From Monolith to Micro-Services

An Architectural Strategy

Software Intelligence for Digital Leaders

O'REILLY®

Software Architecture

softwarearchitecturecon.com | #OReillySACon

www.castsoftware.com

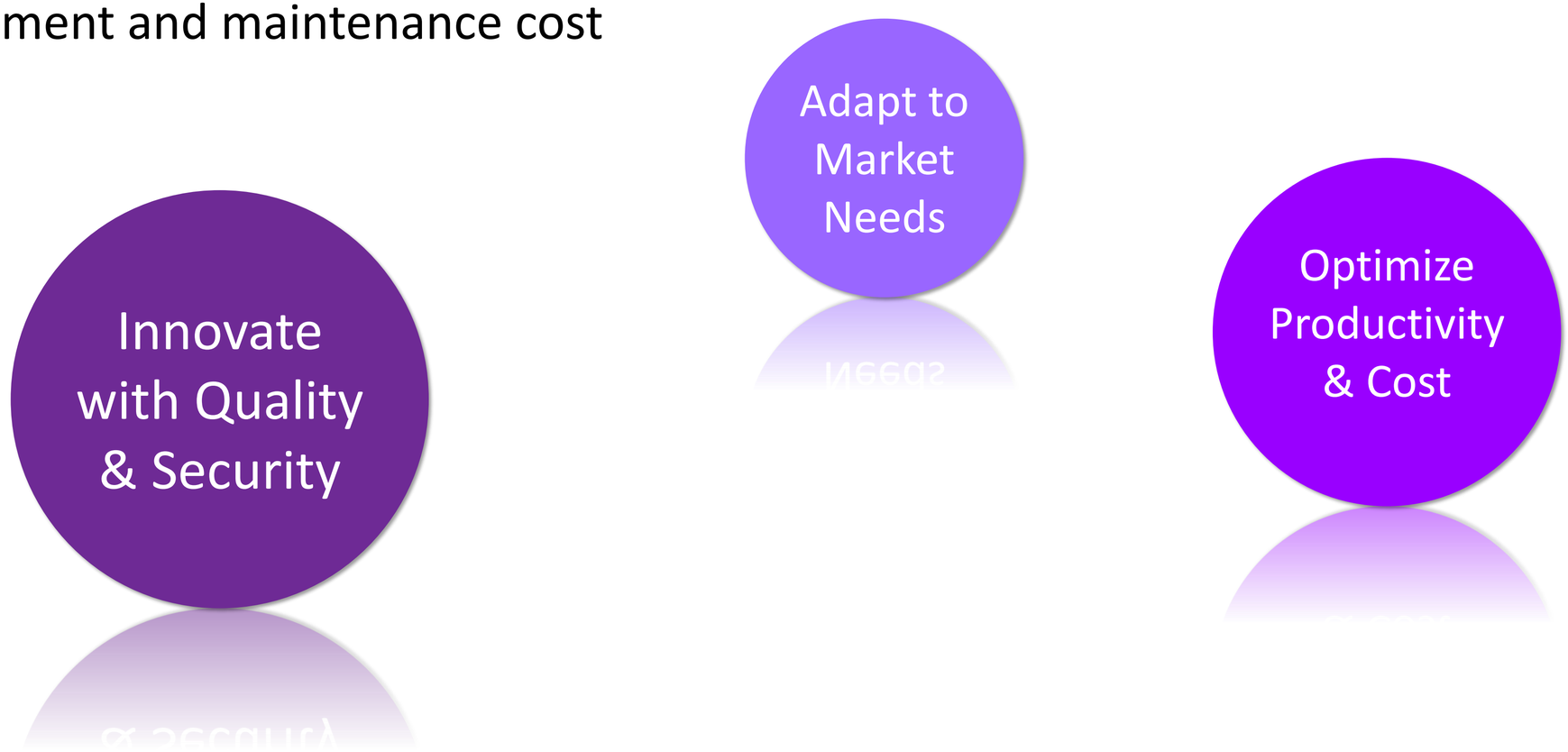
Goal of this Workshop

- Present approaches and techniques used on various architectures to support the transformation of a monolithic application into a microservices application.

- The importance of tracking the changes of the transformation and balancing the loss of performance or stability with the introduction of new layers.

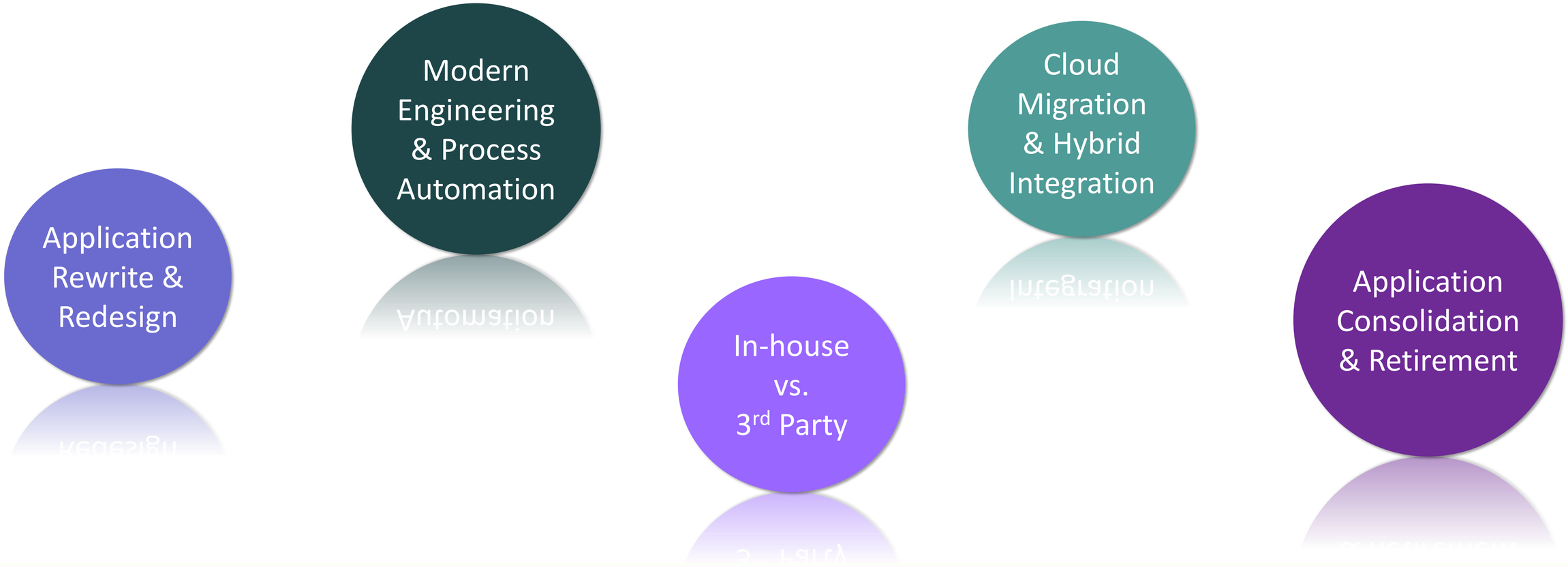
Digital technologies are changing the face of business

Businesses expect IT to ADAPT to evolving market needs, Leverage robust and secure technology solutions and build with optimal development and maintenance cost structure.



Immediate Decisions to Stay Relevant

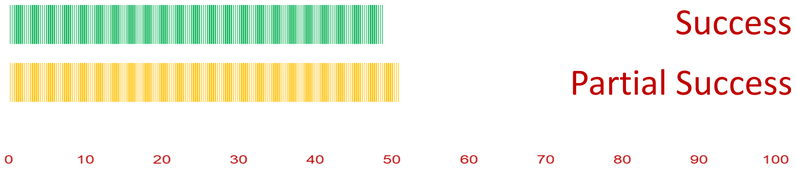
Successful leaders have strategized, designed and executed modernization plans that would prepare their organizations to compete and excel



Common Obstacles in Modernization

Many organizations commit heavy investment in process and technology transformation

Modernization Success Rate

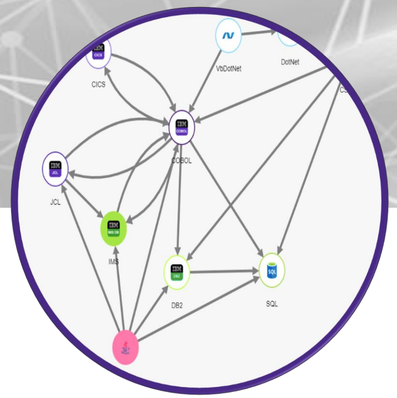


Yet, more than 50% found their initiatives did not meet expectations and experienced cost overrun

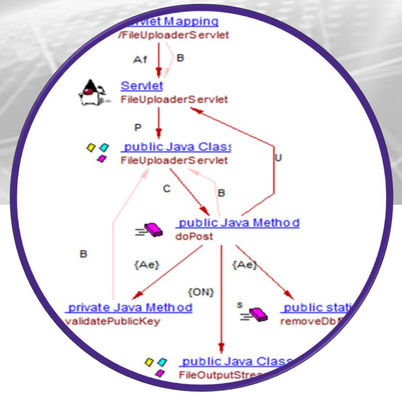
Software Intelligence for Modernization Journey

Portfolio Assessment

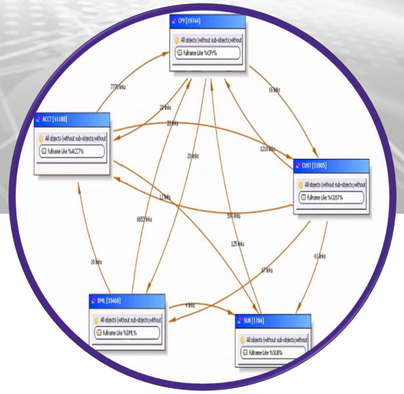
AS-IS Architecture (Discovery)



TO-BE Architecture (Design)



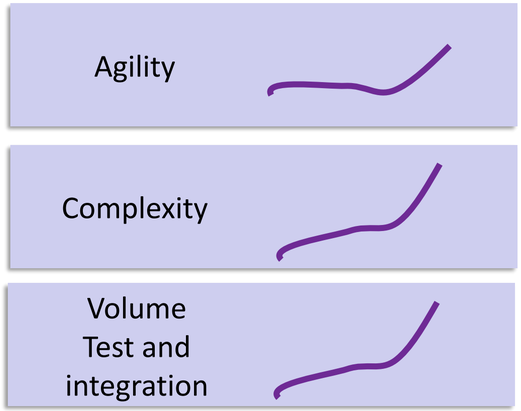
Implementation (Engineering)



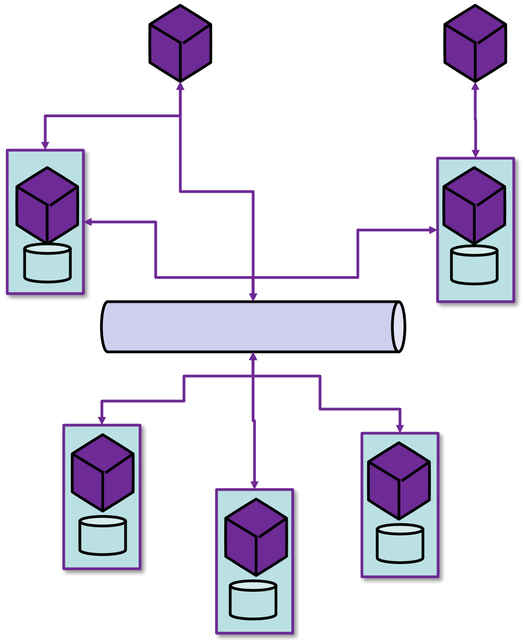
Portfolio Management (Trends/Progress)

Being more Agile but…

- Continuous **deployment is difficult**
- Application can be **difficult** to **understand** and **modify**
- Requires a **long-term commitment** to a **technology** stack
- **Obstacle to scaling** development



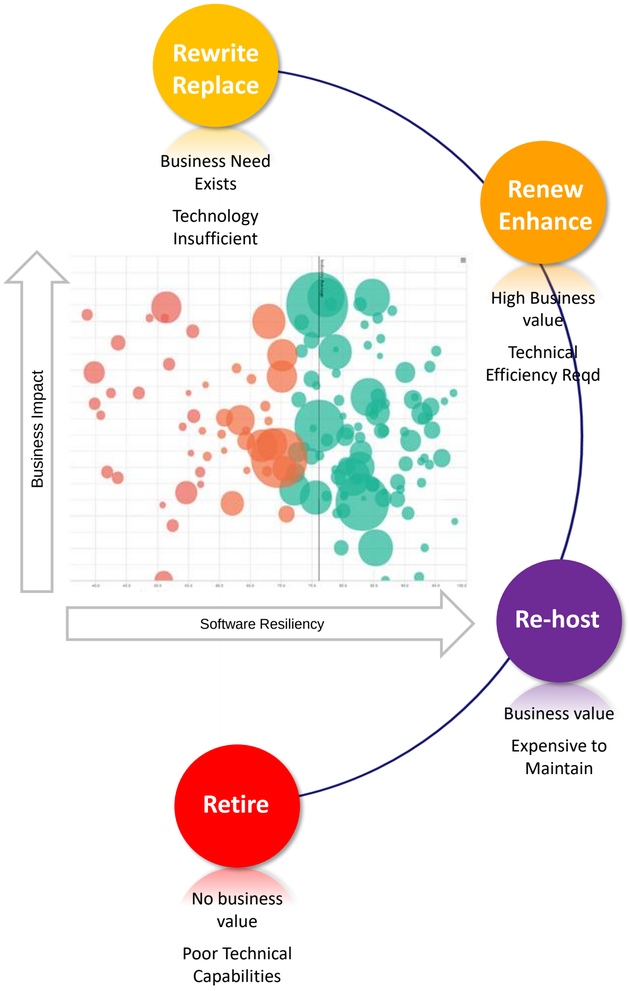
- ✓**Independent deployments** and development
- ✓Team **agility**
- ✓**Mixed technology** stacks
- ✓Granular **scaling**



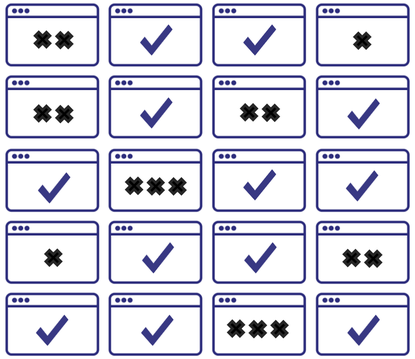
Where to Start

- Which app to modernize first?
- Business critical, data sensitive or troubled one?

- Is that app the most vulnerable?
- Which app do I work on next?



- Efficiently scan the application portfolio
- Prioritize by difficulty and effort

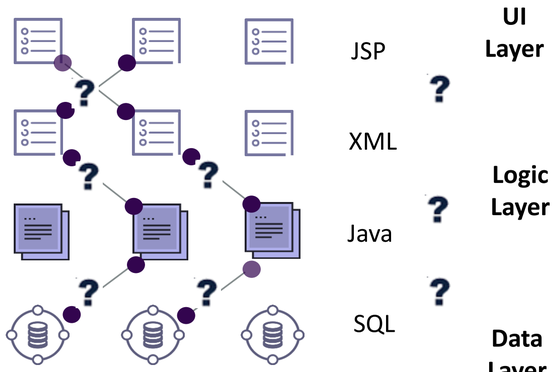


- Evaluate cost/benefit trade-off
- Create modernization plan - plain and simple

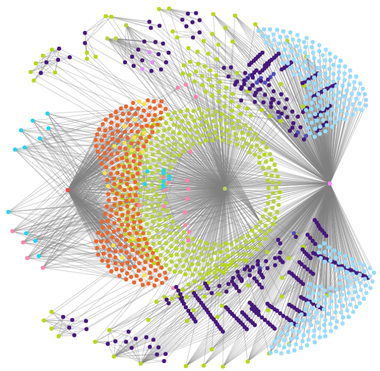
Take an MRI of the As-Is Architecture

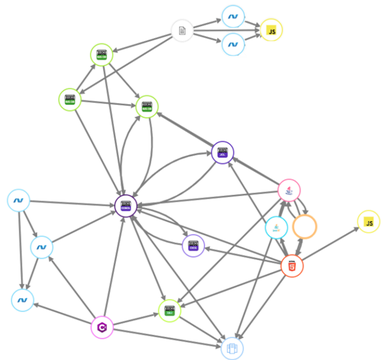
- Manual investigation of IT assets and code base is resource intensive and ineffective
- Many lack capability to understand how data and system function as a whole

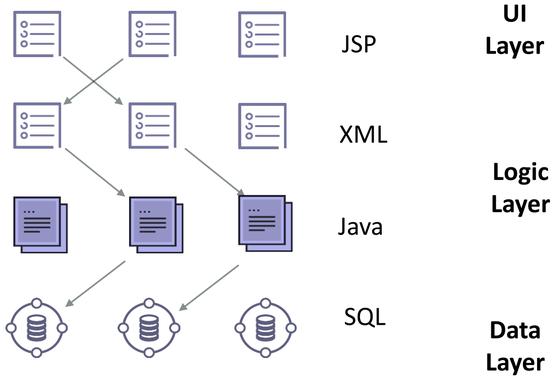
- Make system architectural blueprint visible
- Map data flow across system components
- Identify application vulnerabilities and other unknowns



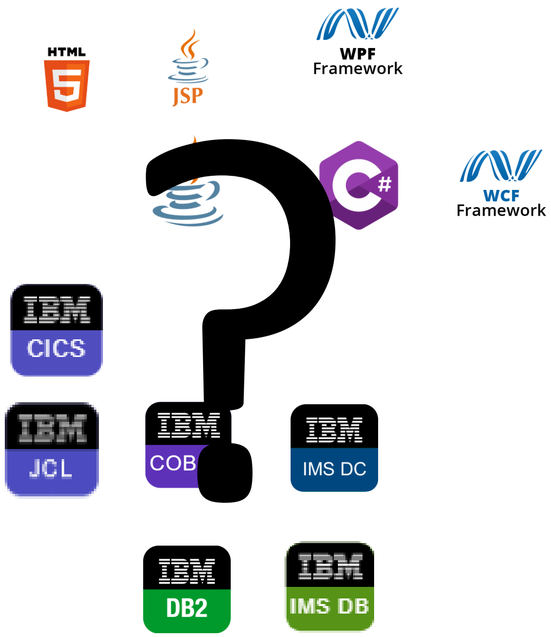
As-Is Architecture:
Understand what should change

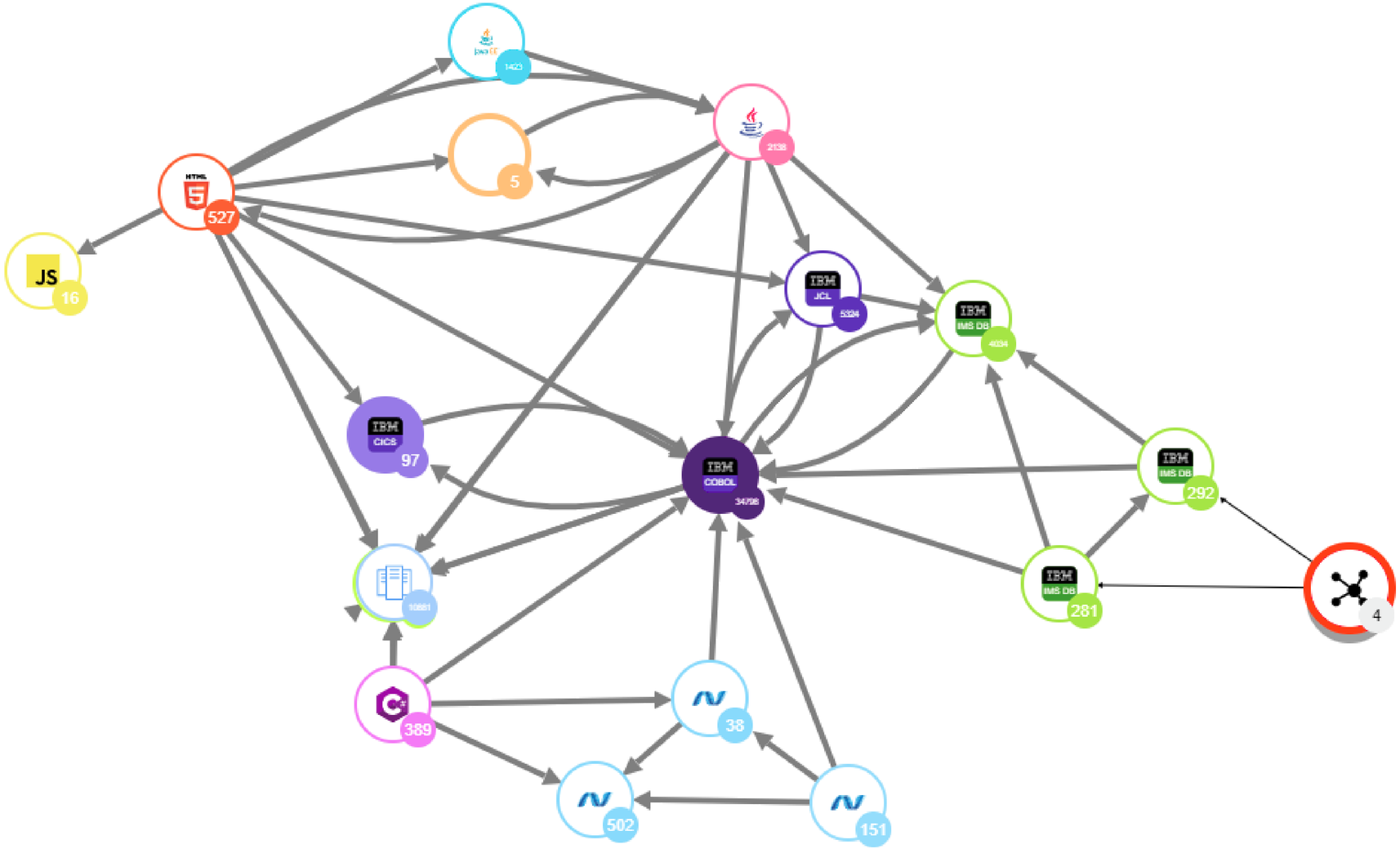






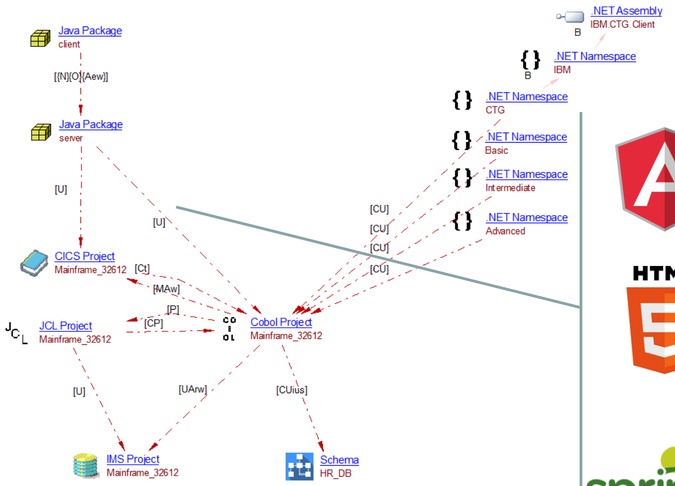
Review the As-Is Architecture

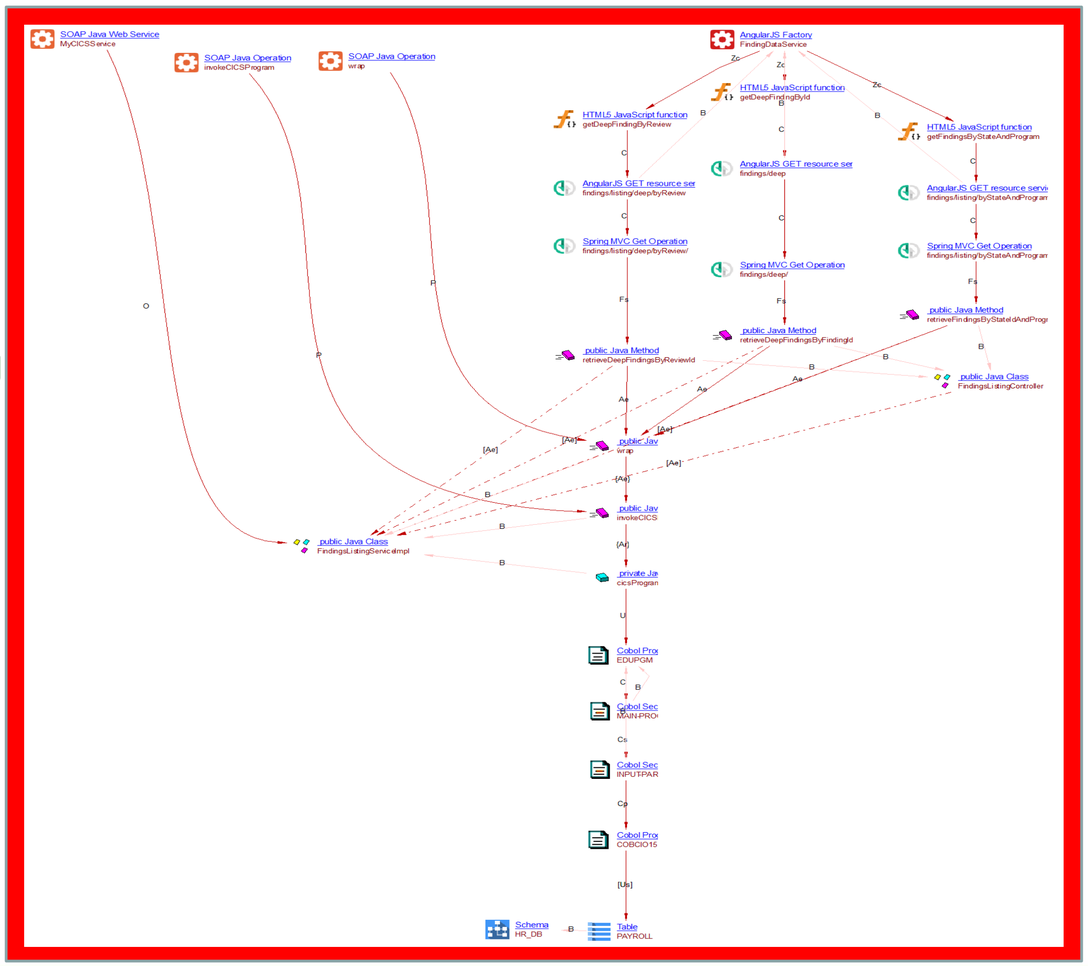




Review the As-Is Architecture

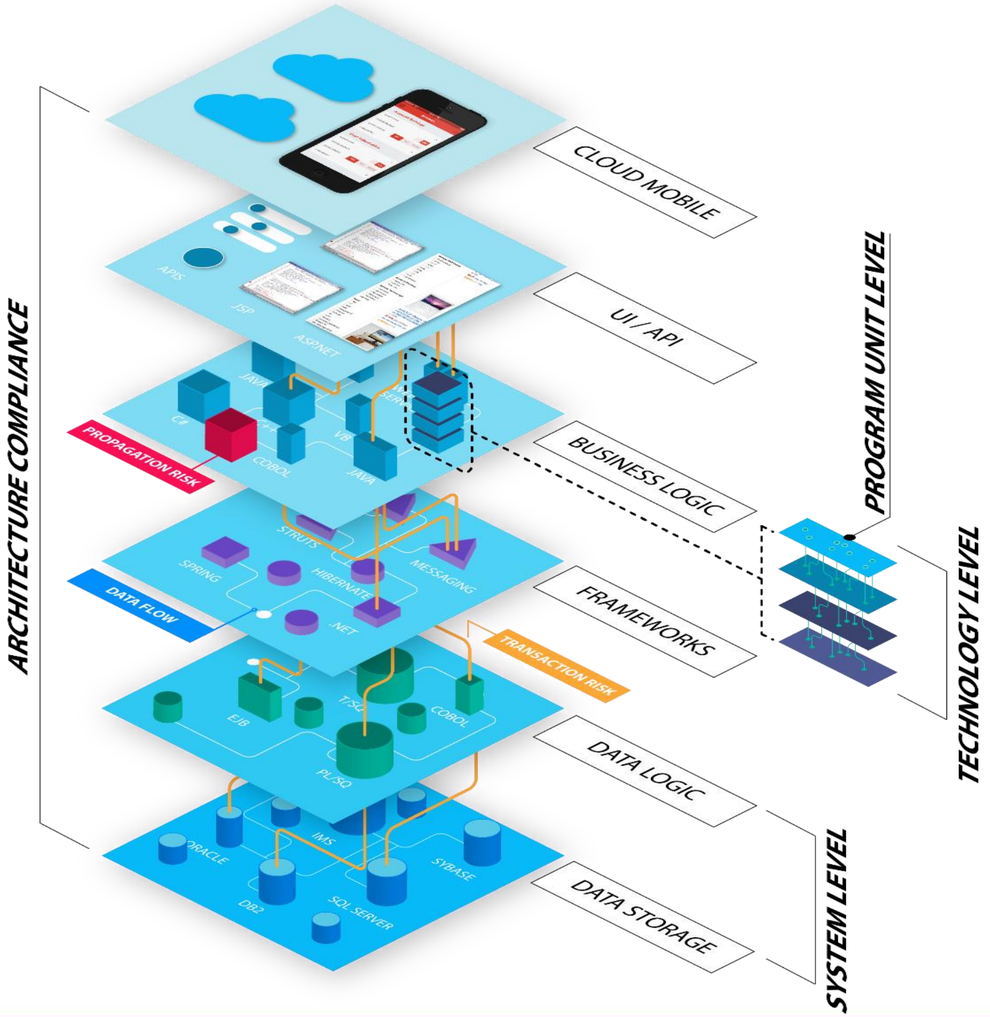
1. Map out components and dependencies based on calls in the code
2. See the complexity and dependencies BEFORE you begin modification
3. Plan, budget and allocate resources correctly and appropriately
4. Mitigate or avoid unexpected overruns due to unknown dependencies



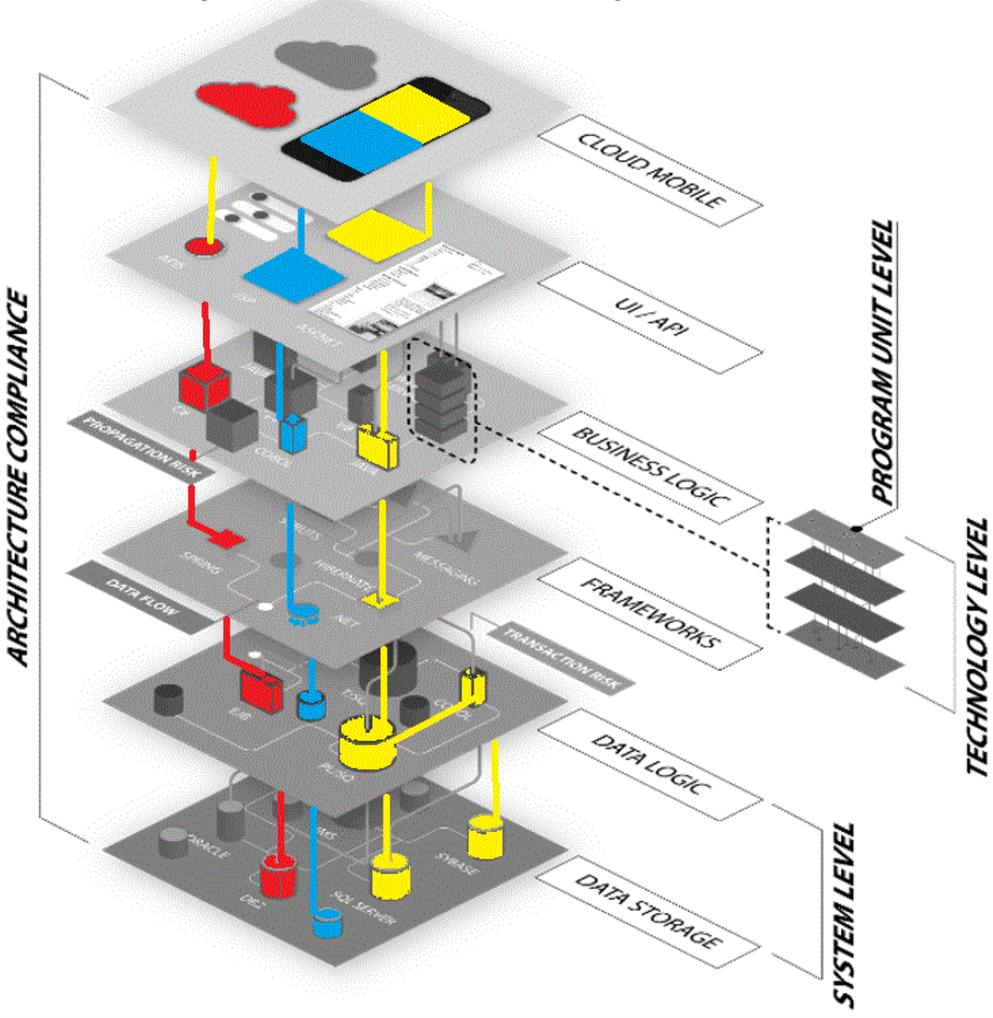


Two Sides to Modernize a System

By layers (East - West)



By transaction (North - South)



How to Solve

A Split the application architecture model by **technology, frameworks, functionality**

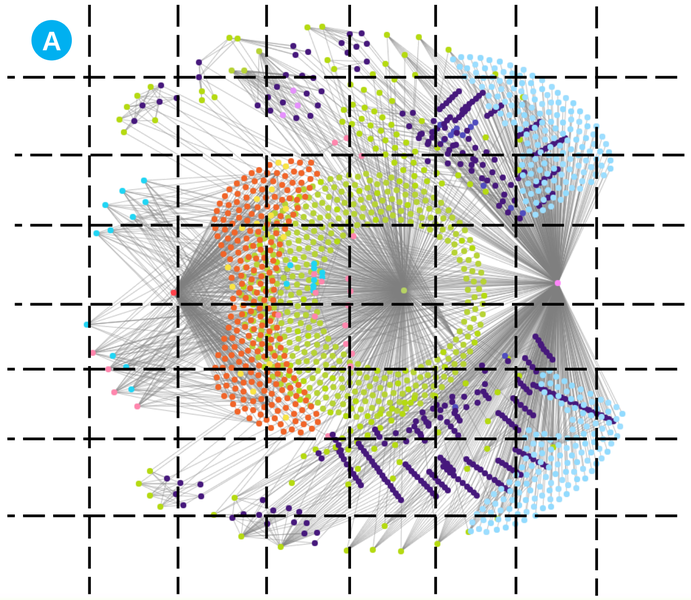
B Identification of **actual dependency** between technologies

C Review the exposed **interfaces**

D **Fix** violations on **non compliance** of use of interfaces

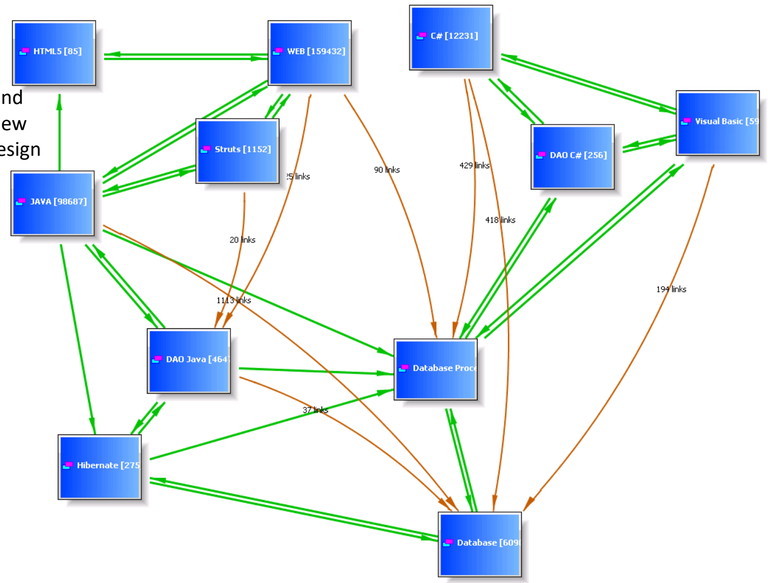
E Define a **more granular exposed interfaces**, **externalize** a specific piece of the application **or** move to the **next phase**

A



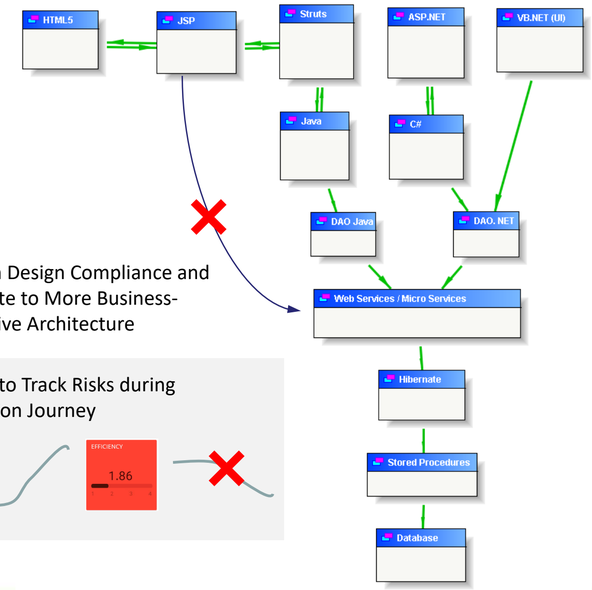
B

Clearly define and communicate new Architecture/Design



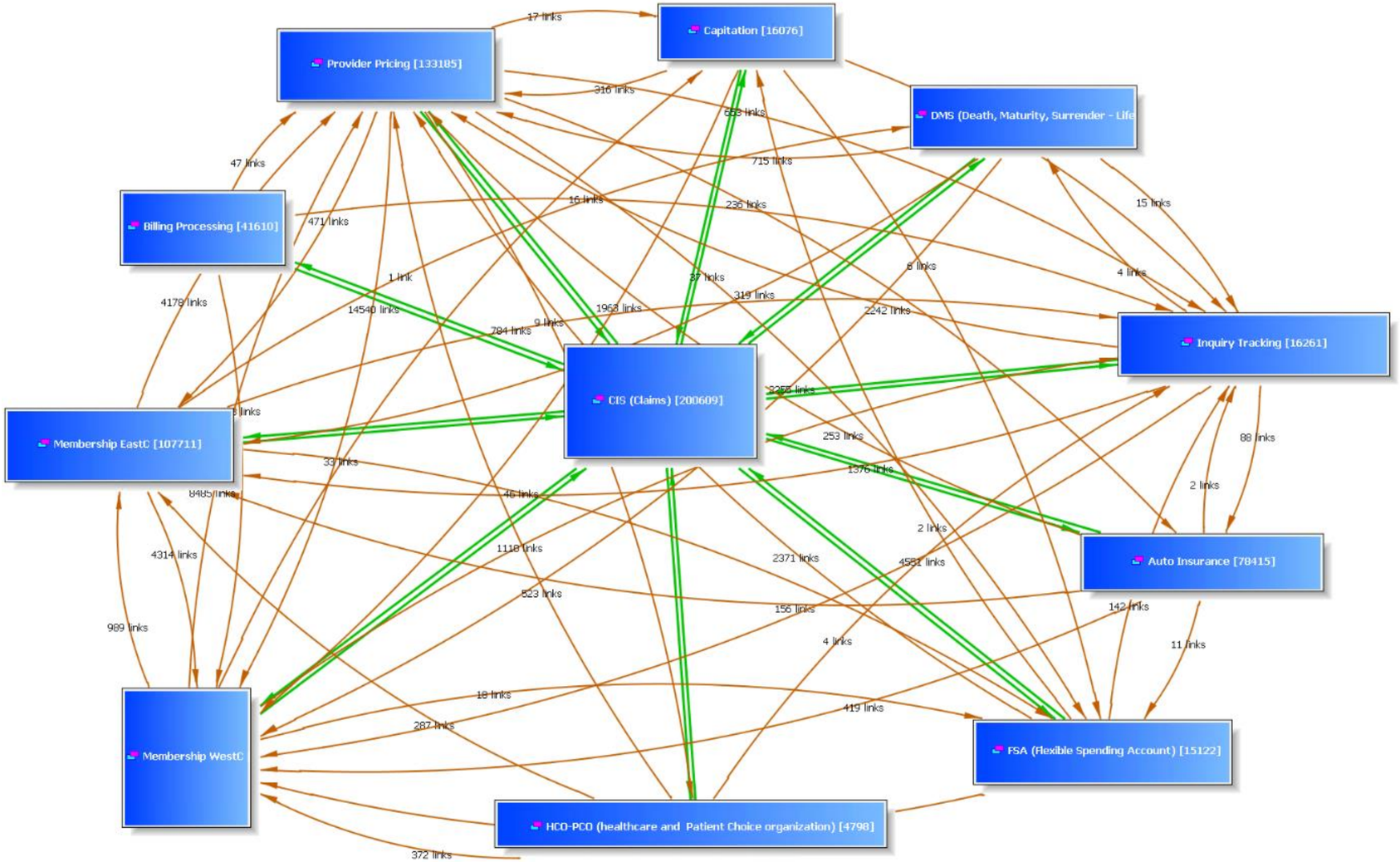
C

Maintain Design Compliance and Accelerate to More Business-Responsive Architecture

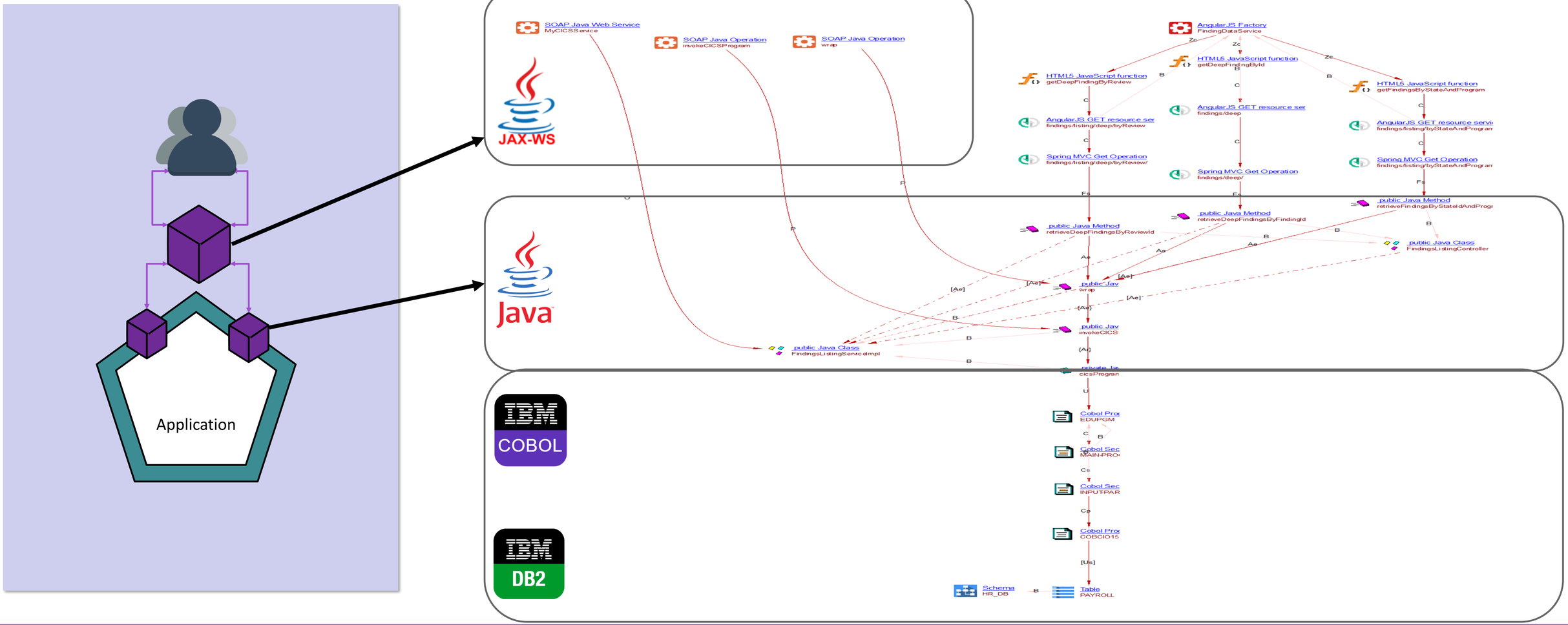


Use Metrics to Track Risks during Transformation Journey

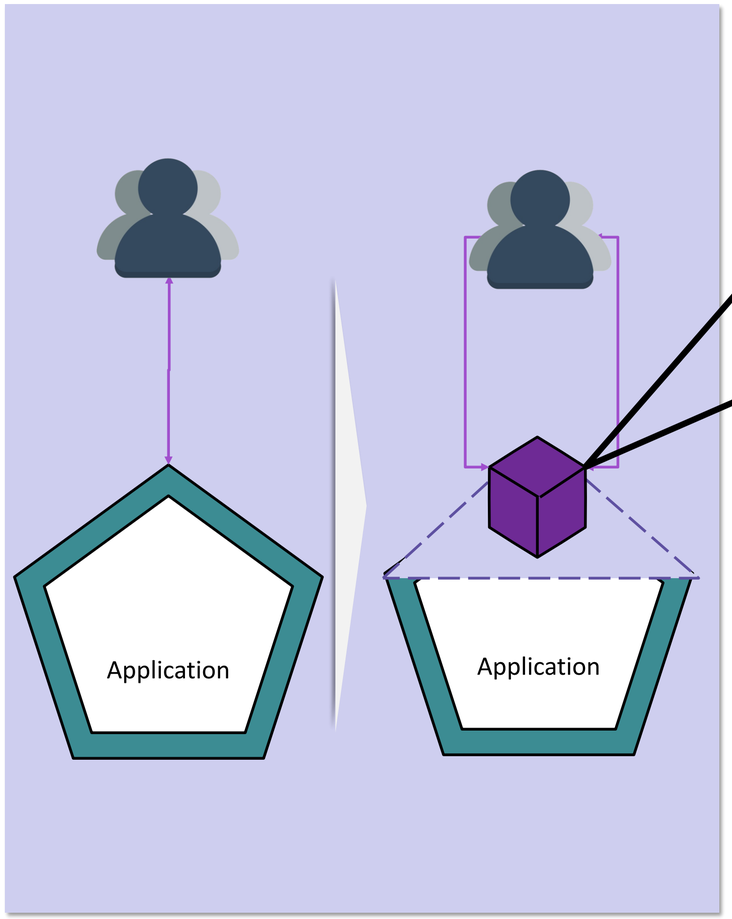
Don’t Forget it is a Journey (step-by-step)

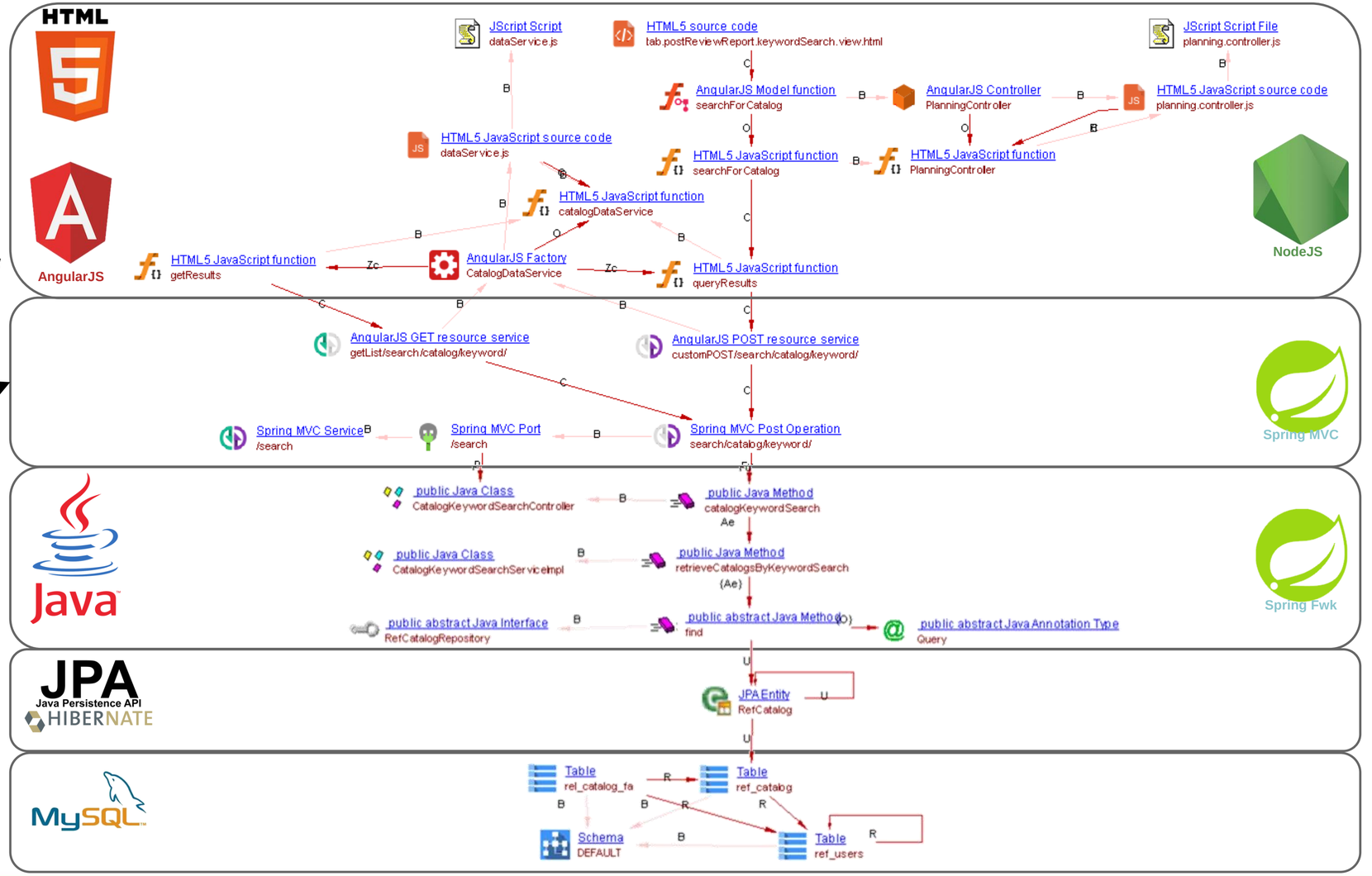


Revamp your Application

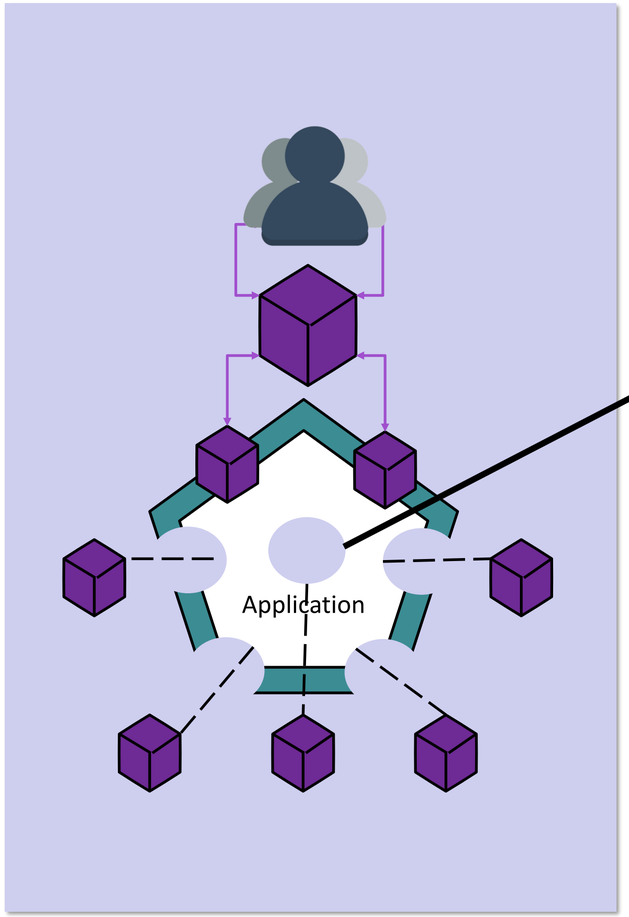


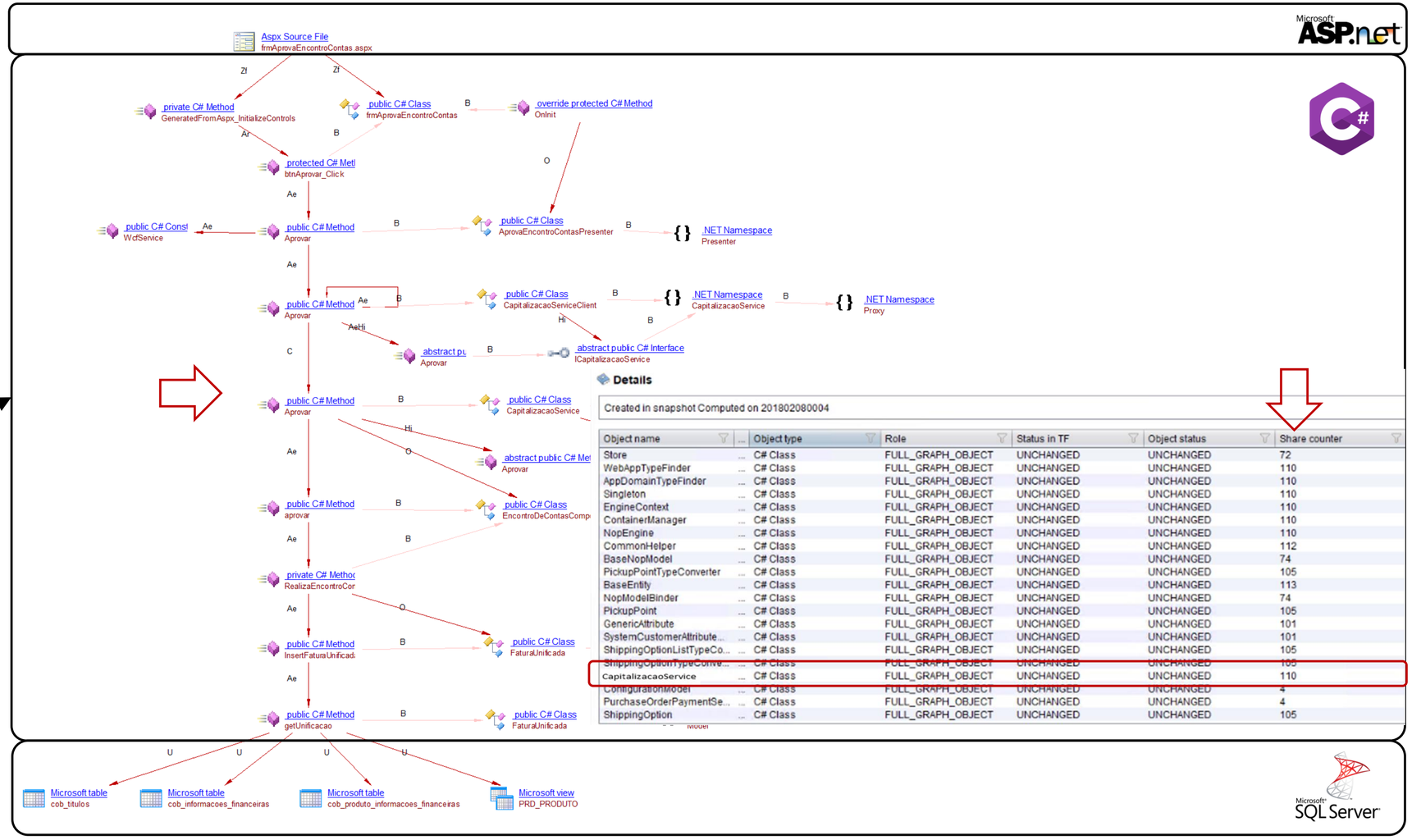
Split Frontend and Backend



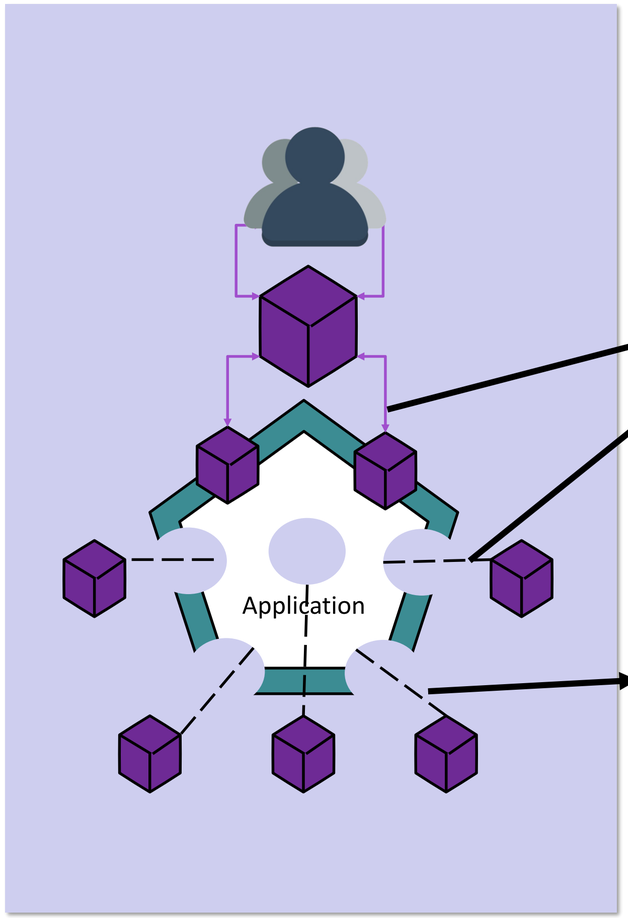


Extract Services





Beyond APIs



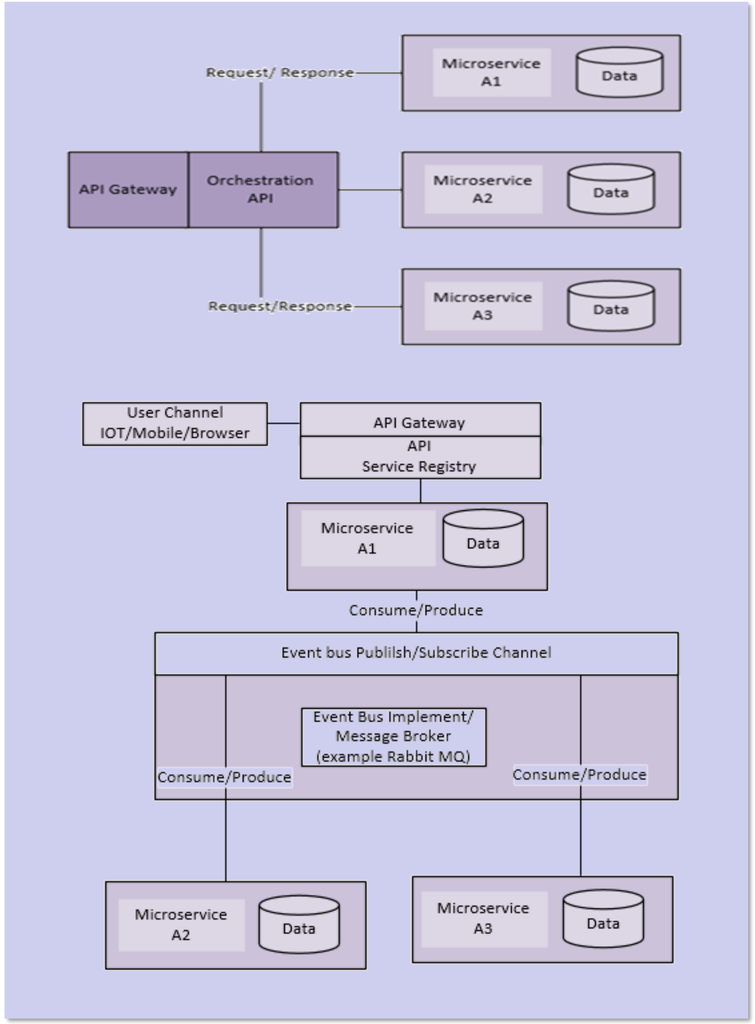
REST

SOAP

- **HTTP/HTTPS** Support
- **Limited bandwidth and resources**; REST and JSON are easy to develop. REST approach uses the standard GET, PUT, POST, and DELETE verbs.
- **Totally stateless operations**; if you need stateless CRUD operations, then REST is the solution.
- **Caching situations**; if the information can be cached because of the totally stateless operation of the REST approach, this is perfect.

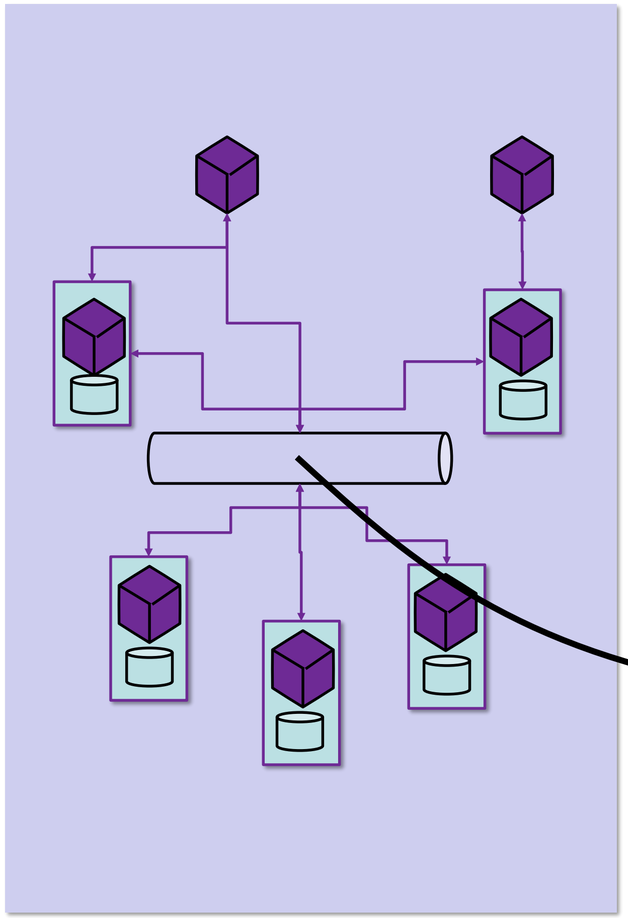
- **Extensibility**
- "**Generic**" **transport** not limited to HTTP
- **Asynchronous processing and invocation;** If your application needs to be continued after the first interaction with the API and/or if your application needs a guaranteed level of reliability and security then SOAP offers additional standards to ensure this type of operation.
- **Formal contracts**; if you have to define your own exchange format
- **Stateful operations;** if the application needs contextual information and conversational state management then SOAP

Orchestration or Reactive (aka Event-Driven)

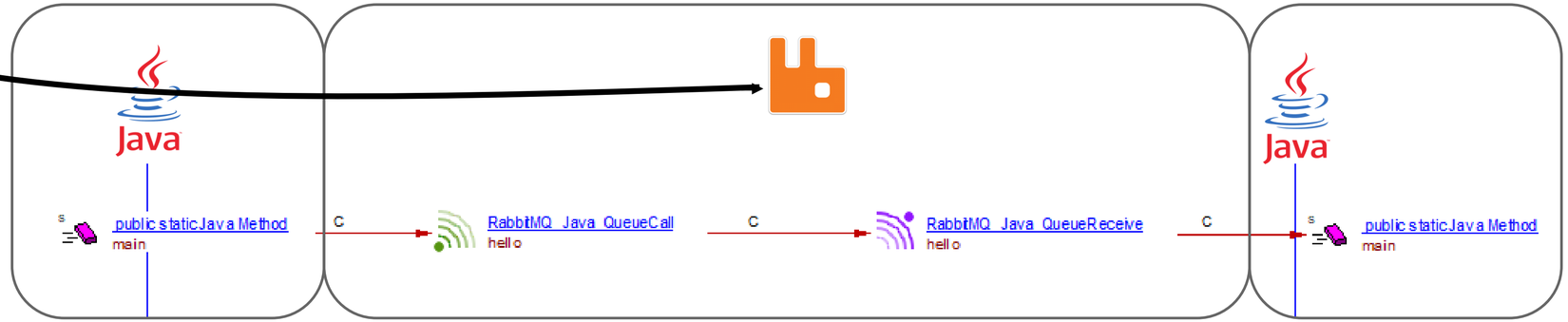


- *Orchestration*, there is typically **one controller** that acts as the "orchestrator" of the overall **service interactions**.
- Microservices should **minimize synchronous** invocations (for example, through REST) for intra-microservices communications to ensure the best possible isolation and atomicity.
- **Reactive** programming focuses on building **decoupled scalable** and **resilient** services. Focus on **asynchronously** and **parallel** use of sub microservices.
- **Reactive** programming removes the **blocking**, or **waiting**.
- Publish/subscribe system is usually performed by using an implementation of an event bus.
- *Recommended of an* Hybrid Model; mix of synchronous and asynchronous processing.

Identify Which Process can be Desynchronized



- There are no direct connections between programs.
- Communication between programs can be independent of time.
- Recovery support.
- Communication can be driven by events.
- Applications can assign a priority to a message.
- Security and Data integrity (unit of work to be committed or rolled back).
- Work can be carried out by small, self-contained programs.



Size Practical Concerns

A Bounded Context

Micro-service should support a specific functionality, this should be the prime trigger.

E Infrastructure

Micro-service use increase the infrastructure consumption. Creation of more layer should support more agility or more functionality.

B Team Size

Micro-service should be maintain and enhance by a small team, and lead his own roadmap.

D Data Consistency

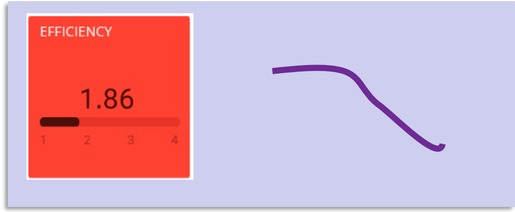
Consistency of data and transactions can only be ensured within a micro-service.

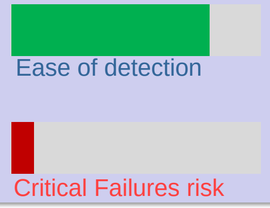
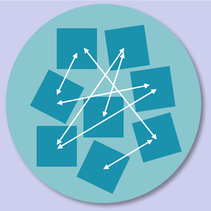
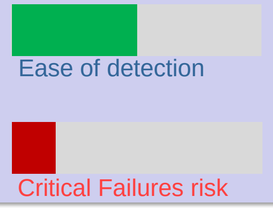
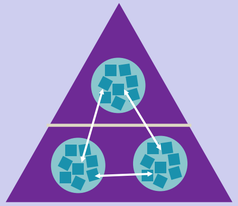
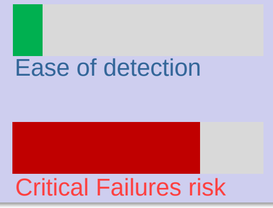
C Modularization - Replaceability

Functionalities within a micro-service should be used by more than one other module. Micro-service should be easy to understand in term of service contract (input and output).

Quality Practical Concerns

The use of many small, granular services can result in more inter-service communication.



Unit	Micro-Services	System
 	 	 
• Code at developer level	• Single technology layer • Micro-service level	• Entire application • Interaction and orchestration between application components and micro-services
• Code syntax • Code style & layout • Code hygiene • Code documentation • Common standards	• Component quality and structure • Some security vulnerabilities • Best coding practices	• Transaction integrity • Risk propagation • Entire application security • Inter-layer resiliency • Data access control • Architectural cohesion

Software Intelligence for Architect

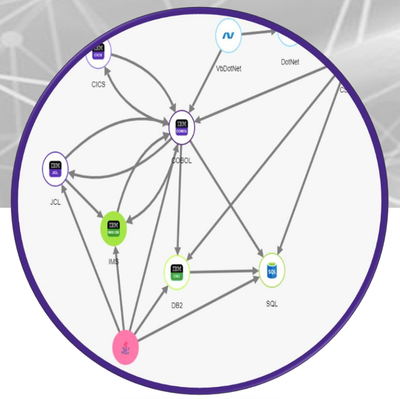
Portfolio Assessment

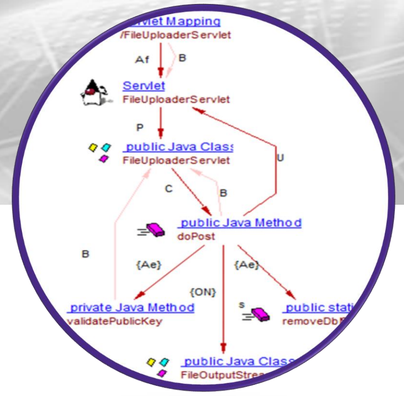
AS-IS Architecture (Discovery)

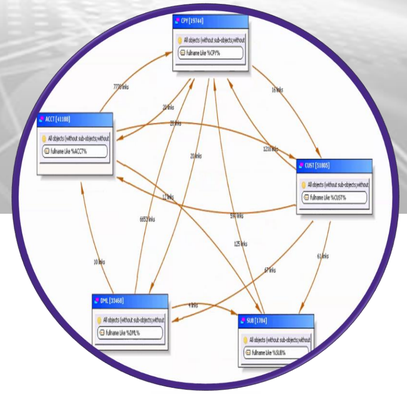
TO-BE Architecture (Design)

Implementation (Engineering)

Portfolio Management (Trends/Progress)







Thank You

Visit CAST at Booth #107

on the show floor

Philippe Guerin – Chris White

p.guerin@castsoftware.com – c.white@castsoftware.com

Software Intelligence for Digital Leaders

Software Architecture

softwarearchitecturecon.com | #OReillySACon

www.castsoftware.com