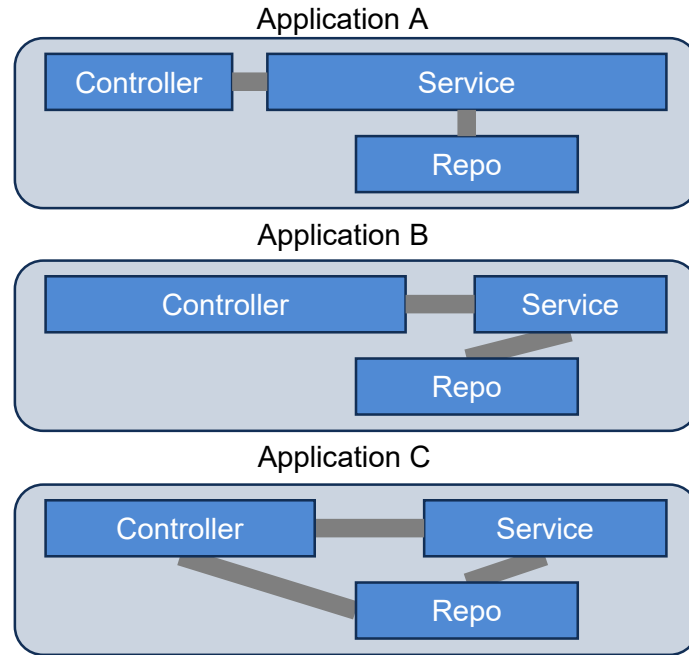
## System Artifact:

SpringMVC framework Service  
with focus on Controller

## Problem:

**Application B and C** put business logic in SpringMVC Controller (versus Service), which creates tight coupling between UI layer and data layer

- ✓ Exists in a **system artifact**
- ✓ Is traced to **several systems or projects**





# MVC Impact and Quantification

Impact	Quantifiable Effect	Who Impacted
The organization has an <b>enterprise goal to clean up the physical data storage tables</b> , but because there is coupling from UI to DB layer, there may be impact if data storage level changes	Maintenance <b>cost</b> , feature <b>delays</b>	<b>Business Management, O&amp;M Management</b>
Changes to Controller could impact Data Layer, which would impact multiple applications	<b>Delays</b> making UI changes, <b>cost</b> of change	<b>O&amp;M Management</b>
Because business logic is not in consistent places, maintainers take longer to create new features	<b>Cost</b> of feature implementation, feature delivery <b>delays</b>	<b>Business Management, O&amp;M Management</b>

- ✓ Has a **quantifiable** effect at **enterprise level**
- ✓ Conflicts with **enterprise goal** or standard

# SpringMVC Recommended Solution


**Executive management** must

- direct the projects to make the changes
- set aside funds for projects to do rework if needed
- reprioritize other tasks to get it done

**Projects B and C** must move business logic out of Controller to Service Component

- ✓ Requires **enterprise-level involvement** and investment
- ✓ Requires **project resources** to fix

# Examples of Enterprise Technical Debt

1. Shared schema integration (versus service API)
2. Business logic confusion in SpringMVC framework
3. Decentralized access control  **We are here**

# Decentralized Access Control Context

For many years, project teams have been building applications independently in silos

There is no centralized access control repository or maintenance screen

Each application writes access control roles/permission logic itself

Consequently, the teams write access control information for **each application** in different places in the database schemas

The project teams are fine with this, but the enterprise is suffering due to security and maintainability issues

# Example: Decentralized Access Control

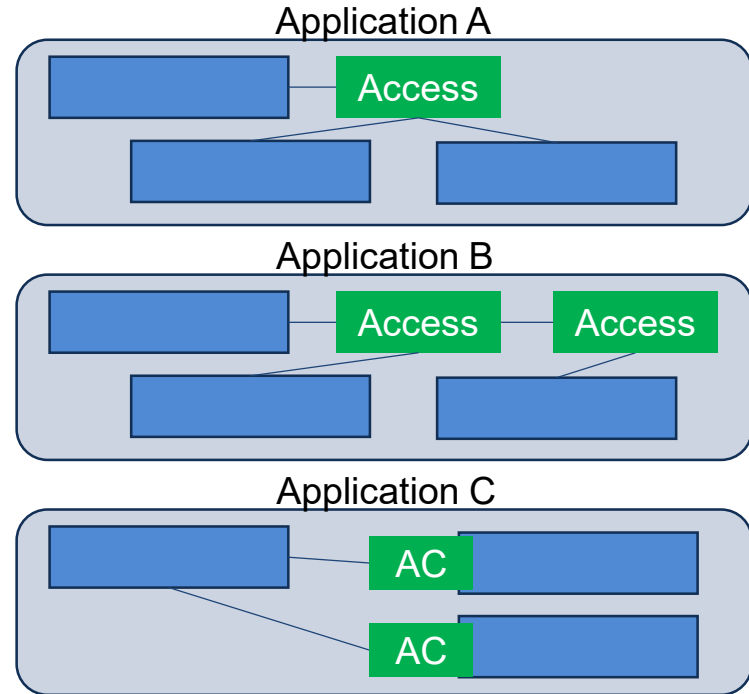
## System Artifact:

Access control architecture (or lack thereof)

## Problem:

Each application team writes access control roles/permission logic itself; Access control information is spread all over the databases

- ✓ Exists in a **system artifact**
- ✓ Is traced to **several systems or projects**



# Decentralized Access Control Impact

Impact	Quantifiable Effect	Who Impacted
<p><b>Security</b> – Makes it more difficult and time consuming to delete a person from multiple applications and you run the risk of forgetting to delete the person from one or more applications.</p>	<p>Data loss <b>cost</b>, risk of exploitation <b>cost</b></p>	<p><b>O&amp;M Management, Business Managers</b></p>
<p>Every application does access control independently, leading to <b>too many different implementations, so features take longer to develop</b>. Approach <b>conflicts with enterprise goal to create common authentication component</b></p>	<p><b>Cost</b> of feature implementation</p>	<p><b>O&amp;M Management, Business Managers</b></p>
<p>Roles are hard coded in some applications, which limits flexibility; if something changes in the organization, the code must change</p>	<p><b>Cost</b> of role changes, admin change <b>delays</b></p>	<p><b>O&amp;M Management, Business Managers</b></p>

- ✓ Conflicts with an **enterprise goal** or standard
- ✓ Has a **quantifiable** effect at **enterprise level**

# Access Control Recommended Solution

**Executive management** must

- direct the projects to make the changes
- set aside funds for projects to do rework if needed
- reprioritize other tasks to get it done

A Common Authentication capability should be created to store role and permission information with an administrative capability to allow project teams to make changes.

**Then project teams will need to adapt applications to use it.**

- ✓ Requires **enterprise-level involvement** and investment
- ✓ Requires **project resources** to fix

# Managing Technical Debt

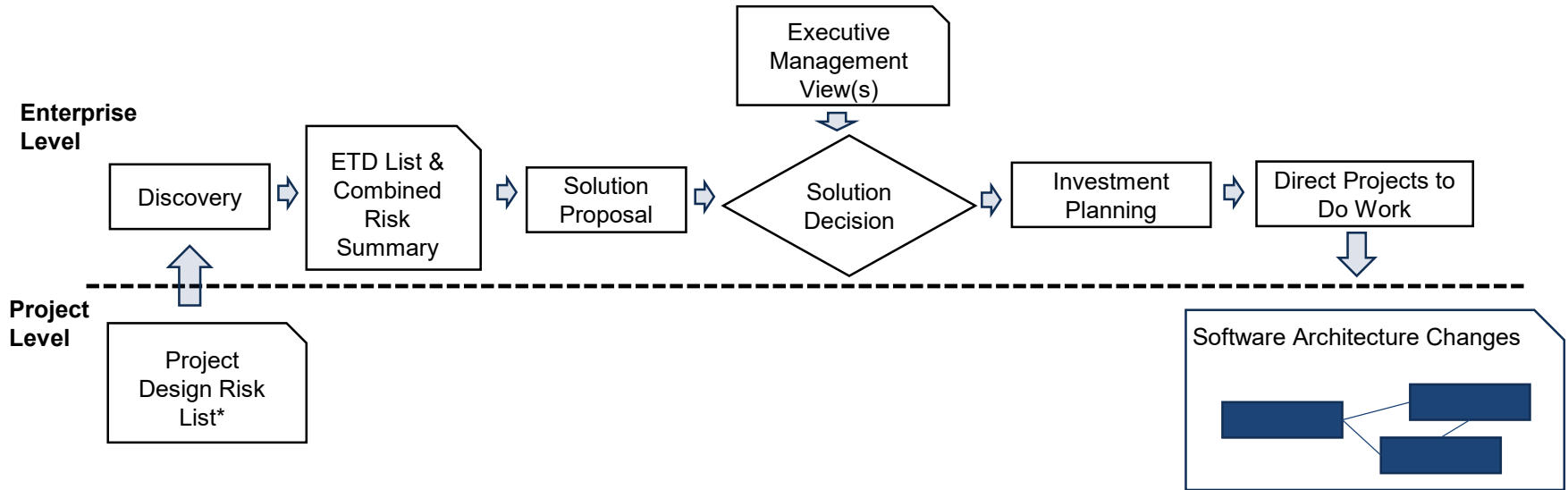
Discovering enterprise technical debt is just the start; after discovery we need to do something with it

On the next few slides, we will cover these topics:

- Process overview for managing ETD
- Why and how we make ETD visible
- Implications of managing ETD as a continuous process



# Process Overview for Managing ETD



To keep the diagram simple, we combine the Project Design Risk List and Project Technical Debt Item List into one list

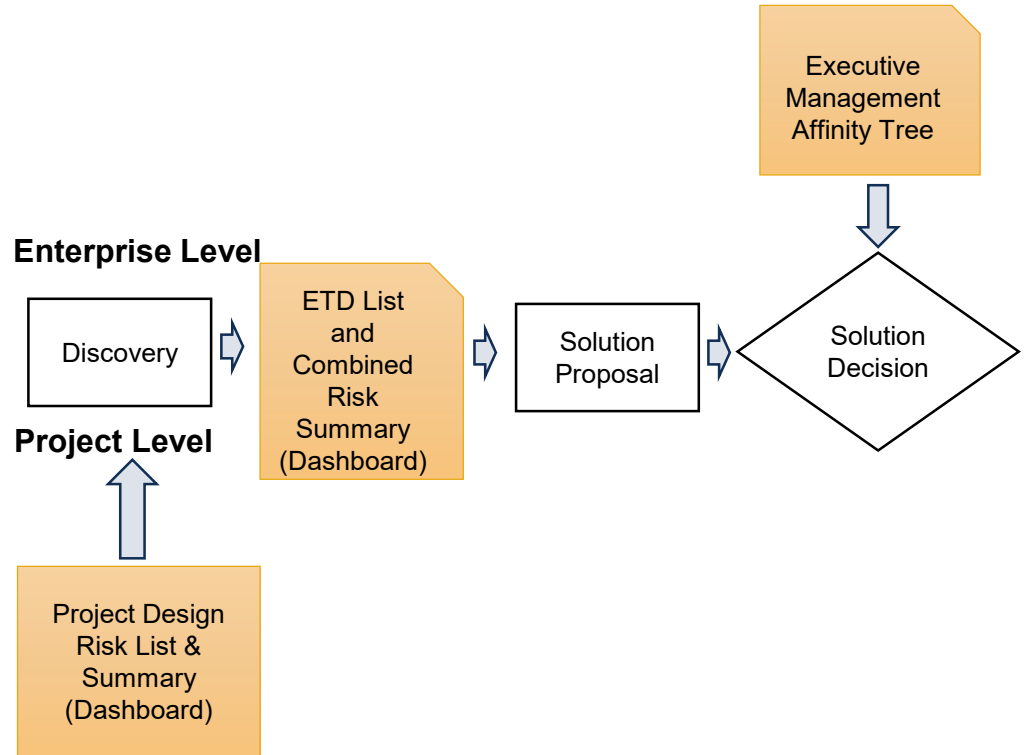
If a Project Design Risk has accumulation, we flag as Technical Debt Item in the list

# Making Data Visible

If not managed correctly, ETD will be forgotten until it is too late

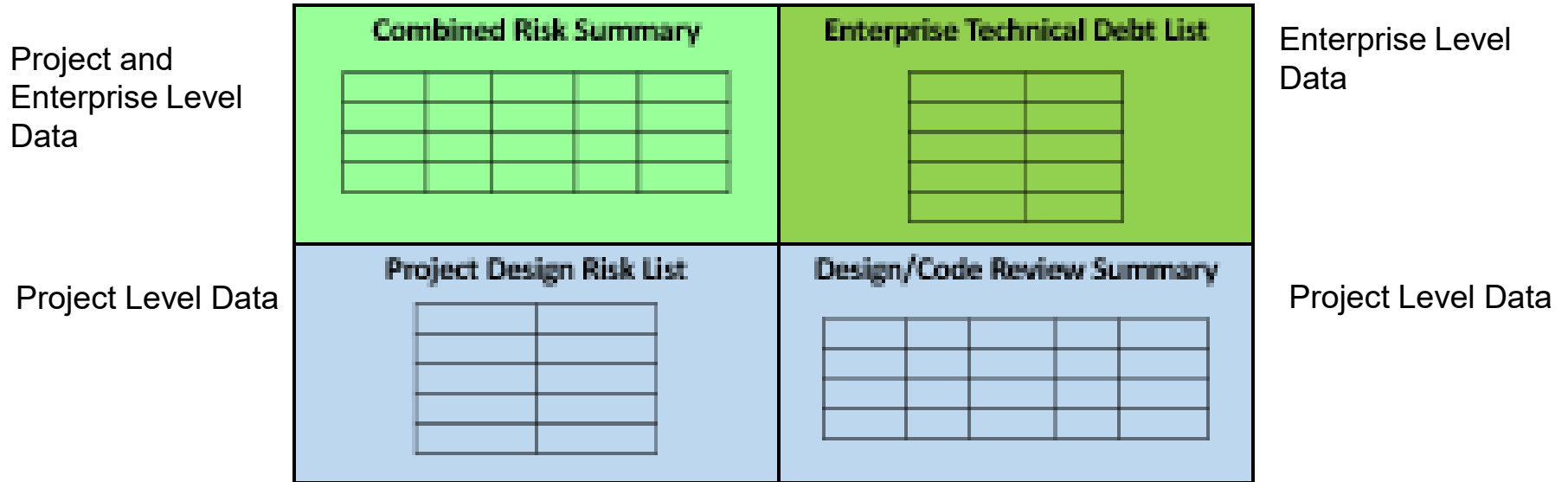
The key to getting action is to make data visible

We make different kinds of data visible for different stakeholders and purposes



# Software Architecture Dashboard

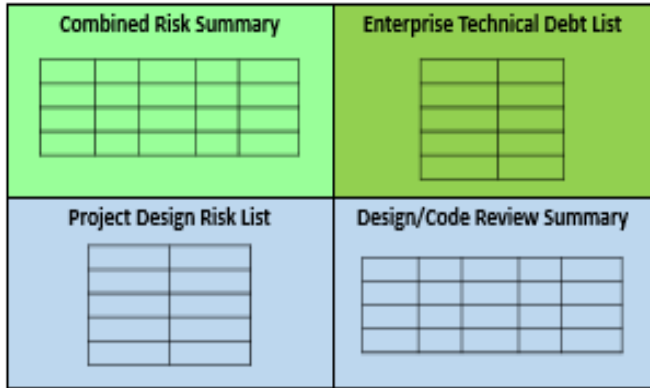
Our Software Architecture Dashboard has four parts; the bottom two quadrants are project level and the top two quadrants are enterprise level



# Dashboard – Project Design Risk List

The Project Design Risk List shows the project design risks

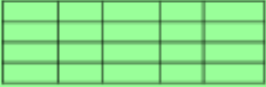
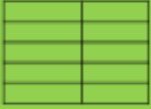
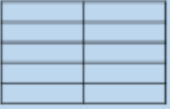
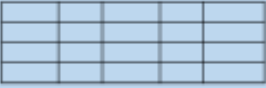
Each project design risk associated with an ETD increases the probability the risk will have an impact



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 is a project-level roll-up of open issues by design or code review activity

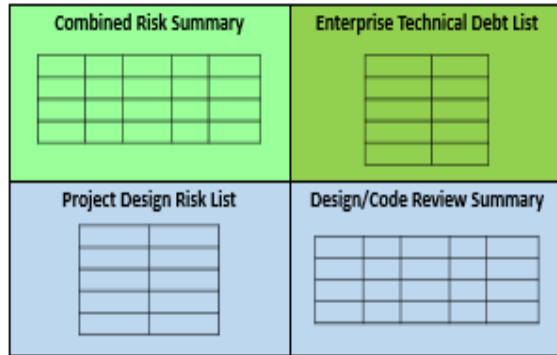
<b>Combined Risk Summary</b> 	<b>Enterprise Technical Debt List</b> 
<b>Project Design Risk List</b> 	<b>Design/Code Review Summary</b> 

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



# Enterprise Technical Debt List

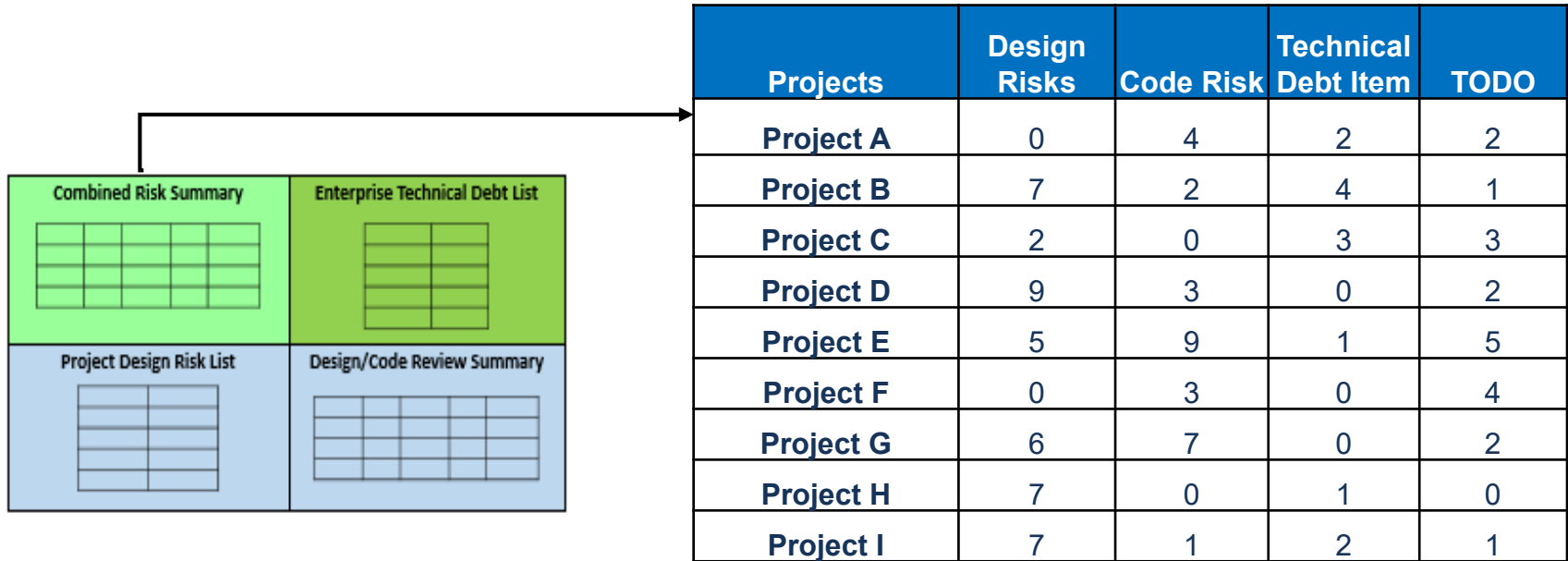
This is an example ETD List derived from our current work (cleansed)



ID	Enterprise Technical Debt Description
<b>SWQ-131</b>	ABC Portal using XYZ doc storage instead of enterprise doc storage
<b>SWQ-128</b>	The path for crossing extranet to intranet makes OSB redundant
<b>SWQ-127</b>	Externally facing data exchange Apps A and B use data syncing
<b>SWQ-126</b>	Standardized enterprise access control missing so App D team rewrites
<b>SWQ-89</b>	Microservices pattern misused, introducing overhead and latency
<b>SWQ-88</b>	Multiple sets of services for Case Management in heterogeneous ways
<b>SWQ-85</b>	There are two versions of C Application; features are duplicated in both
<b>SWQ-74</b>	Create Address Information Shared Service
<b>SWQ-68</b>	Project E moving business logic to SpringMVC PLSQL is copied "as is"
<b>SWQ-66</b>	Project A and C SpringMVC UI to data layer dependencies
<b>SWQ-65</b>	Enterprise authentication and access control partially finished
<b>SWQ-64</b>	A Shared Lookup Data Service partially finished
<b>SWQ-58</b>	Application A and D using a brittle shared database schema

# Dashboard: Combined Risk Summary

The Combined Risk Summary is at the enterprise level and is a roll-up of open issues by design or code review activity



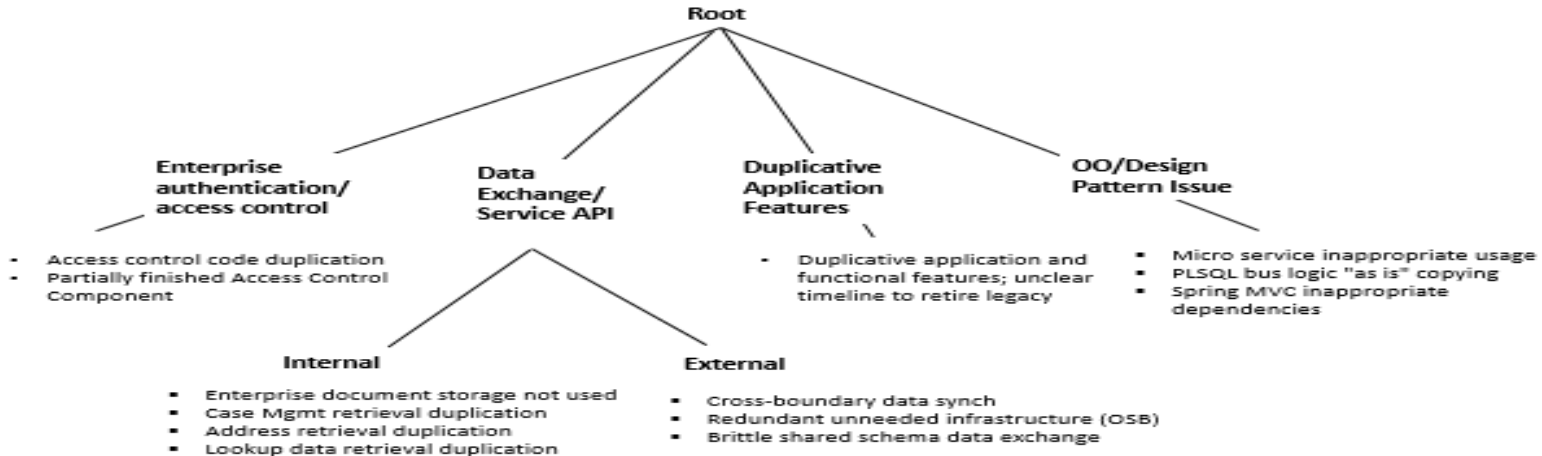
Projects	Design Risks	Code Risk	Technical Debt Item	TODO
Project A	0	4	2	2
Project B	7	2	4	1
Project C	2	0	3	3
Project D	9	3	0	2
Project E	5	9	1	5
Project F	0	3	0	4
Project G	6	7	0	2
Project H	7	0	1	0
Project I	7	1	2	1

# Executive Management Affinity Tree

This view is useful for helping Executive Management prioritize investments

Each bullet is an item on the ETD List

For example, this chart makes clear there are a lot of issues under Data Exchanges and Service API, so this is an area for the organization to focus on

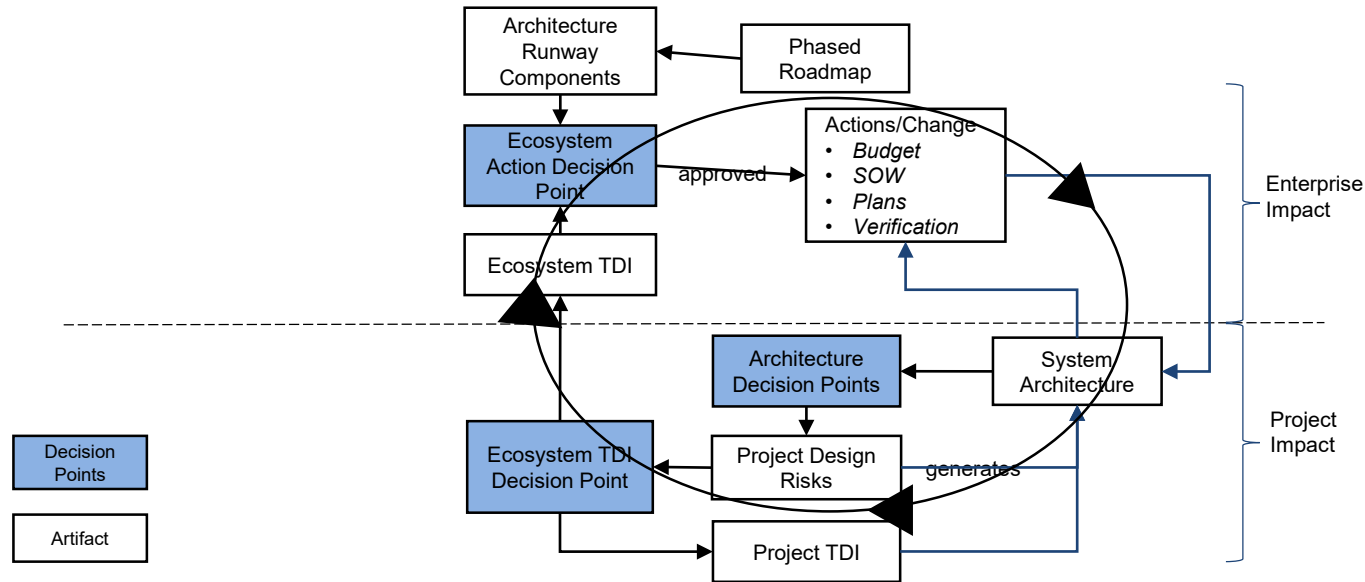




# Continuous Process

Making enterprise technical debt visible is not a one-time exercise

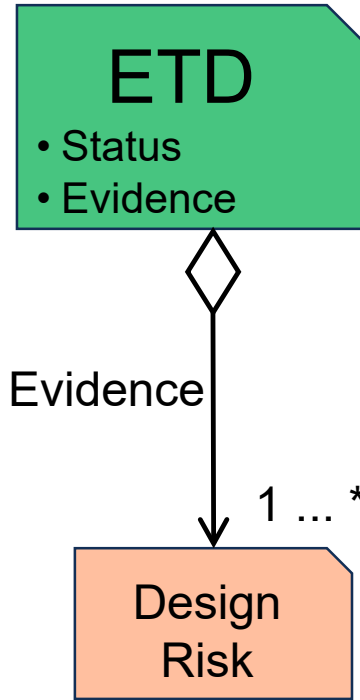
It is part of a continuous process, and the architect is the driving force!



# The More Evidence, the Stronger the Case

One of the architect guide's jobs is to continuously look for more evidence in risk data

The more evidence, the stronger the case for taking action



Status:

- **Proposed:** Not sure if it is EDT (assumed evidence)
- **Defined:** An open EDT item (clear evidence)
- **Resolved:** Decision to pay back (it is hurting!)
- **In progress:** Being resolved in one or more projects
- **Done**

# Wrap-up

## Summary

We discussed characteristics of ETD and provided several examples

We talked about the importance of data visibility and gave examples

We emphasized the architect guide's role in continuously monitoring for ETD

## Next Steps

We shared the ETD views with management for the first time in January

Encouraging outcome; ETD data gave us credibility when arguing for investments

Need to keep collecting data; more data will strengthen the case!

In the future, we plan to focus on tactics for proactive management of ETD

# Contact Information

Felix Bachmann

[fb@sei.cmu.edu](mailto:fb@sei.cmu.edu)

Stephany Bellomo

[sbellomo@sei.cmu.edu](mailto:sbellomo@sei.cmu.edu)