Refactor or replace

Determining the evolution of a software system

Bill Penberthy

September 30, 2020

Agenda

- Housekeeping
- Definitions and context
- Implementing the change
- Evaluating an application
- “Real-world” examples
- Applying to your application(s)

Definitions and Context

Refactoring

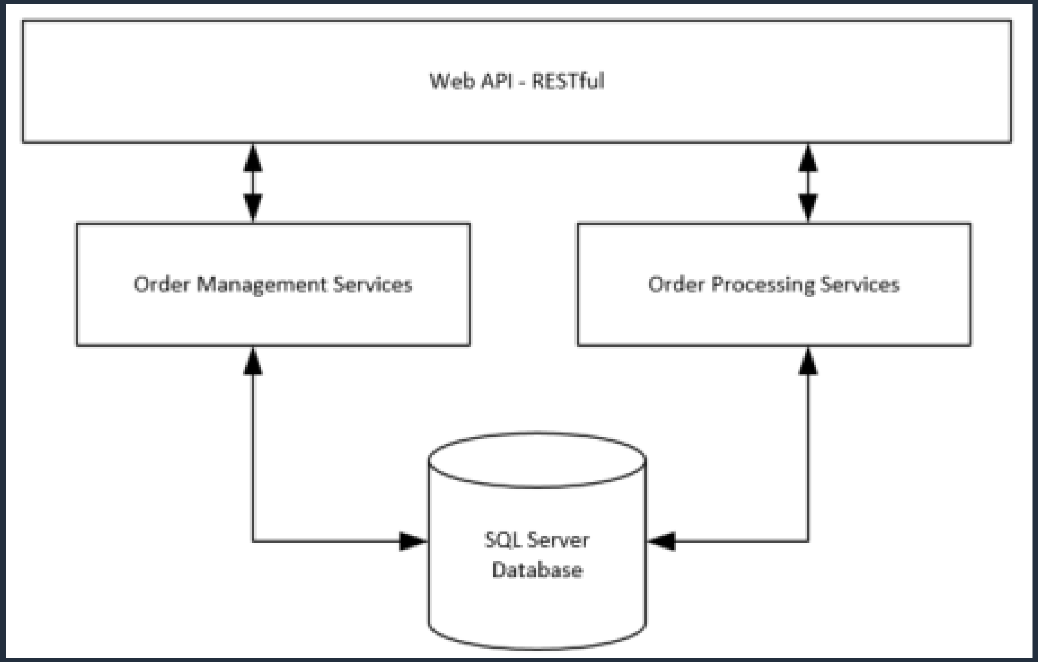
A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.

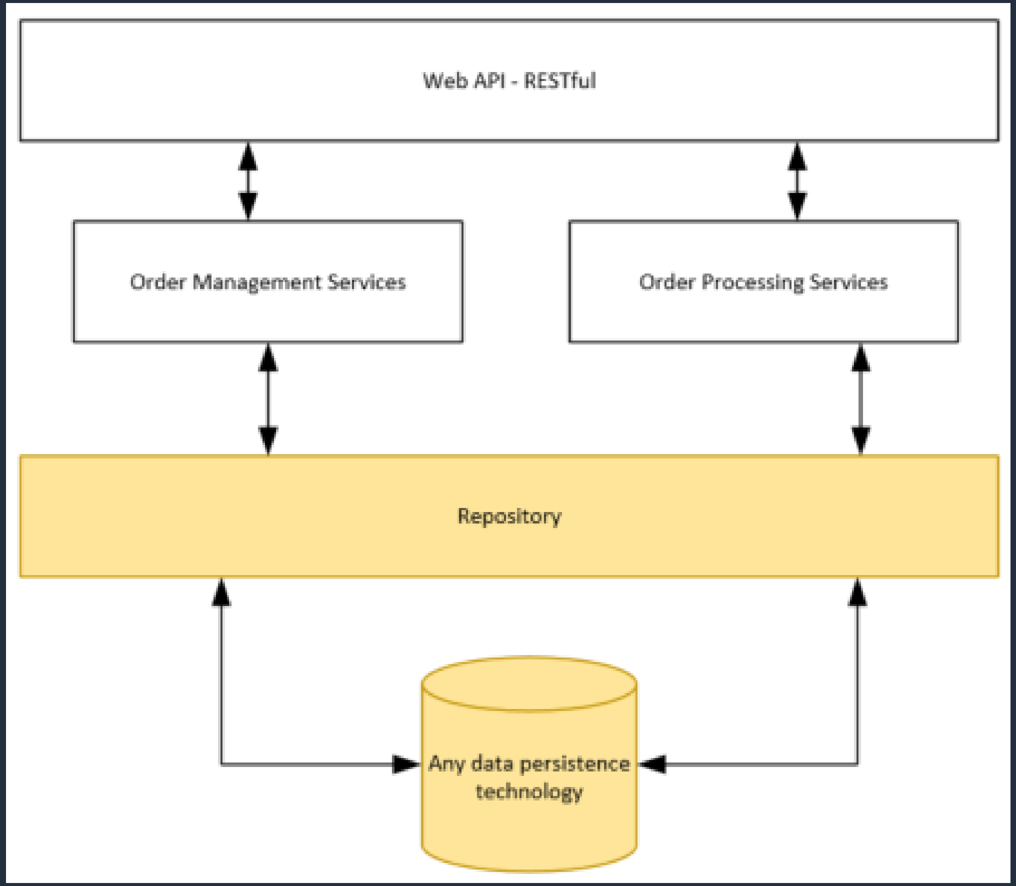
https://martinfowler.com/bliki/DefinitionOfRefactoring.html

Refactoring

A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.

https://martinfowler.com/bliki/DefinitionOfRefactoring.html



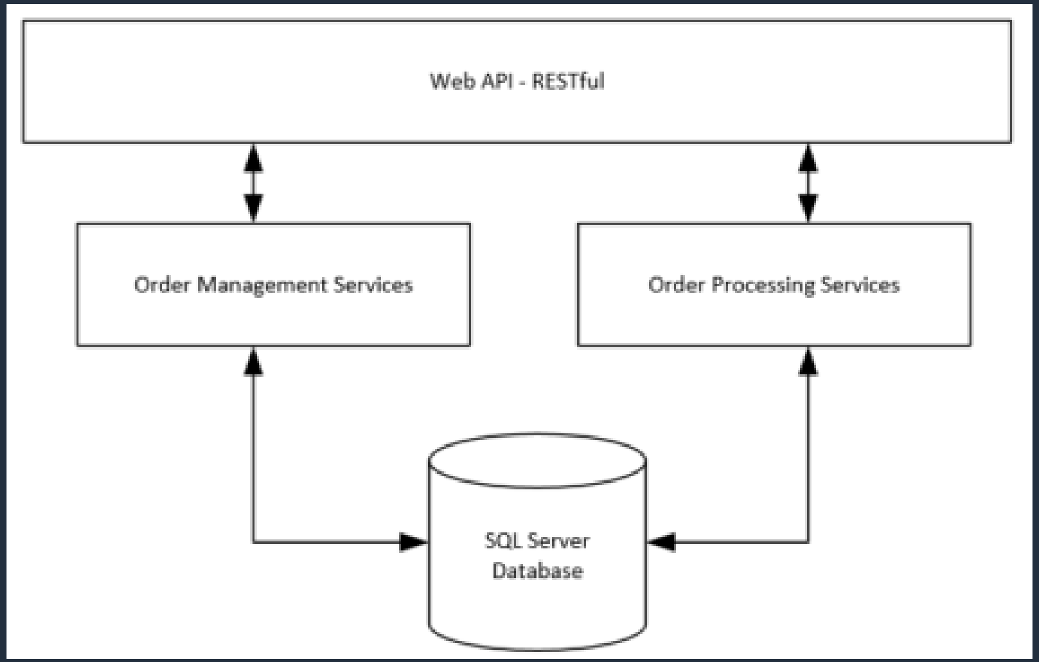

Web API - RESTful
Order Management Services
Order Processing Services
Repository
Any data persistence technology

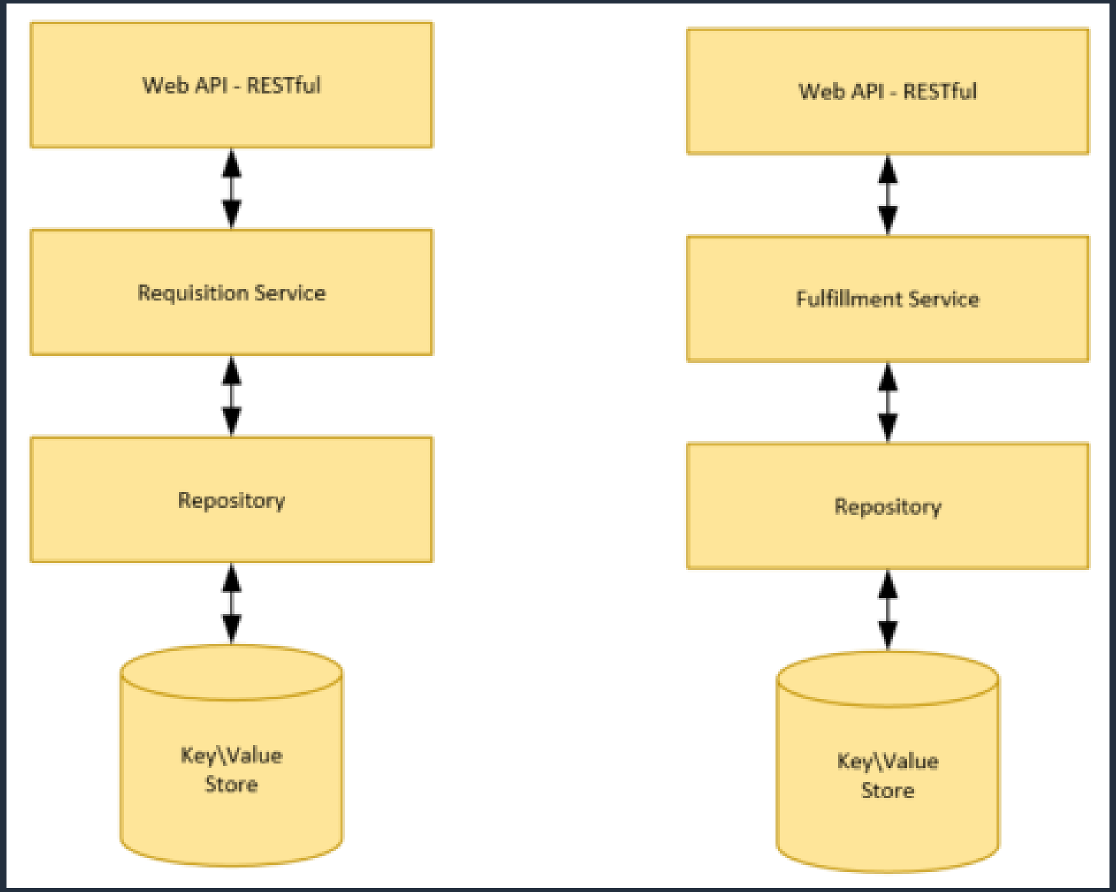
Replacing

Any change made to software that alters its already-existing observable behavior.

Replacing

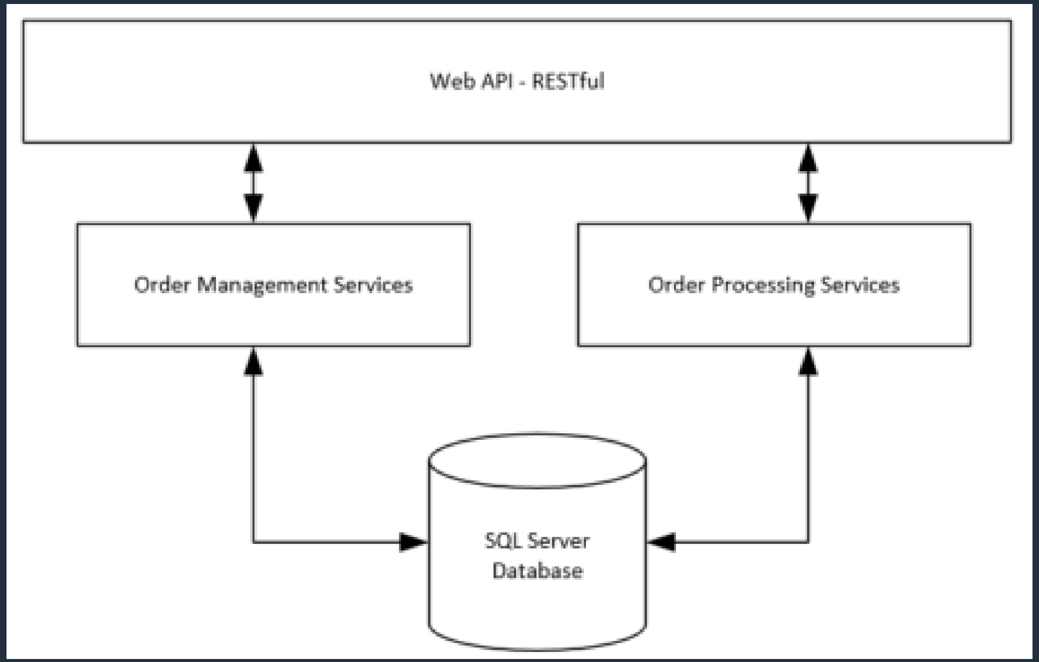
Any change made to software that alters its already-existing observable behavior.

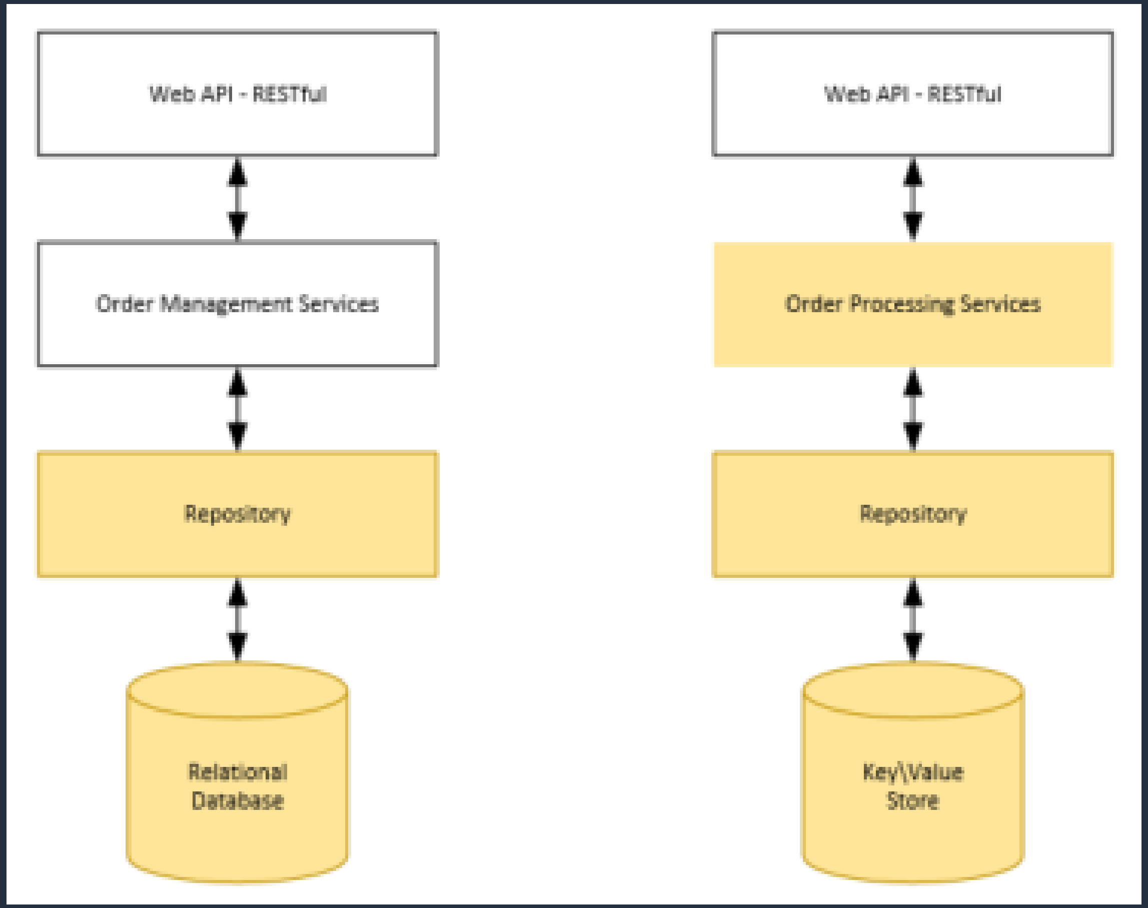




Replacing vs Refactoring

Sometimes it not as obvious…



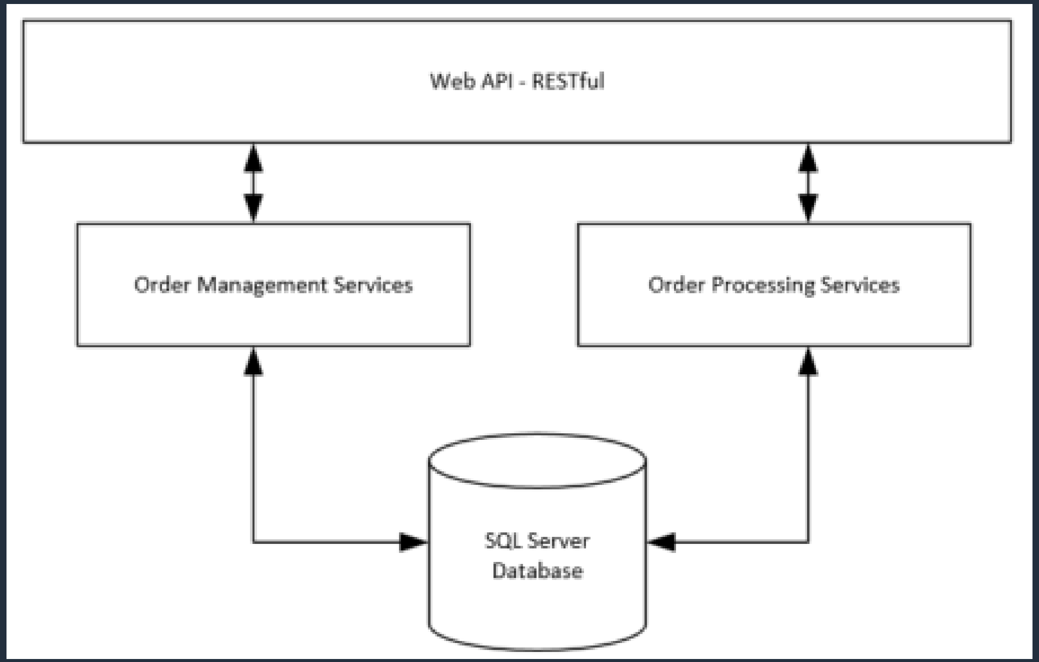


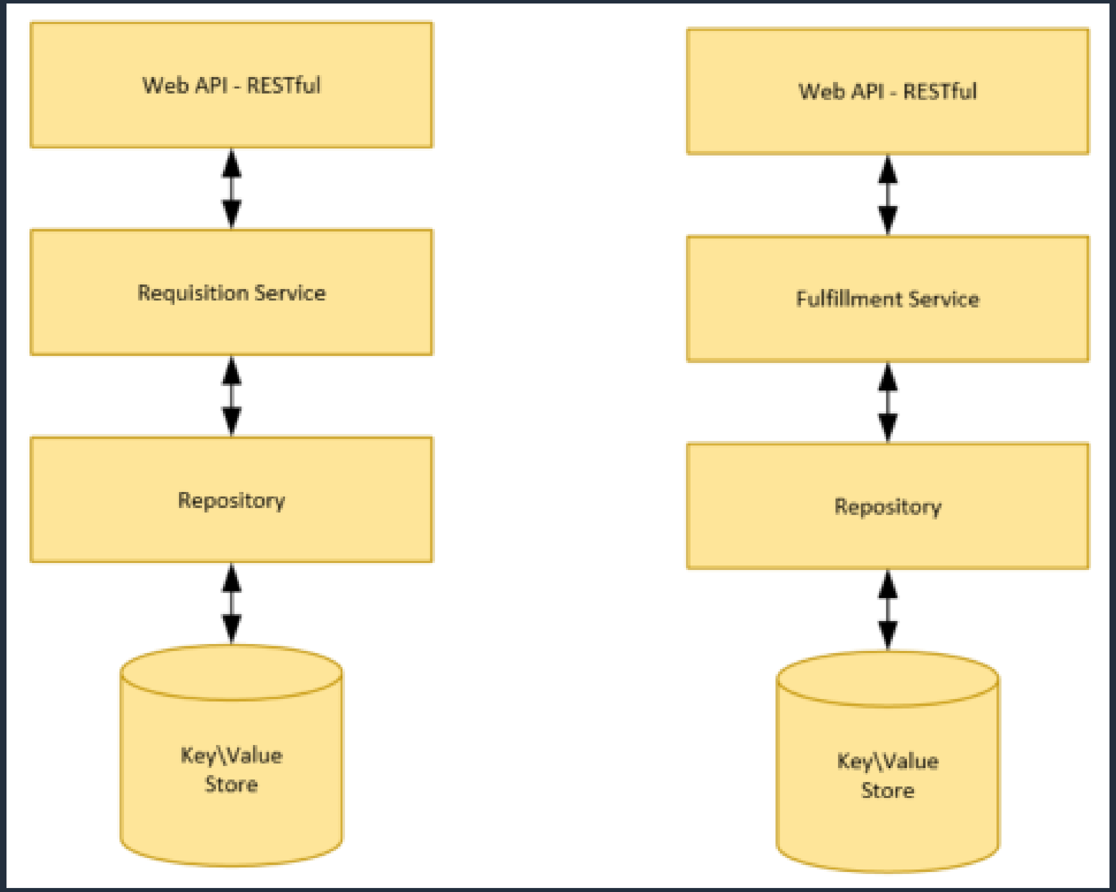
Context

Replacement vs Refactoring requires a definition of the software being evaluated.

Context

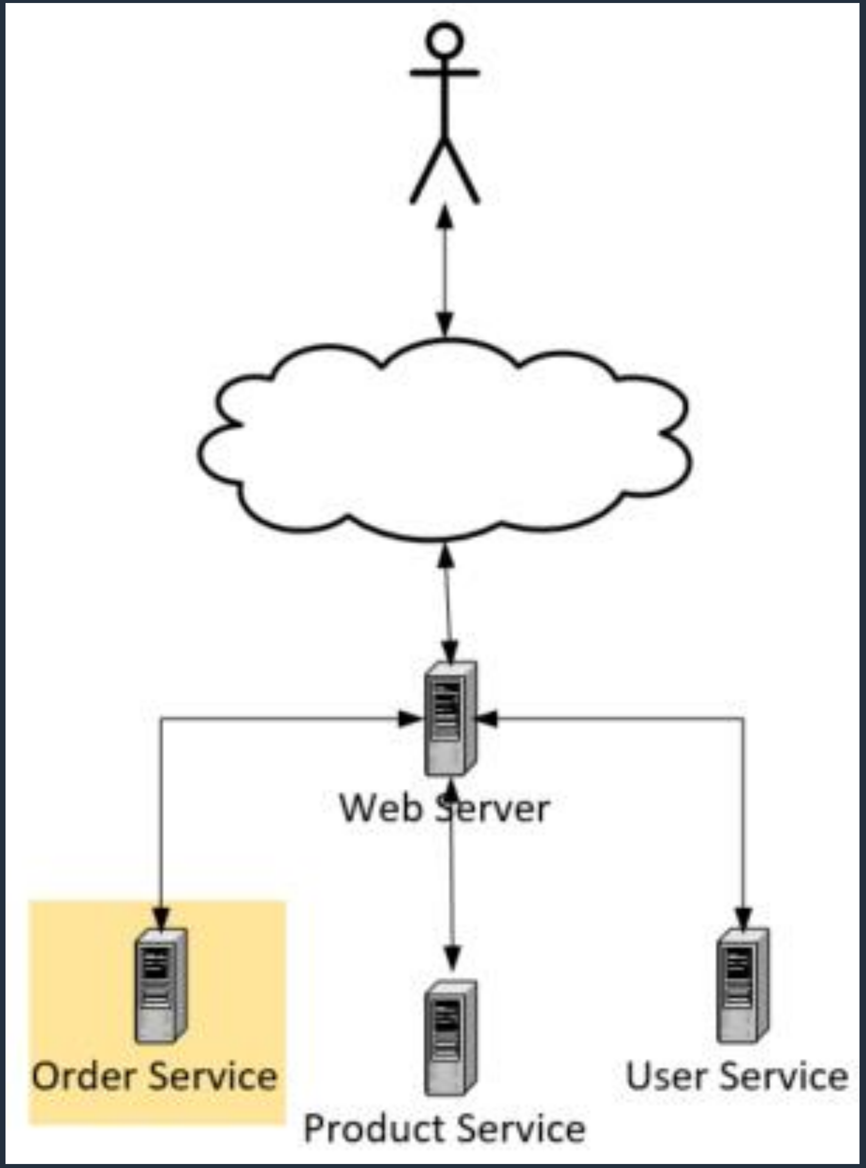
Replacement vs Refactoring requires a definition of the software being evaluated.





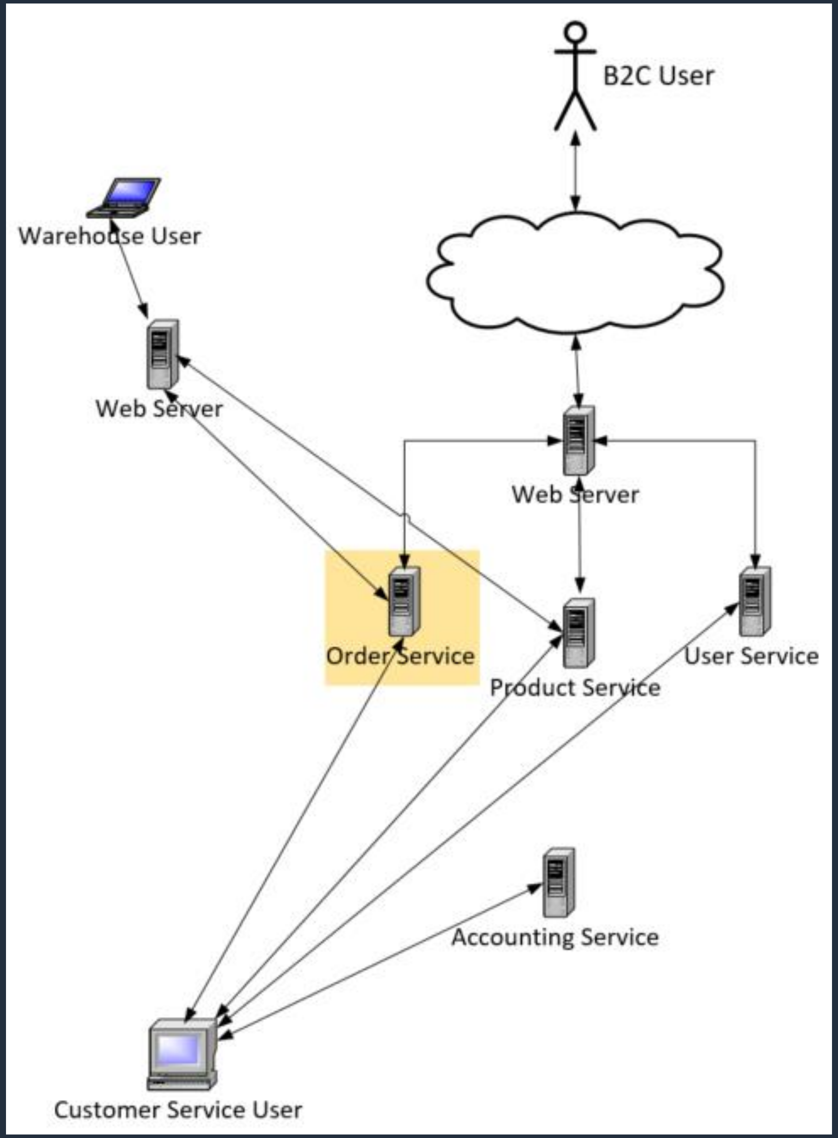
Context

Replacement vs Refactoring requires a definition of the software being evaluated.



Context

Observable behavior means observed by ?



Refactoring - New Definition

<u>Old</u>

A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.

Refactoring - New Definition

Old

A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.

New

Any change made to software that does not require that a person significantly change the way in which they use the software to do their work.

Replacing - New Definition

Old

Any change made to software that alters its already-existing observable behavior.

Replacing - New Definition

Old

Any change made to software that alters its already-existing observable behavior.

New

A change made to software that requires that a person significantly change they way in which they are doing their work OR the underlying structure of the application will change so much that it eliminates almost all currently existing technical debt

These definitions matter

- Gives an understanding of the end state of any systemic changes
- Defines potential implementation approaches for those changes

New Definitions

Refactor

Any change made to software that does not require that a person significantly change the way in which they use the software to do their work.

Replace

A change made to software that requires that a person significantly change they way in which they are doing their work OR the underlying structure of the application will change so much that it eliminates almost all currently existing technical debt

Implementing the change

Project Implementation Approaches

<u>Big Bang</u>

Build\Buy a new system, or part of a system, and swap in for the application being evaluated

Most common in:

- Buy decisions
- Re-platforming – changing underlying systems (esp. frameworks)
- Re-architecting – completely changing the way the application functions

You can stop\limit work on existing application during the construction period

Project Implementation Approaches

Incremental

Evolve the software through multiple iterations with each iteration getting closer to the end state

Project Implementation Approaches

Incremental

Evolve the software through multiple iterations with each iteration getting closer to the end state

If system is being replaced, then you are doing the replacement through a series of incremental changes

Project Implementation Approaches

Big Bang

Build\Buy a new system, or part of a system, and swap in for the application being evaluated

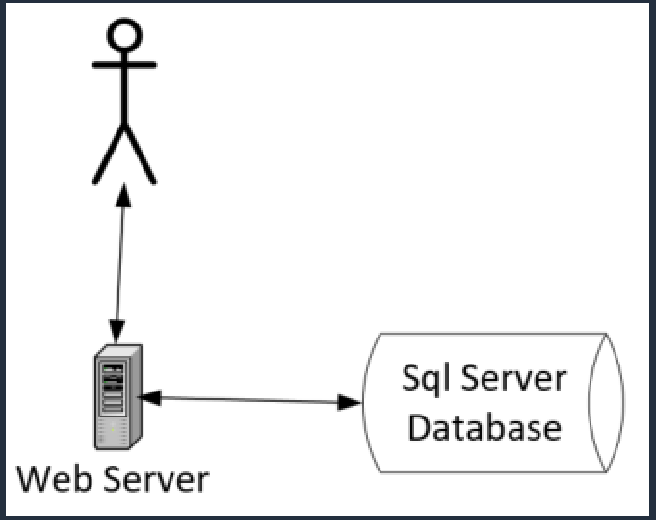
Incremental

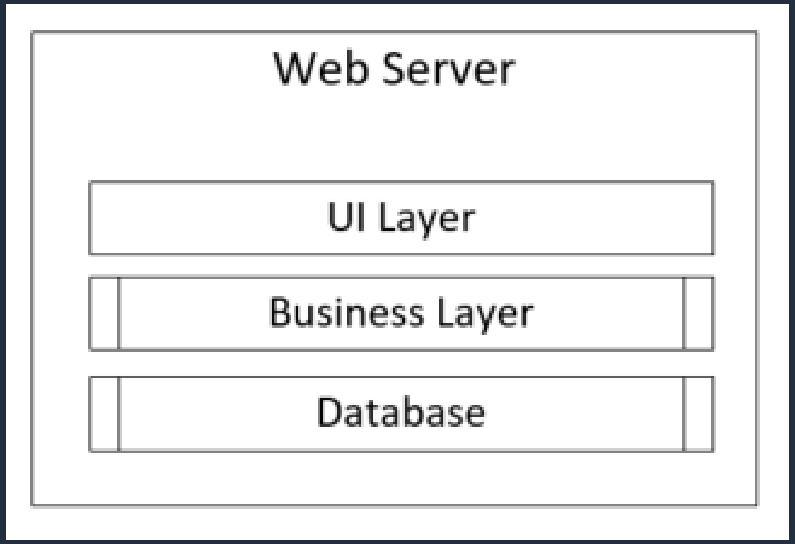
Evolve the software through multiple iterations with each iteration getting closer to the end state

Evaluating an application

Our discussion application

- B2C website designed to sell widgets





What you need to determine

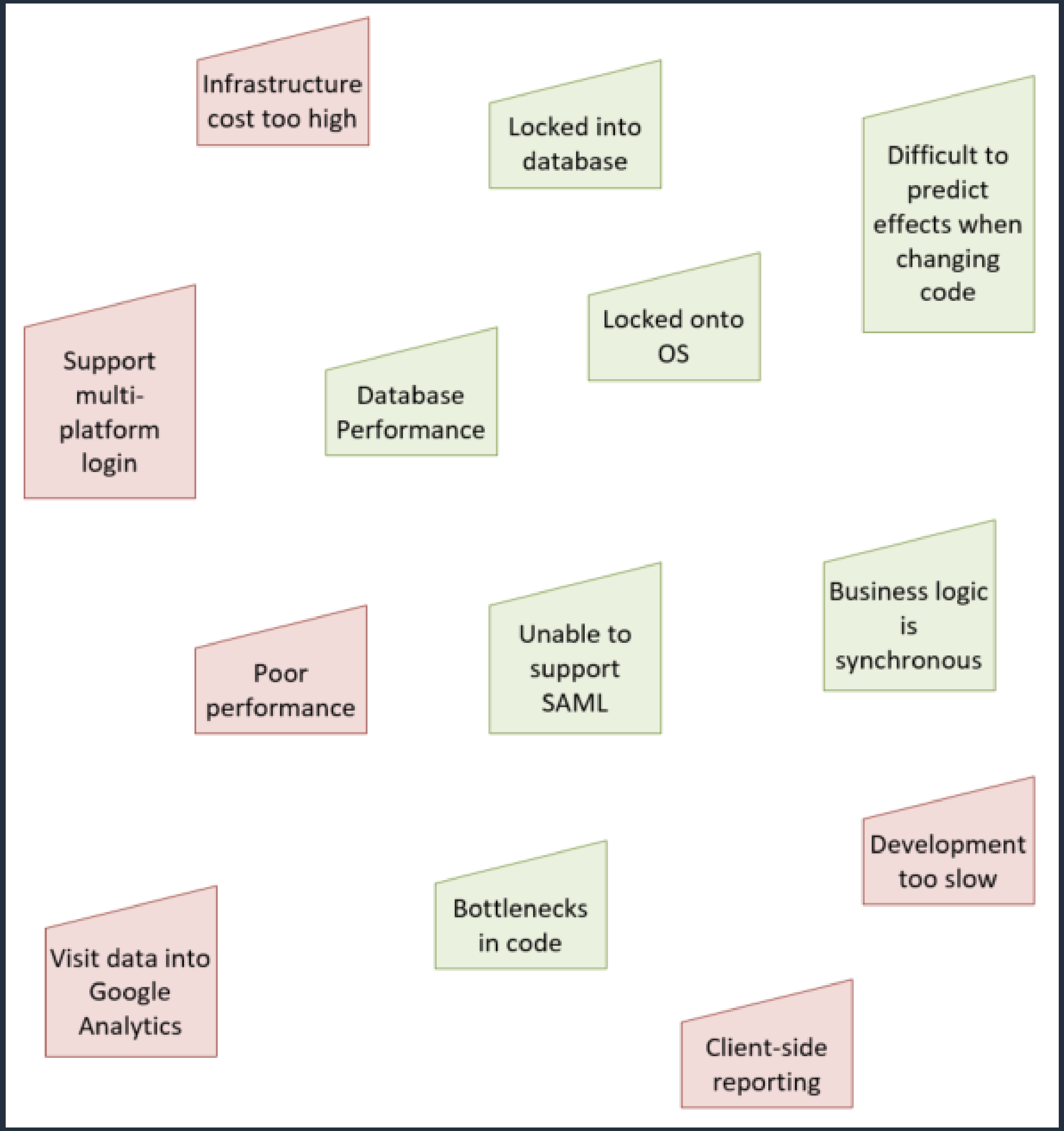
- What business problems do you currently have that is making you do this analysis?

- Example:
  - Spending too much money running the app
  - Business wants user to be able to authenticate using some 3rd party credentials
  - The site is generally slow, and there are times it gets so slow that it is almost unusable
  - All data about our visitors should be put into Google Analytics so the business can understand visitor behavior
  - We want to be able to provide users reporting about their orders and spend.
  - It takes too long for needed changes to get put into the system.

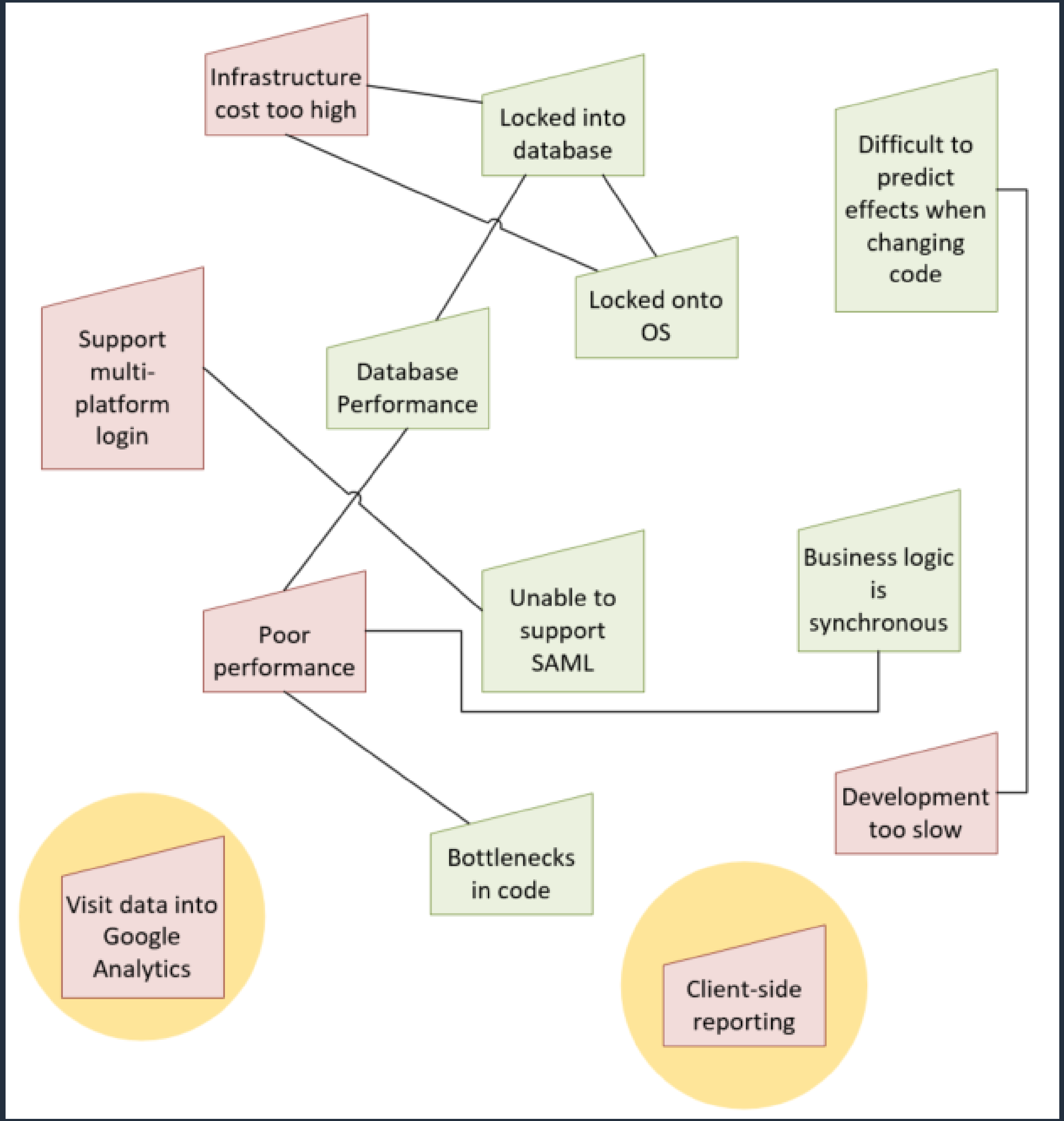
What you need to determine

- What business problems do you currently have that is making you do this analysis?
- What technical problems do you currently have that is making you do this analysis?

The technical and business problems


Infrastructure cost too high
Locked into database
Difficult to predict effects when changing code
Support multi-platform login
Database Performance
Locked onto OS
Poor performance
Unable to support SAML
Business logic is synchronous
Development too slow
Bottlenecks in code
Visit data into Google Analytics
Client-side reporting

The technical and business problems


Infrastructure cost too high
Locked into database
Difficult to predict effects when changing code
Support multi-platform login
Database Performance
Locked onto OS
Business logic is synchronous
Unable to support SAML
Poor performance
Development too slow
Bottlenecks in code
Visit data into Google Analytics
Client-side reporting

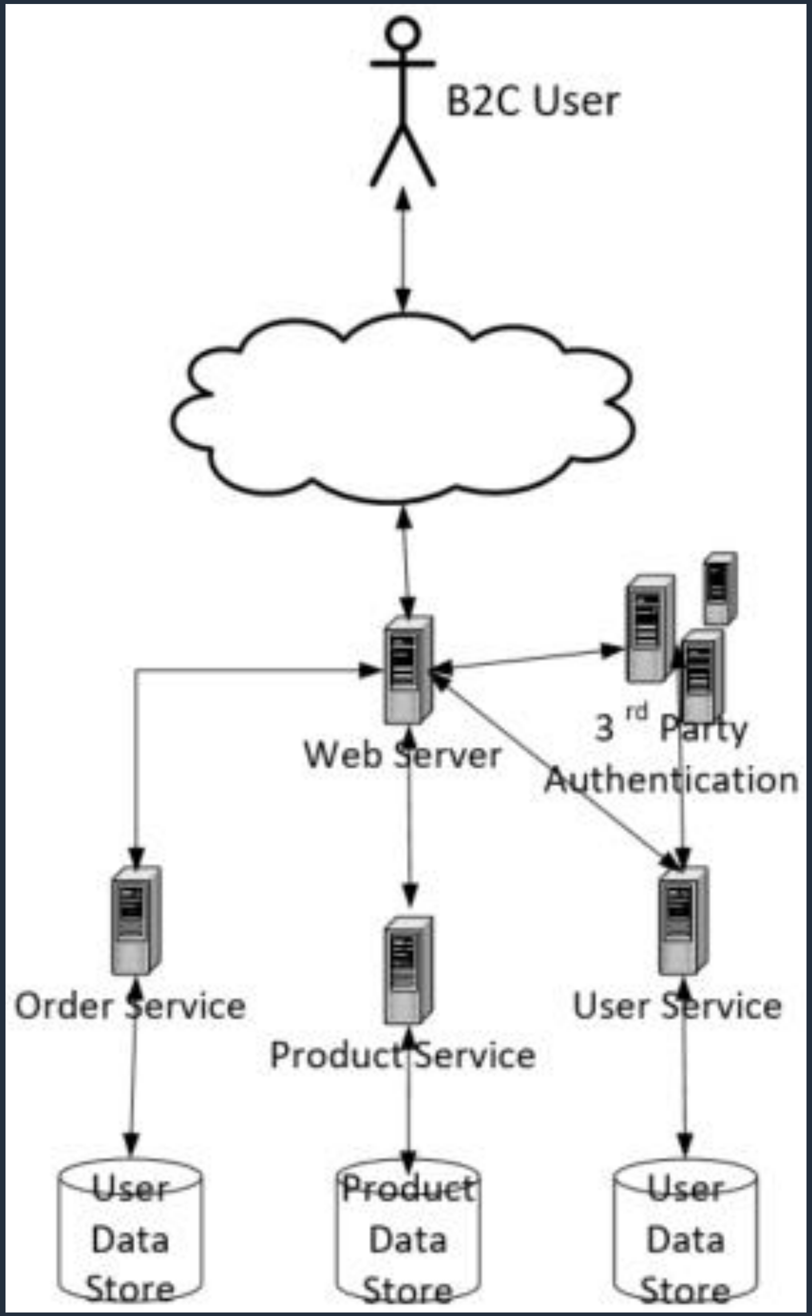
What you need to determine

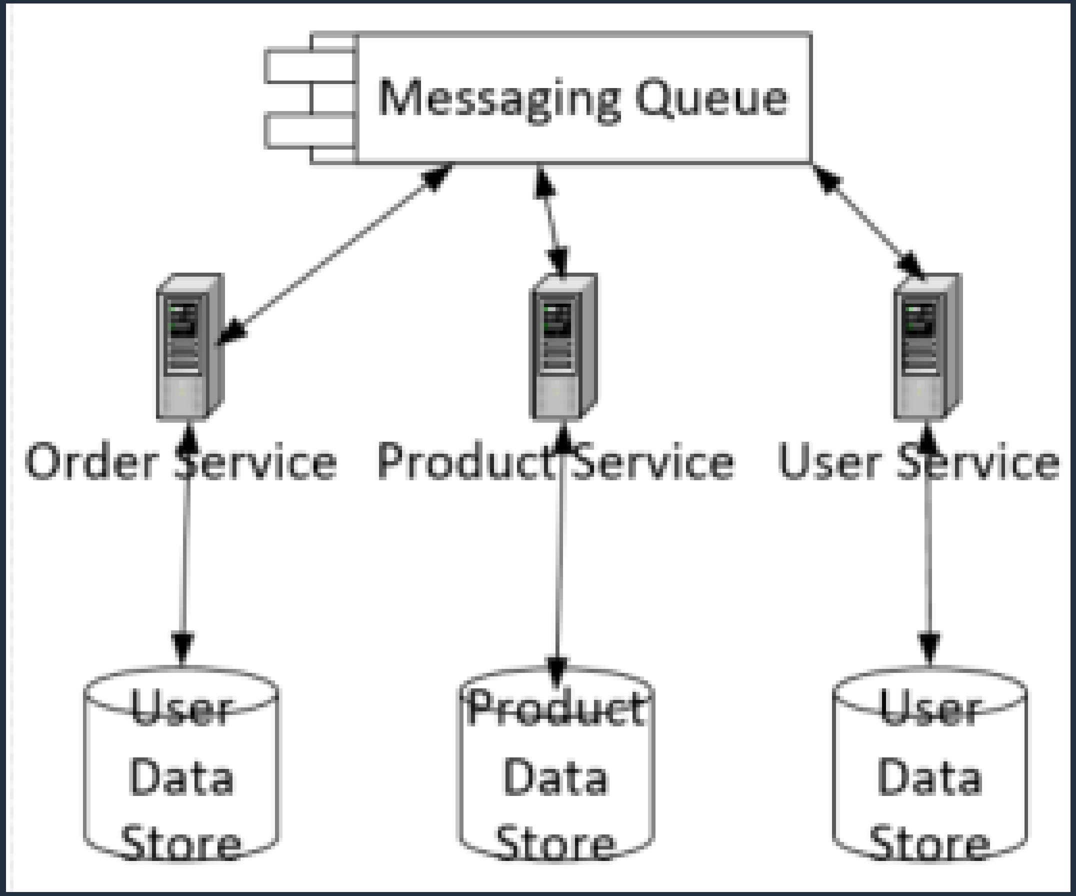
- What business problems do you currently have that is making you do this analysis?
- What technical problems do you currently have that is making you do this analysis?
- What other business needs are out there that do not directly relate to this application?

What you need to determine

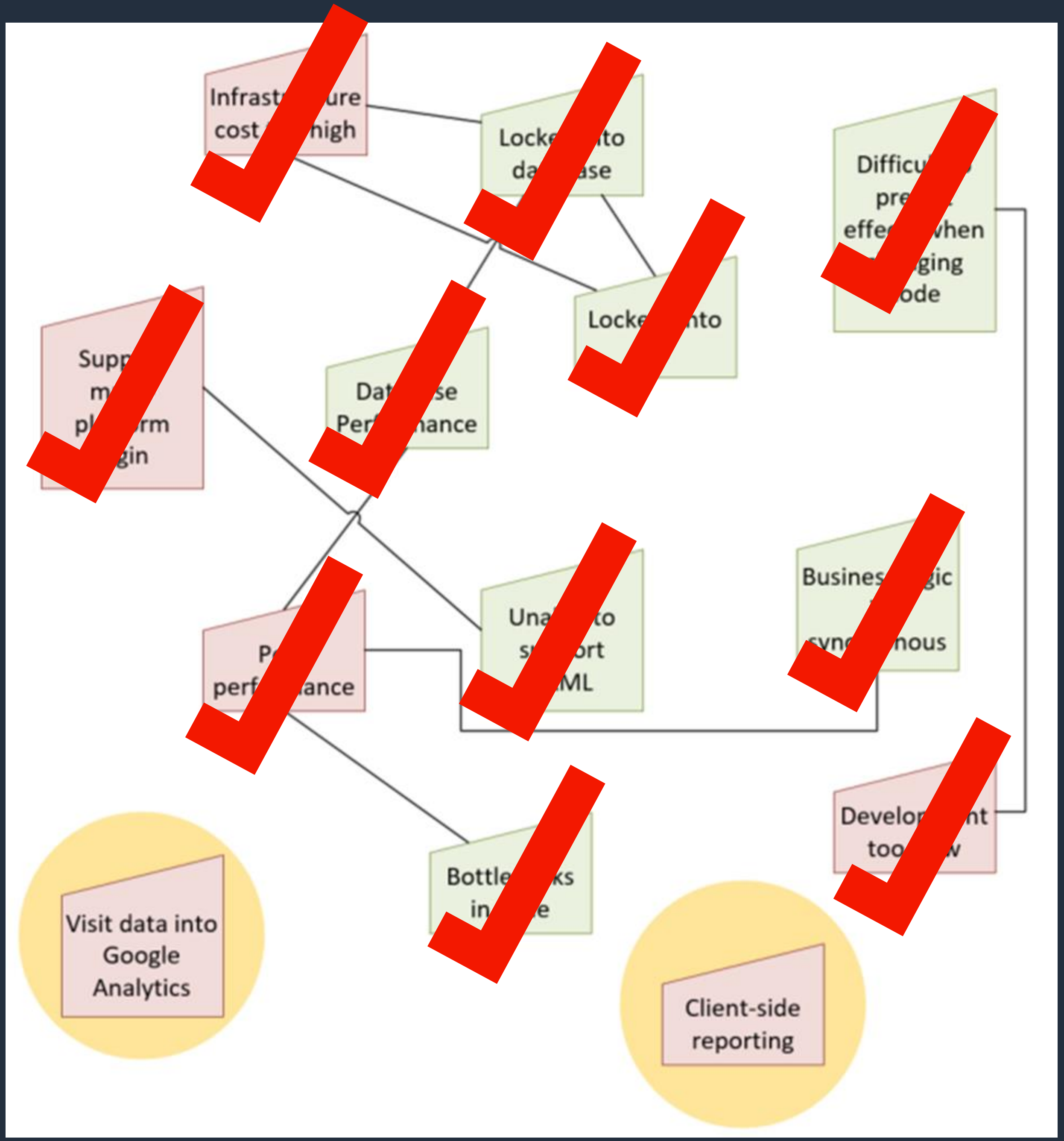
- What business problems do you currently have that is making you do this analysis?
- What technical problems do you currently have that is making you do this analysis?
- What other business needs are out there that do not directly relate to this application?
- What do you want everything to look like when you are *done done*?

Final state

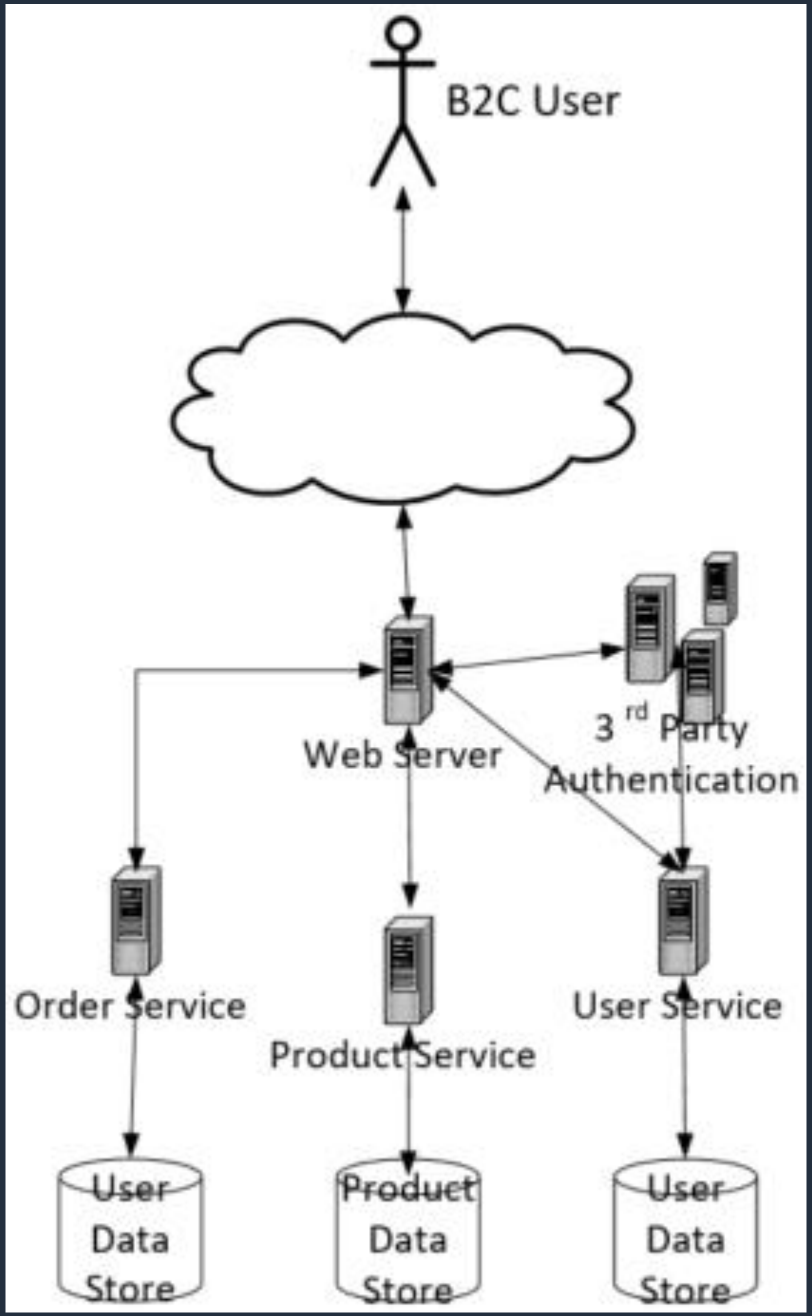

B2C User
Web Server
3rd Party Authentication
Order Service
Product Service
User Service
User Data Store
Product Data Store
User Data Store

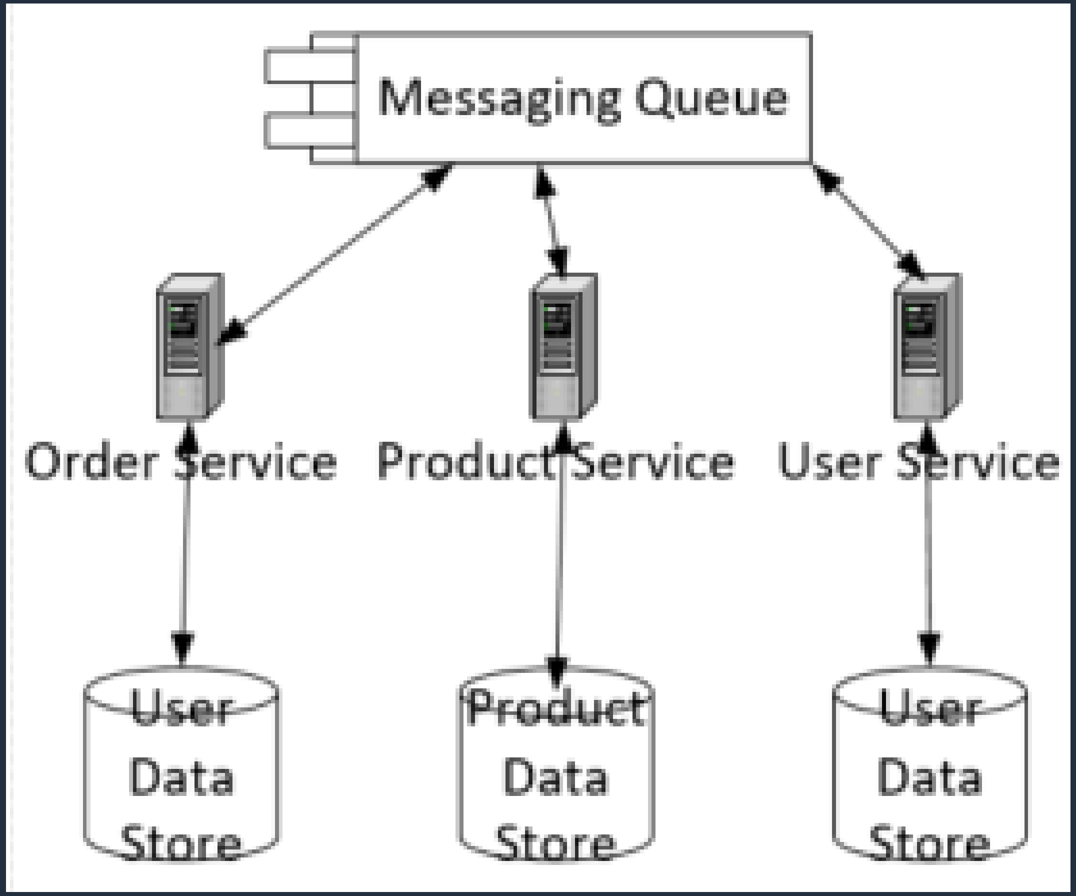

Messaging Queue
Order Service
Product Service
User Service
User Data Store
Product Data Store
User Data Store

Final state


Infrast ure cost high
Locke to da ase
Difficu pre effe when ging ode
Supp m pl rm gin
Dat se Per mance
Locke nto
P perf ance
Una to su ort ML
Busines gic sync nous
Develop nt too
Bottle ks in e
Visit data into Google Analytics
Client-side reporting

Final state


B2C User
Web Server
3rd Party Authentication
Order Service
Product Service
User Service
User Data Store
Product Data Store
User Data Store


Messaging Queue
Order Service
Product Service
User Service
User Data Store
Product Data Store
User Data Store

Replace or Refactor

Refactor

Any change made to software that does not require that a person significantly change the way in which they use the software to do their work.

Replace

A change made to software that requires that a person significantly change they way in which they are doing their work OR the underlying structure of the application will change so much that it eliminates almost all currently existing technical debt

Replace or Refactor

Refactor

Any change made to software that does not require that a person significantly change the way in which they use the software to do their work.

Replace

A change made to software that requires that a person significantly change they way in which they are doing their work OR **the underlying structure of the application will change so much that it eliminates almost all currently existing technical debt**

Real-world examples

Refactor with a Bang - Background

- Large SOA-based application

- Decided to do a limited re-architecture
    - ~40 business problems, 30 to be solved
    - ~70 technical problems, 35 to be solved

- Big-Bang implementation approach

Refactor with a Bang - Activities

- Create initial design for new service approach
- Build a proof-of-concept
- “Copy” business layer logic to REST service
- Update entry\exit points to updated model
- Update client library used by web application to call the service

Refactor with a Bang - Outcome

- Positive
    - Created the refactored Service
    - Launched service

Refactor with a Bang - Outcome

- Positive
  - Created the refactored Service
  - Launched service
- Negative
  - Scope creep
  - Imperfect architecture

Combo Approach - Replacement - Background

- Large SOA-based application
- Decided to do a limited re-architecture
  - ~12 business problems, 12 to be solved
  - ~60 technical problems, 60 to be solved
  - Multiple other business considerations
- Incremental replacement implementation approach

Combo Approach - Replacement - Activities

- Create new version of website (CMS)
- Update back-end services
  - Create an orchestration service for each property
  - Change intra-service communications to be message-based
  - Add service-specific persistence mechanisms
- Update existing website to use new back-end services as they roll out

Combo Approach - Replacement - Outcome

- Positive
  - Finished the Website re-platforming
  - Finished the back-end service conversion to be eventually consistent
  - Increased perceived performance by ~300%

Combo Approach - Replacement - Outcome

- Positive
  - Finished the Website re-design
  - Finished the back-end service conversion to be eventually consistent
  - Increased perceived performance by ~300%

- Negative
  - Way longer to do any of the paths than expected
  - Missing features
  - Poor communication between business and technology

Applying to your application(s)

List your problems

- What business problems do you currently have that is making you do this analysis?

- What technical problems do you currently have that is making you do this analysis?

Define the relationships between the problems

- Technical problems may be related to each other
- Business problems may be related to technical problems
- Pull out business problems that are not directly blocked by technical problems

Define other business thoughts around the domain

- Everything you pulled out in the last step
- Other things the business may be considering

Design the *Done Done* Application

- Create a high-level technical design of your desired end state
    - You don’t have to define APIs
    - Or database table structure
    - Or other lower-level constructs

- Relate this end-state back to the list of problems that you have.

- Account for as many of the problems and future considerations as possible

Evaluate implementation approach

- What work has to be done concurrently on the application?
- Does the team have the knowledge to build the *done done* application?
- How big of a change in the way they do their work is it for the users of the system?
- Is there support for a 2-system approach?
- How much work needs to be done?
- Do you think it is possible to iterate from current state to new state?

Summary

Refactor vs Replace – Make it the same journey

Focus on the order in which you will fix the problems

Strive towards your ideal system

Update your ideal system as needed

Always improve

Q&A

Bill Penberthy

@BillVest
https://www.linkedin.com/in/billpenberthy
enterprise-das@amazon.com

Thank you!