O'REILLY®
Software Architecture

META for microservices

Getting your enterprise migration in motion

Matt McLarty, Chief Architect, Islay Bank
@mattmclartybc

oreillysacon.com/ny
#OReillySACon

Agenda

- Dealing with Complexity in the Digital Age
- META: Microservice-based Enterprise Transformation Approach
- Program Design
- System Design
- Service Design
- Foundation Design
- Practice Design
- Next Steps

The Digital Age

THE WALL STREET JOURNAL.

Why Software Is Eating The World

By MARC ANDREESSEN

This week, Hewlett-Packard (where I am on the board) announced that it is exploring jettisoni PC business in favor of investing more heavily in software, where it sees better potential for gr Google plans to buy up the cellphone handset maker Motorola Mobility. Both moves surprised But both moves are also in line with a trend I've observed, one that makes me optimistic about of the American and world economies, despite the recent turmoil in the stock market.

In an interview with WSJ's Kevin Delaney, Groupon and LinkedIn investor Marc Andreessen insists that the recent popularity of tech companies does not constitute a bubble. He also stressed that both Apple and Google are undervalued and that "the market doesn't like tech."

In short, software is eating the world.

More than 10 years after the peak of the 1990s a dozen or so new Internet companies like Fac Twitter are sparking controversy in Silicon Va rapidly growing private market valuations, an occasional successful IPO. With scars from th Webvan and Pets.com still fresh in the investc are asking, "Isn't this just a dangerous new bu

I, along with others, have been arguing the otl case. (I am co-founder and general partner of firm Andreessen-Horowitz, which has investe

Forbes

Now Every Company Is A Software Company

Techonomy

How Tech Transforms Business and Society FULL BIO

Opinions expressed by Forbes Contributors are their own.

David Kirkpatrick, Contributor

Ford sells computers-on-wheels. McKinsey hawks consulting-in-a-box. FedEx boasts a developer skunkworks. The era of separating traditional industries and technology industries is over—and those who fail to adapt right now will soon find themselves obsolete.

GOVERNMENT

Disrupting the Public Sector

by William D. Eggers and Ruben Gonzalez

MARCH 08, 2012

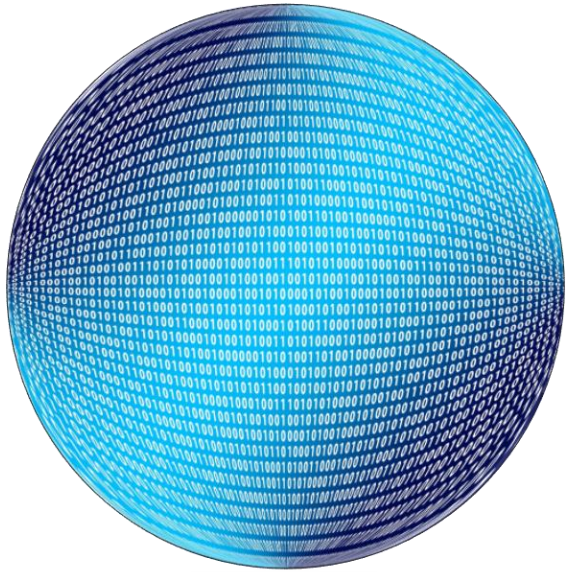
Harvard Business Review

SAVE SHARE 0 COMMENT TEXT SIZE PRINT

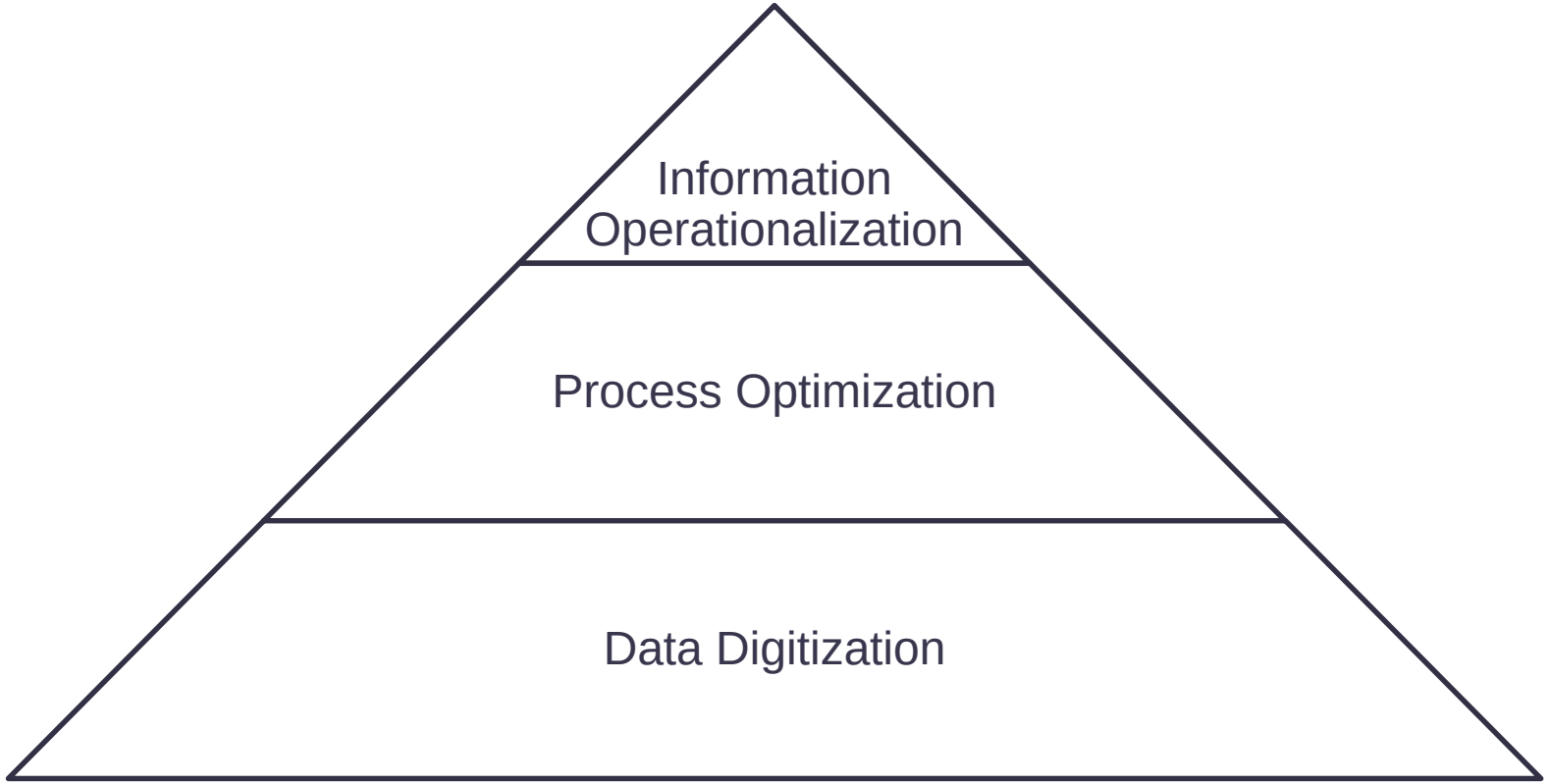
Thanks to disruptive innovations, much of our world today looks radically different than it did just a decade or two ago. Remember flying in the old days? Air travel used to be inevitably expensive and cumbersome -- until Southwest Airlines. Trips to the video store and looming late fees are now a distant memory, thanks to Netflix.

Digital Transformation

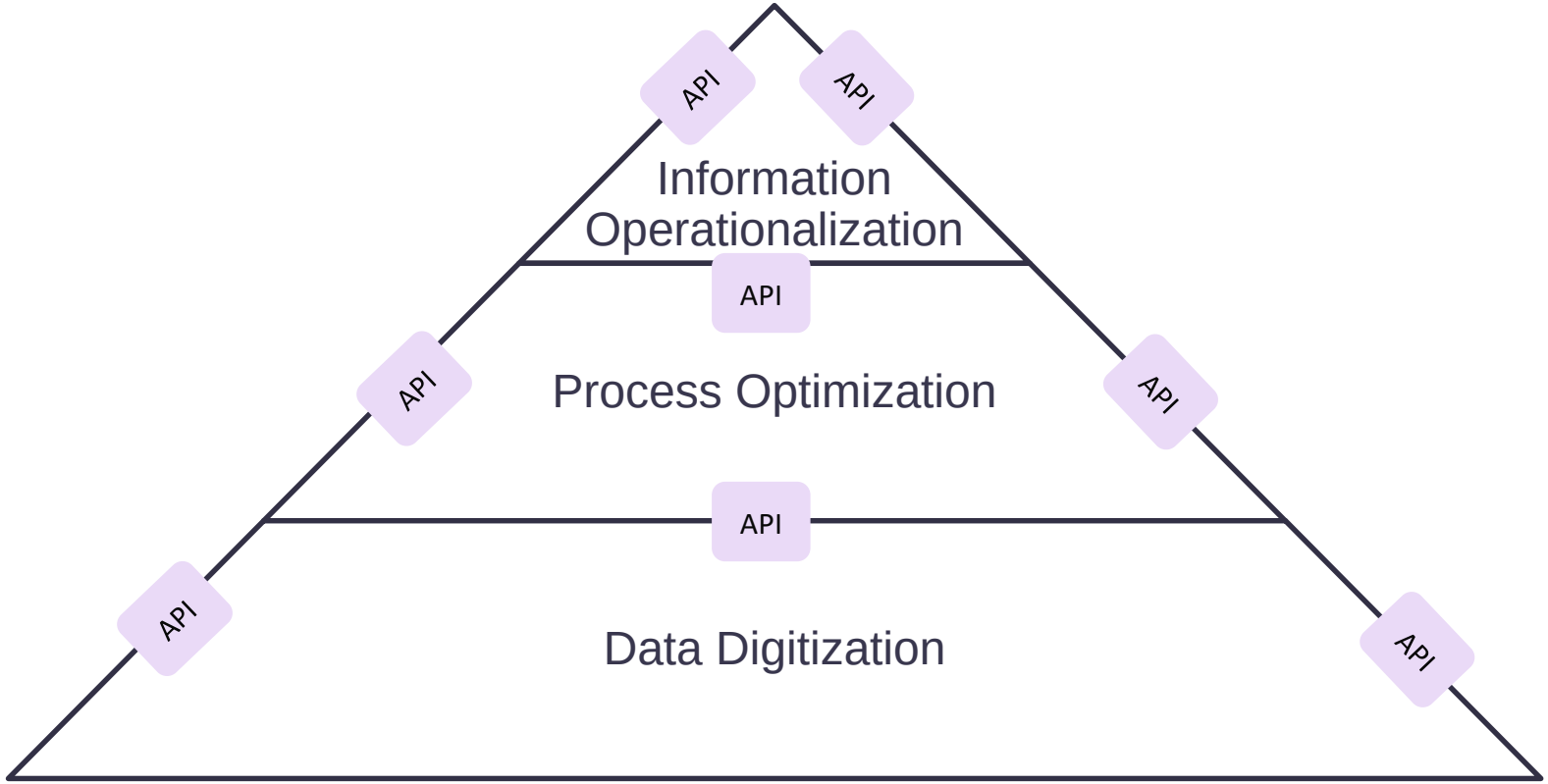
- A movement to make every business a software business
  - *Not just a business that uses software*
- The primary goals are innovation and optimization
- Cloud computing, microservices, and APIs are technology enablers



Digital Transformation



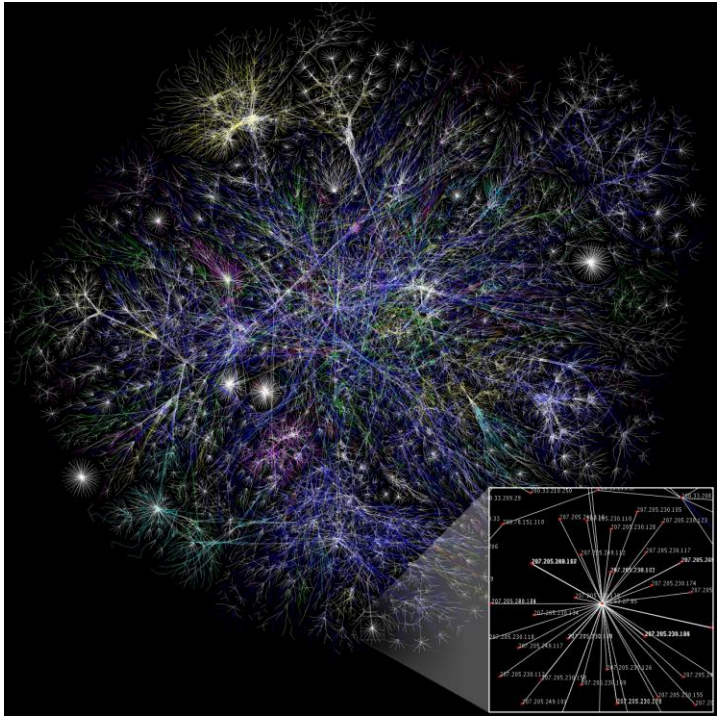
Digital Transformation



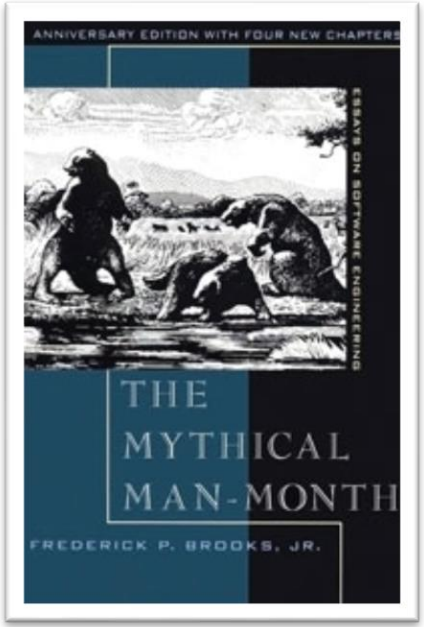
Barriers to Digital Transformation

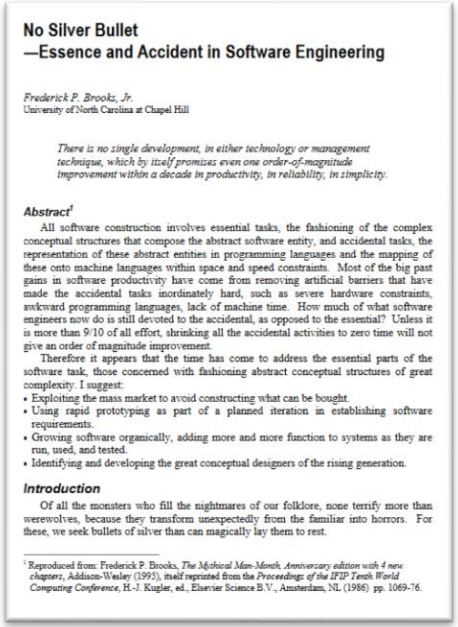
- Highly-integrated applications and data
- Legacy technology, packaged solutions, skill gaps
- Large, hierarchical organizations
- Outsourcing, geographical distribution
- Industry regulations, shareholder/public scrutiny

Complexity



Dealing with Complexity





No Silver Bullet
—Essence and Accident in Software Engineering

Frederick P. Brooks, Jr.
University of North Carolina at Chapel Hill

There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity.

Abstract[1]

All software construction involves essential tasks, the fashioning of the complex conceptual structures that compose the abstract software entity, and accidental tasks, the representation of these abstract entities in programming languages and the mapping of these onto machine languages within space and speed constraints. Most of the big past gains in software productivity have come from removing artificial barriers that have made the accidental tasks inordinately hard, such as severe hardware constraints, awkward programming languages, lack of machine time. How much of what software engineers now do is still devoted to the accidental, as opposed to the essential? Unless it is more than 9/10 of all effort, shrinking all the accidental activities to zero time will not give an order of magnitude improvement.

Therefore it appears that the time has come to address the essential parts of the software task, those concerned with fashioning abstract conceptual structures of great complexity. I suggest:

- Exploiting the mass market to avoid constructing what can be bought.
- Using rapid prototyping as part of a planned iteration in establishing software requirements.
- Growing software organically, adding more and more function to systems as they are run, used, and tested.
- Identifying and developing the great conceptual designers of the rising generation.

Introduction

Of all the monsters who fill the nightmares of our folklore, none terrify more than werewolves, because they transform unexpectedly from the familiar into horrors. For these, we seek bullets of silver than can magically lay them to rest.

[1] Reproduced from: Frederick P. Brooks, *The Mythical Man-Month, Anniversary edition with 4 new chapters*, Addison-Wesley (1995), itself reprinted from the *Proceedings of the IFIP Tenth World Computing Conference*, H.-J. Kugler, ed., Elsevier Science B.V., Amsterdam, NL (1986) pp. 1069-76.

Essential vs. Accidental Complexity

Essential Complexity

- The complexity of the software's functional scope and the problems it solves (e.g. correlating and analyzing large amounts of data in real time)

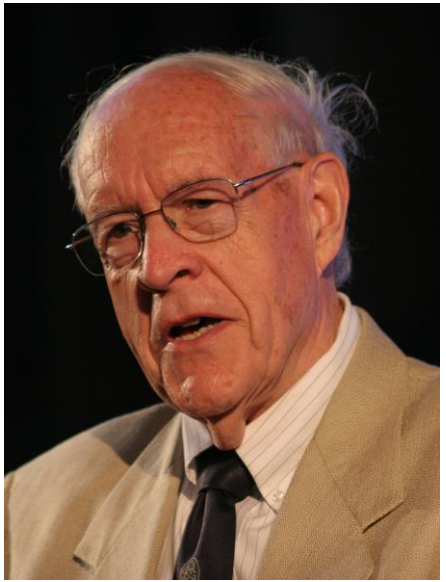
Accidental Complexity

- The complexity of the software's implementation details (e.g. the languages, processes and messages used to do the work)

The Truth About Software Complexity

"Many of the classical problems of developing software products derived from this essential complexity and its nonlinear increase with size."

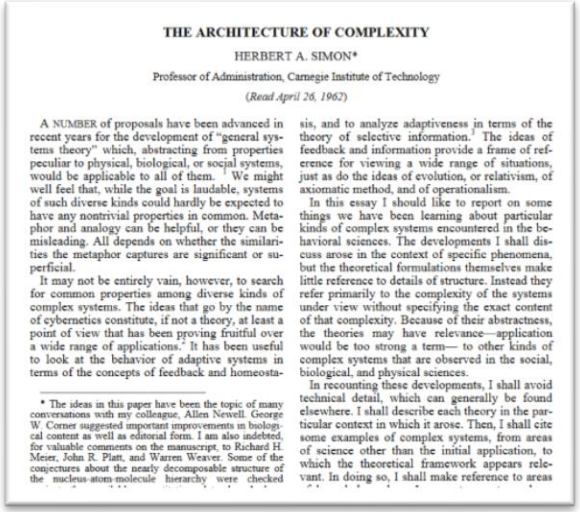
- Fred Brooks, No Silver Bullet—Essence and Accident in Software Engineering



Dealing with Complexity

- **Lesson #1 – Differentiate the complexity**

Abstraction, Hierarchy and Modularization



THE ARCHITECTURE OF COMPLEXITY

HERBERT A. SIMON*

Professor of Administration, Carnegie Institute of Technology

(*Read April 26, 1962*)

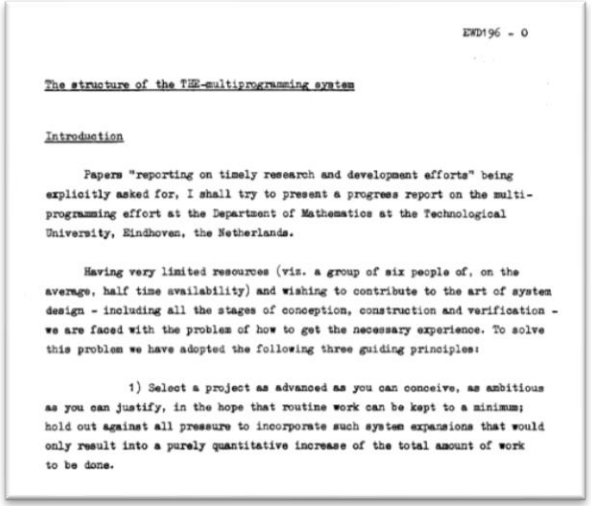
A NUMBER of proposals have been advanced in recent years for the development of "general systems theory" which, abstracting from properties peculiar to physical, biological, or social systems, would be applicable to all of them. [1] We might well feel that, while the goal is laudable, systems of such diverse kinds could hardly be expected to have any nontrivial properties in common. Metaphor and analogy can be helpful, or they can be misleading. All depends on whether the similarities the metaphor captures are significant or superficial.

It may not be entirely vain, however, to search for common properties among diverse kinds of complex systems. The ideas that go by the name of cybernetics constitute, if not a theory, at least a point of view that has been proving fruitful over a wide range of applications.[2] It has been useful to look at the behavior of adaptive systems in terms of the concepts of feedback and homeostasis, and to analyze adaptiveness in terms of the theory of selective information.[3] The ideas of feedback and information provide a frame of reference for viewing a wide range of situations, just as do the ideas of evolution, or relativism, of axiomatic method, and of operationalism.

In this essay I should like to report on some things we have been learning about particular kinds of complex systems encountered in the behavioral sciences. The developments I shall discuss arose in the context of specific phenomena, but the theoretical formulations themselves make little reference to details of structure. Instead they refer primarily to the complexity of the systems under view without specifying the exact content of that complexity. Because of their abstractness, the theories may have relevance—application would be too strong a term— to other kinds of complex systems that are observed in the social, biological, and physical sciences.

In recounting these developments, I shall avoid technical detail, which can generally be found elsewhere. I shall describe each theory in the particular context in which it arose. Then, I shall cite some examples of complex systems, from areas of science other than the initial application, to which the theoretical framework appears relevant. In doing so, I shall make reference to areas

* The ideas in this paper have been the topic of many conversations with my colleague, Allen Newell. George W. Corner suggested important improvements in biological content as well as editorial form. I am also indebted, for valuable comments on the manuscript, to Richard H. Meier, John R. Platt, and Warren Weaver. Some of the conjectures about the nearly decomposable structure of the nucleus-atom-molecule hierarchy were checked



EWD196 - 0

The structure of the THE-multiprogramming system

Introduction

Papers "reporting on timely research and development efforts" being explicitly asked for, I shall try to present a progress report on the multiprogramming effort at the Department of Mathematics at the Technological University, Eindhoven, the Netherlands.

Having very limited resources (viz. a group of six people of, on the average, half time availability) and wishing to contribute to the art of system design - including all the stages of conception, construction and verification - we are faced with the problem of how to get the necessary experience. To solve this problem we have adopted the following three guiding principles:

1) Select a project as advanced as you can conceive, as ambitious as you can justify, in the hope that routine work can be kept to a minimum; hold out against all pressure to incorporate such system expansions that would only result into a purely quantitative increase of the total amount of work to be done.

Programming Techniques

R. Morris
Editor

On the Criteria To Be Used in Decomposing Systems into Modules

D.L. Parnas
Carnegie-Mellon University

This paper discusses modularization as a mechanism for improving the flexibility and comprehensibility of a system while allowing the shortening of its development time. The effectiveness of a "modularization" is dependent upon the criteria used in dividing the system into modules. A system design problem is presented and both a conventional and unconventional decomposition are described. It is shown that the unconventional decompositions have distinct advantages for the goals outlined. The criteria used in arriving at the decompositions are discussed. The unconventional decomposition, if implemented with the conventional assumption that a module consists of one or more subroutines, will be less efficient in most cases. An alternative approach to implementation which does not have this effect is sketched.

Key Words and Phrases: software, modules, modularity, software engineering, KWIC index, software design

CR Categories: 4.0

Introduction

A lucid statement of the philosophy of modular programming can be found in a 1970 textbook on the design of system programs by Gouthier and Pont [1, ¶10.23], which we quote below:[2]

A well-defined segmentation of the project effort ensures system modularity. Each task forms a separate, distinct program module. At implementation time each module and its inputs and outputs are well-defined, there is no confusion in the intended interface with other system modules. At checkout time the integrity of the module is tested independently; there are few scheduling problems in synchronizing the completion of several tasks before checkout can begin. Finally, the system is maintained in modular fashion; system errors and deficiencies can be traced to specific system modules, thus limiting the scope of detailed error searching.

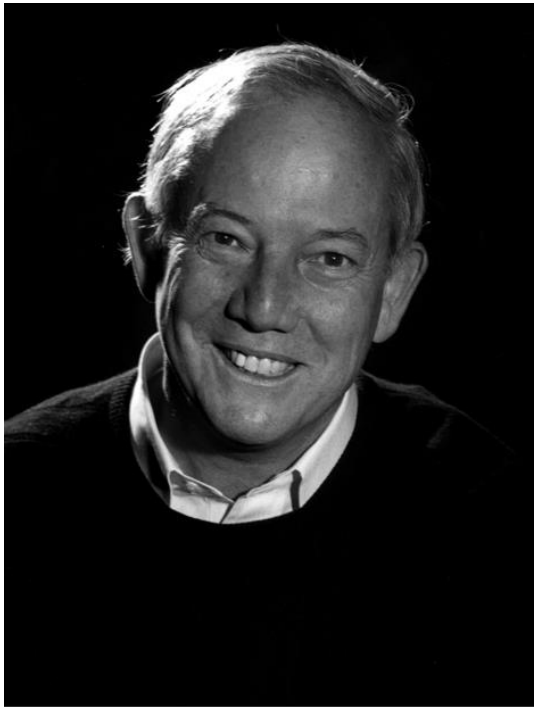
Usually nothing is said about the criteria to be used in dividing the system into modules. This paper will discuss that issue and, by means of examples, suggest some criteria which can be used in decomposing a

- Herbert Simon, [The Architecture of Complexity](), 1962
- Edsger Dijkstra, [Structure of the “THE”-multiprogramming language](), 1968
- David Parnas, [On the Criteria To Be Used in Decomposing Systems into Modules](), 1972

Modularity

"Modularity … is to a technological economy what the division of labor is to a manufacturing one."

- W. Brian Arthur, The Nature of Technology



Aligned Modularization with Domains

"The heart of software is its ability to solve domain-related problems for its user."

- Eric Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software

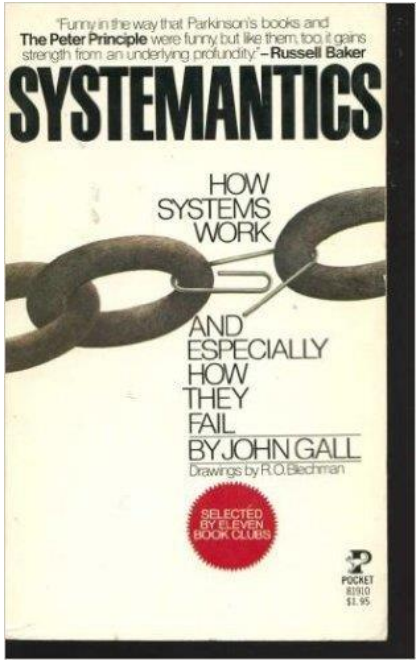
Dealing with Complexity

- Lesson #1 – Differentiate the complexity
- **Lesson #2 – Modularize the system**

Systems Thinking Perspective

"A complex system that works is invariably found to have evolved from a simple system that worked. A complex system designed from scratch never works and cannot be patched up to make it work. You have to start over with a working simple system."

- John Gall, [Systemantics: How Systems Really Work and How They Fail]



Agile Manifesto Principles

“Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.”

“Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.”

http://agilemanifesto.org/principles.html

Dealing with Complexity

- Lesson #1 – Differentiate the complexity
- Lesson #2 – Modularize the system through abstractions
- **Lesson #3 – Start small and iterate**

System Control

“We can't impose our will on a system. We can listen to what the system tells us, and discover how its properties and our values can work together to bring forth something much better than could ever be produced by our will alone.”

– Donella H. Meadows, Thinking in Systems: A Primer

System Control

"REST emphasizes evolvability to sustain an uncontrollable system. If you think you have control over the system or aren't interested in evolvability, don't waste your time arguing about REST."

– Roy Fielding, REST in AEM

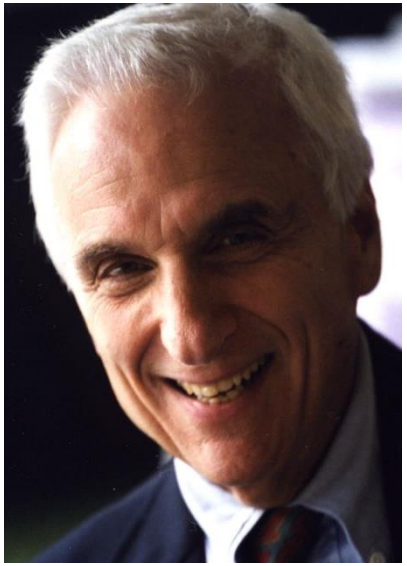
Dealing with Complexity

- Lesson #1 – Differentiate the complexity
- Lesson #2 – Modularize the system through abstractions
- Lesson #3 – Start small and iterate
- **Lesson #4 – Influence—don’t control—the system**

The Software-People System

"Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure."

- Mel Conway, How Do Committees Invent?



Beyond “The System”

“The biggest cause of failure in software-intensive systems is not technical failure; it’s building the wrong thing.”

“Almost everything we know about good software architecture has to do with making software easy to change.”

- Mary Poppendieck, Creator of Lean Software Development

Dealing with Complexity

- Lesson #1 – Differentiate the complexity
- Lesson #2 – Modularize the system through abstractions
- Lesson #3 – Start small and iterate
- Lesson #4 – Influence—don’t’ control—the system
- **Lesson #5 – The system is more than “The System”**

Software Engineering Movements

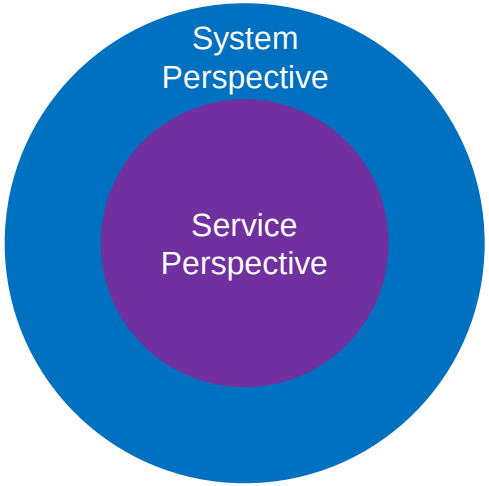
- Object-Oriented Programming (+ UML, Design Patterns)
- Service-Oriented Architecture
- Domain-Driven Design
- Agile Software Development
- DevOps
- Microservice Architecture

They're all trying to address complexity!

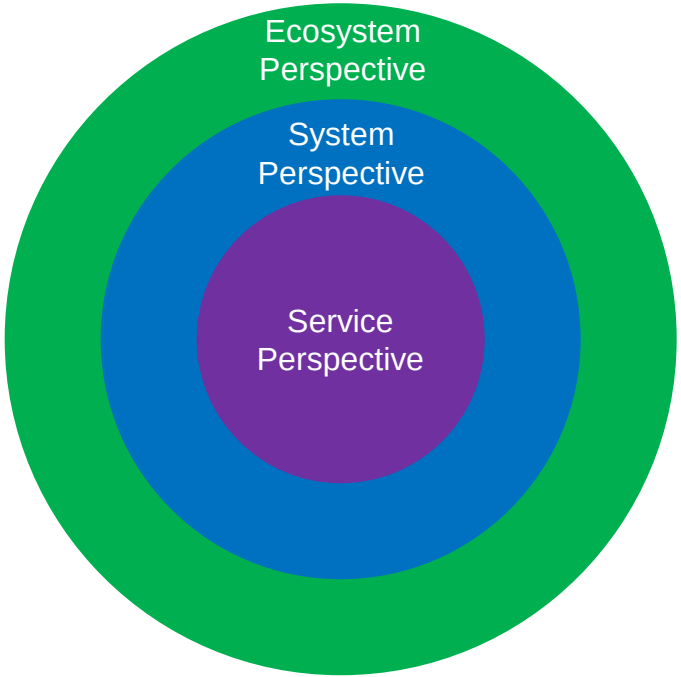
A New Approach

Microservices Design Thinking

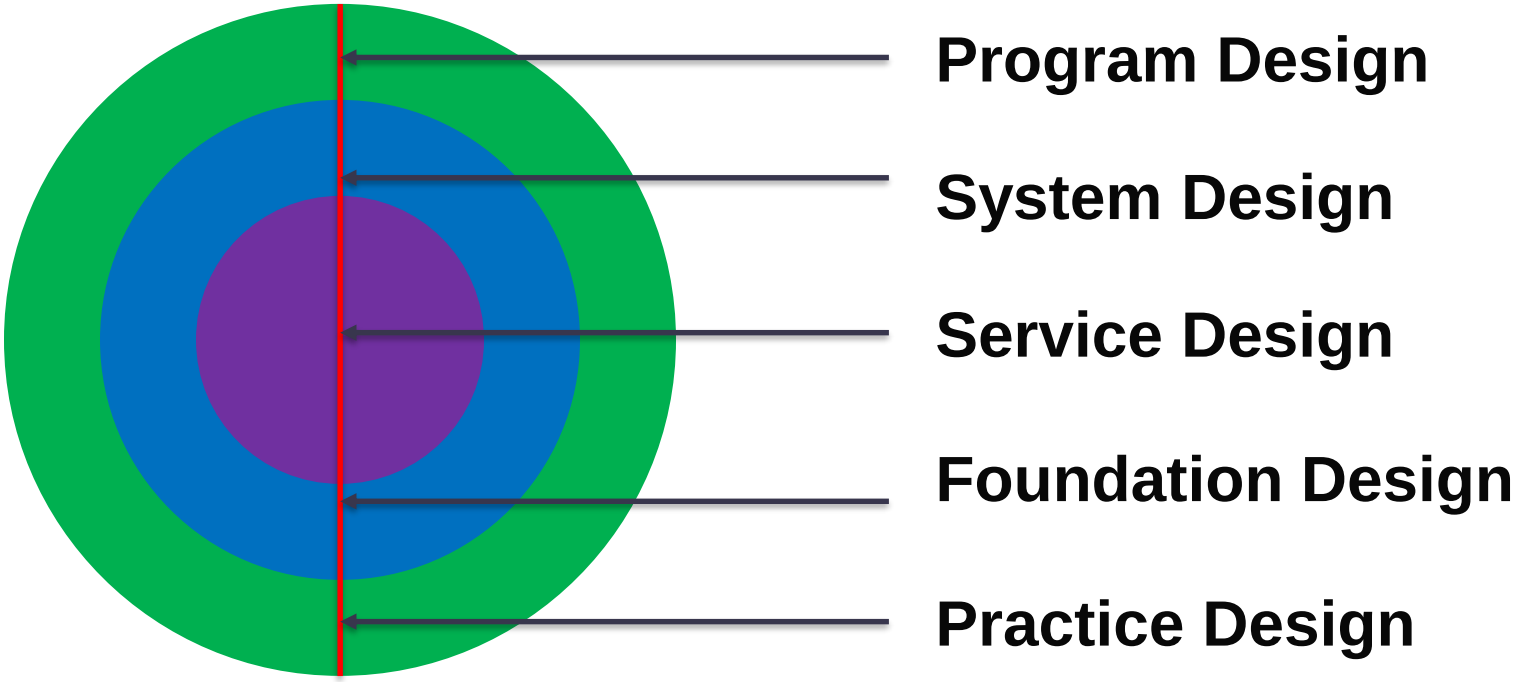
Microservices Design Thinking



Microservices Design Thinking



Microservices Design Thinking



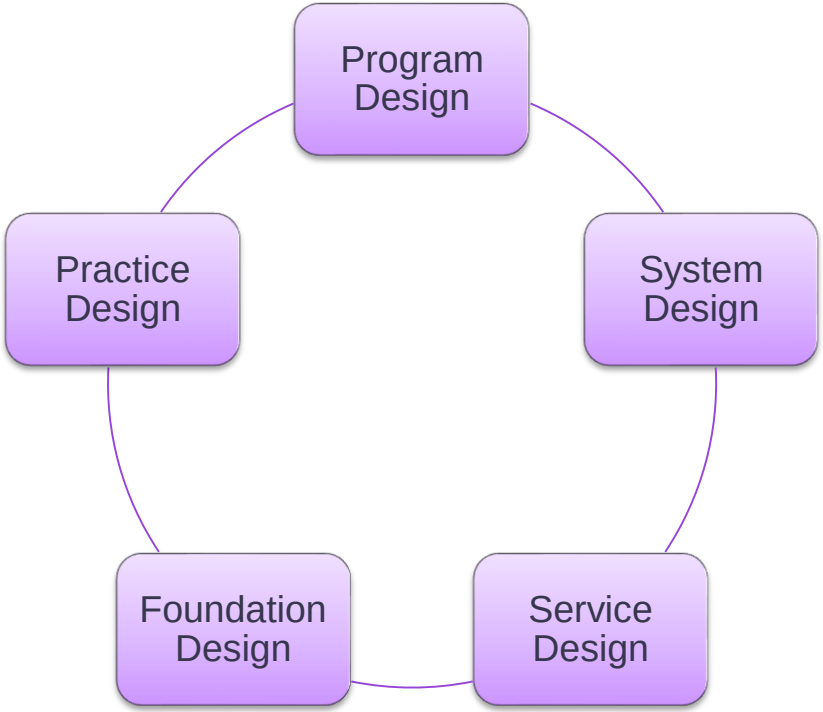
Microservice-based Enterprise Transformation Approach (META)

- A **comprehensive** approach to changing the way an **enterprise** builds and maintains **distributed systems**
- Consists of loosely-coupled **design disciplines** with bi-directional inputs and outputs
- **Organic**, not sequential
- Based on lessons in dealing with **complexity**
- **Synthesizes** elements from other SE approaches

META Goals

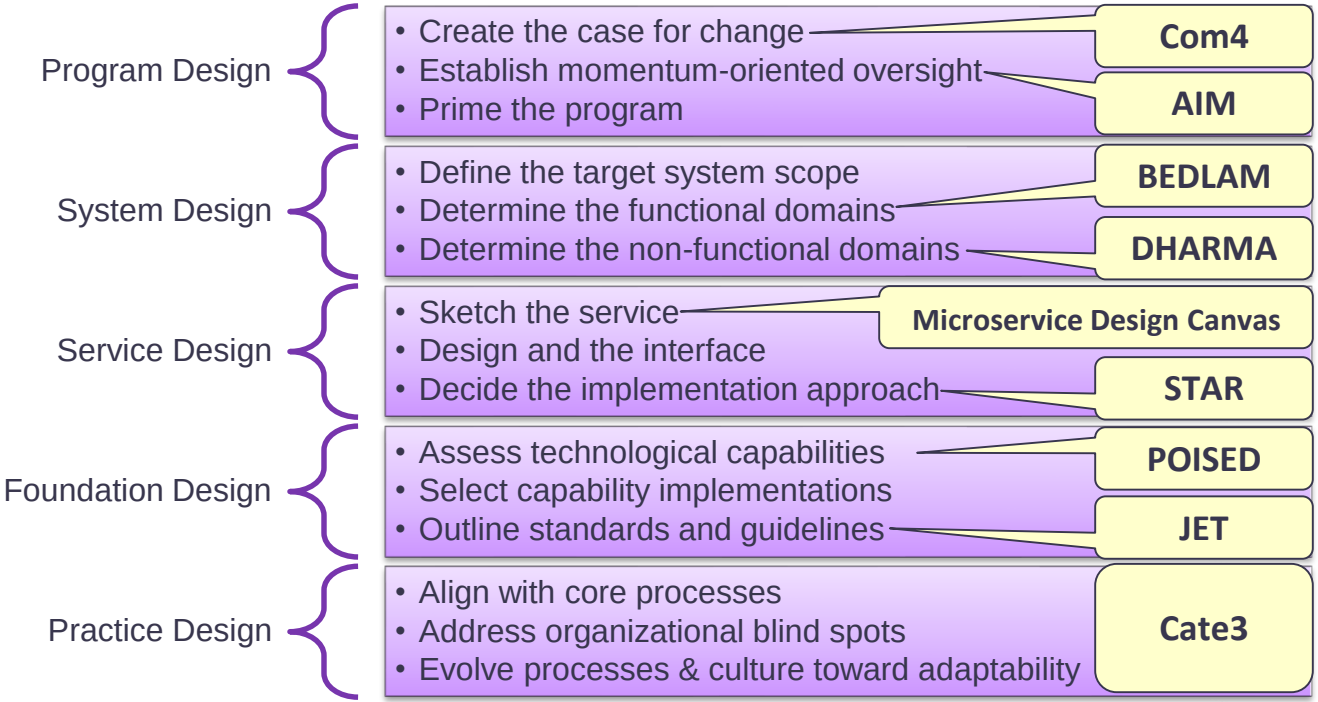
- **Incent** thinking about the right things at the right time
- Address **blind spots**
- Make it practical, usable, streamlined, **memorable**
- **Create** an implementation-agnostic view of the system
- **Test** the designs
- Get things moving and **get out of the way!**

META Design Disciplines



META at a Glance

5 Design Disciplines, 15 Processes, 9 Memes



Background – Islay Bank

- Founded in 1853 by association of whisky distilleries (Bowmore, Scotland)
- $15B+ in revenue, $500B+ in assets, 50K+ employees
- Full spectrum of services: retail banking, merchant banking, business banking, investment banking, wealth management
- IT org has high level split between development and operations
  - Much of Ops is outsourced to third party service provider
  - Development is distributed among business units
  - Architecture, Security and Procurement are centralized
- New “Digital Banking” business unit has been formed
  - But still struggling with pace of change in the market
  - CIO wants to “do microservices” in order to stay competitive

Photo credit: http://jackmaryetc.com/Travel/Europe/UK/Images/Wales/B07BowDis.jpg

What Comes Next…

- Explore META design disciplines in detail
- Observe how the fictional organization (Islay Bank) utilizes the META approach
- Apply META to your own change initiative through activities and exercises

Program Design

Islay Bank Program Context

- See big opportunities in digital banking, but unsure where to start
- Fear disruption from Fintech startups and other digital native players
- Find it very hard to change core services running on legacy tech
- Sense much of their tech stack is antiquated, skills scarce
- Feel current change management processes overly restrictive
- Have had some successful prototypes, but lacked business impact
- Hope that APIs and microservices provide a path forward
- Strongest desire to change is in Retail Banking business unit

System Design

System Design

- Define the target **system scope**
- Determine **functional domains**
  - Bounded contexts, services, and interactions
- Determine **non-functional** domains
  - Trust domains (security), operational domains (availability, reliability, capacity)

Complexity in a Microservice Architecture

Essential Complexity in Microservices

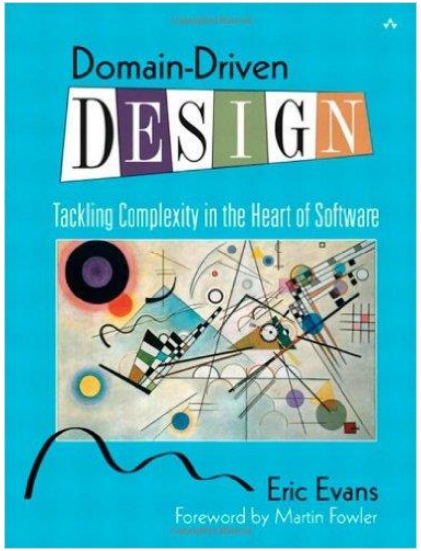
- In a microservice architecture, the topology of the implemented system closely resembles the model of the system's "essence"

Accidental Complexity in Microservices

- In a microservice architecture, accidental complexity can be minimized through automation and distribution

Dealing with Essential Complexity in Microservices

- Eric Evans' Domain-Driven Design provides a framework for defining and modeling the essential capabilities of complex software systems



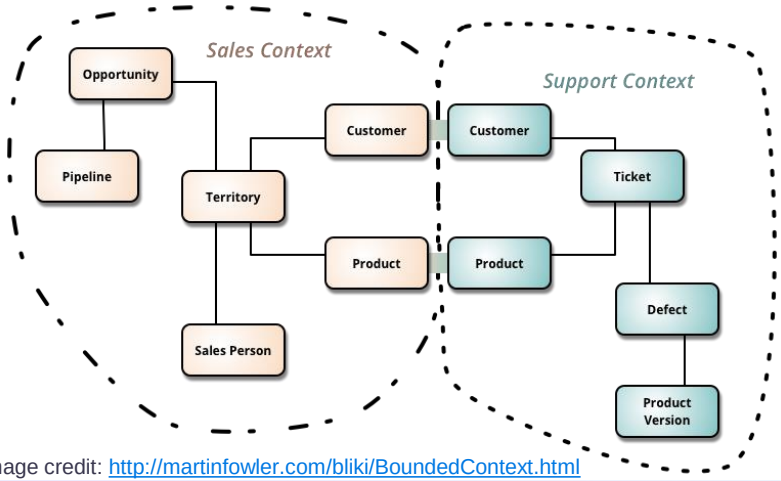


Image credit: http://martinfowler.com/bliki/BoundedContext.html

On Domain-Driven Design and Microservices

- Originally devised for OOP, but frequently cited for microservices
- Important Concepts
  - Domains and Subdomains
  - Bounded Contexts and Context Maps
  - Anti-Corruption Layer
- **Tip**: Use DDD concepts to model the essential complexity of the software system
- **Pitfalls**: data modeling concepts are too deep for the essence of microservice architecture

Event Storming

http://eventstorming.com/ from Alberto Brandolini

On Industry Service Taxonomies

- There are numerous standards that attempt to model specific industries
- Examples
  - Telecommunications:
    - OneAPI (http://www.gsma.com/oneapi)
  - Financial Services:
    - BIAN (https://bian.org/)
    - Open Banking Standards (https://theodi.org/open-banking-standard)
  - Healthcare:
    - FHIR (https://www.hl7.org/fhir/)

- Use these standards as a measuring stick against your own domain models and service boundaries
- DO NOT adopt them as part of your implementation or you will build an external dependency that will inhibit agility and lead to unavoidable breaking changes down the road

Define the Target System Scope

System Design

- Define system scope and enumerate its domains through business classification
  - Scope can be based on org units or could be as simple as a single monolithic application
- For higher level domains, assess microservices fit for domains
  - Which domains have greenfield initiatives?
  - Which domains have the highest change frequency?
  - On which domains do other domains depend the most?
  - Which domains display microservice characteristics such as APIs, containers, Agile methods, DevOps culture, continuous delivery practices and pipelines?
- Prioritize domains for microservice adoption
  - Start small, iterate, build momentum

Define Target System Scope

System Design

- IsB chose to focus on Retail Banking, which includes the following 6 domains:
  - Assisted Service Banking, Self-Service Banking, Customer and Card Management, Deposit Accounts, Retail Lending and Credit, Mortgages
- They assessed the microservices fit for these domains:
  - Which domains have greenfield initiatives? **Self-Service Banking ("SingleMalt" mobile payments initiative), Retail Lending and Credit**
  - Which domains have the highest change frequency? **Self-Service Banking, Customer and Card Management**
  - On which domains do other domains depend the most? **Customer and Card Management**
  - Which domains display microservice characteristics such as APIs, containers, Agile methods, DevOps culture, continuous delivery practices and pipelines? **Self-Service Banking (Many APIs for Mobile, experimenting with Docker-based services), Mortgages (APIs, DevOps)**
- They prioritized the Self-Service Banking and Customer and Card Management domains for initial microservice adoption
  - Started with a marketing-driven mobile initiative
  - Next tackled the "SingleMalt" mobile payments initiative

Domain Background

System Design – Define Target System Scope

Self-Service Banking

- Primarily grew up around online banking
- Built a web-based monolith to support all online banking functions
- Added mobile banking, but was challenging based on monolithic architecture that had the UI and business logic entangled
- Have added some Docker-based APIs that are shared across web and mobile
- Generally, do not own the systems of record for products they service

Customer and Card Management

- Legacy customer information system (CIS) is mainframe-based
- Recently concluded massive SOA-inspired migration of all customer data into CIS
- Responsible for shared services in Retail, including ESB Integration
- ATM and POS transaction processing also in this org unit
- Perceived as slow and expensive by other org units

The “SingleMalt” Initiative

- Mobile is transforming the payments landscape
- Customers want to be recognized for their full portfolio of holdings with IsB, not just account by account
- “SingleMalt” is a customer-centric, dynamic payments initiative that removes the “single account authorization” restriction
- Payments are authorized in real time utilizing a number of data points and customer preferences
- Payments may be posted immediately or later based on customer preferences and situational awareness

“SingleMalt” the old way

- The initiative would be planned out as a waterfall, big bang program
- Responsibility for the initiative would be sub-contracted to an SI
- Work packages would be portioned out to LOB’s, primed by contractors
- Domain experts would used as consultants
- Monolithic system would be built, aligned with the initiative
- Integration would follow the path of least immediate resistance
- Politics, politics, politics!

Thought Exercise: Define Target System Scope

Time: 10 minutes

- List your top goals for moving to a microservice architecture
- Enumerate the areas of your organization where this change will be most expedient or most valuable
- Within those domains, define the scope of the system you are going to address first
  - Existing monolithic application?
  - Greenfield initiative?
  - Integrated legacy applications?

Determine Functional Domains

System Design

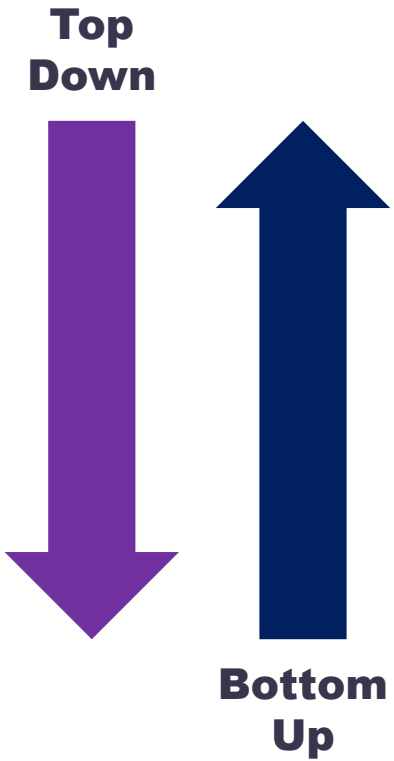
- “Functional” implies business-aligned
- Use “jobs-to-be-done” process to define system scope, sketch service boundaries and outline interface semantics
- Considerations for determining domain and service boundaries:
  - What are the high level business functions in the system?
  - What are the functional areas that always change together?
  - What/who will cause/request changes to the system over time?
  - What functional areas share a common vocabulary?
  - What functions are consumed by multiple components within the system?
  - How many people per domain or service?

NOTE: The “system designer” should focus on service boundaries but not go too deep on the services themselves

BEDLAM for Modeling Service Boundaries

Bidirectional Event- and Domain-based Lightweight Application Modeling

- Events
- Interactions
- Domains
- Bounded Contexts
- Services
- Service Consumers



BEDLAM for Modeling Service Boundaries

Bidirectional Event- and Domain-based Lightweight Application Modeling

Top Down

- Draft an initial set of domains within the system
  - Multiple options for initial classification:
    - **By Process** – Business process taxonomy
    - **By Org Unit** – Associated organizational structure
    - **By Modularization** – For refactoring a monolith
  - Other helpful resources from OpenCredo and InnoQ
- Test domain boundaries using other classification schemes
  - Also think about “linguistic/semantic boundaries”
- Define *bounded contexts* based on synthesis of classifications

BEDLAM for Modeling Service Boundaries

Bidirectional Event- and Domain-based Lightweight Application Modeling

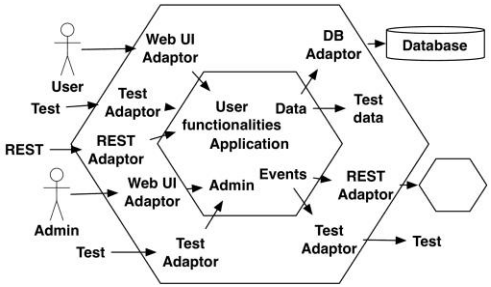
Bottom Up

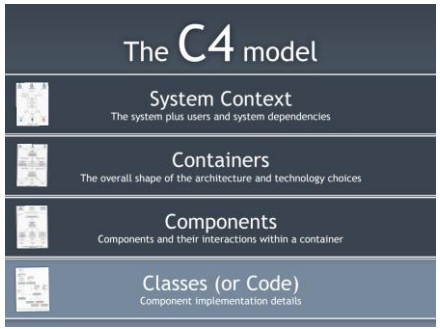
- List the events in the system
  - Event Storming approach helpful
- Enumerate the potential interactions within the system for each event
  - Think through all interaction types:
    - **Queries** – Read only synchronous requests – “Can you please tell me…?”
    - **Commands** – State changing synchronous requests – “Can you please do…?”
    - **Events** – Post-event asynchronous messages – “This/that happened”
- Overlay the domain interactions for all events
  - Identify common interaction and resource combinations
- Sketch service boundaries and vocabularies

BEDLAM for Modeling Service Boundaries

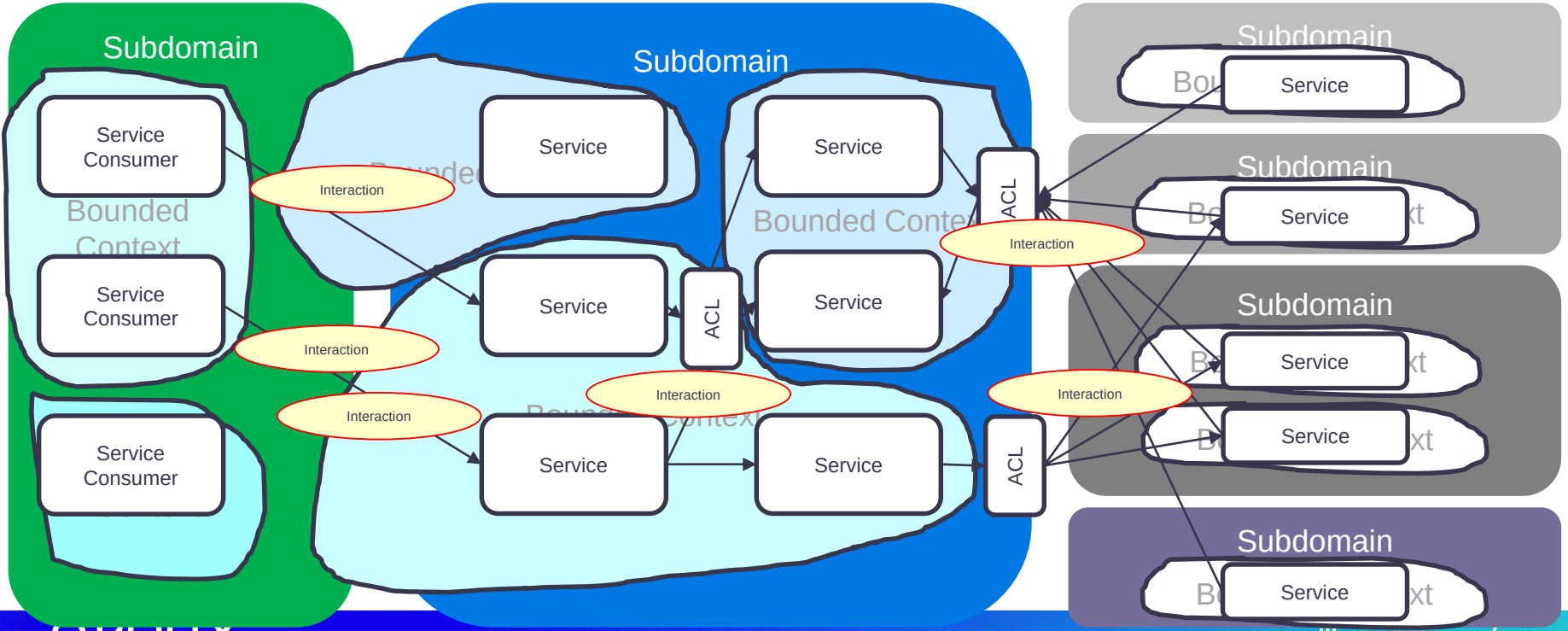
Bidirectional Event- and Domain-based Lightweight Application Modeling

- Visualization helps!
  - The following example uses a derivative of DDD context mapping
  - Other examples: hexagonal architecture from Alistair Cockburn, C4 Model from Simon Brown





BEDLAM Context Map



“SingleMalt” User Experience

- Customer opts into the SingleMalt program, initiates usage, and sets preferences for accessible accounts and payments posting
- Customer attempts to purchase goods online using SingleMalt
  - Authorization decision made using customer preference-scoped accounts and data sources
  - Authorization and purchase recorded; posting date, posting account(s) and fee (optional) determined dynamically based on customer preferences
- **Example**: I elect to join the SingleMalt flat fee program ($10/month) using my chequing account, personal line of credit and credit card as posting accounts. I also include my mortgage and investment accounts as data source for authorization decisions.

“SingleMalt” with BEDLAM

“SingleMalt” Domains

Self-Service Banking

Customer and Card Management

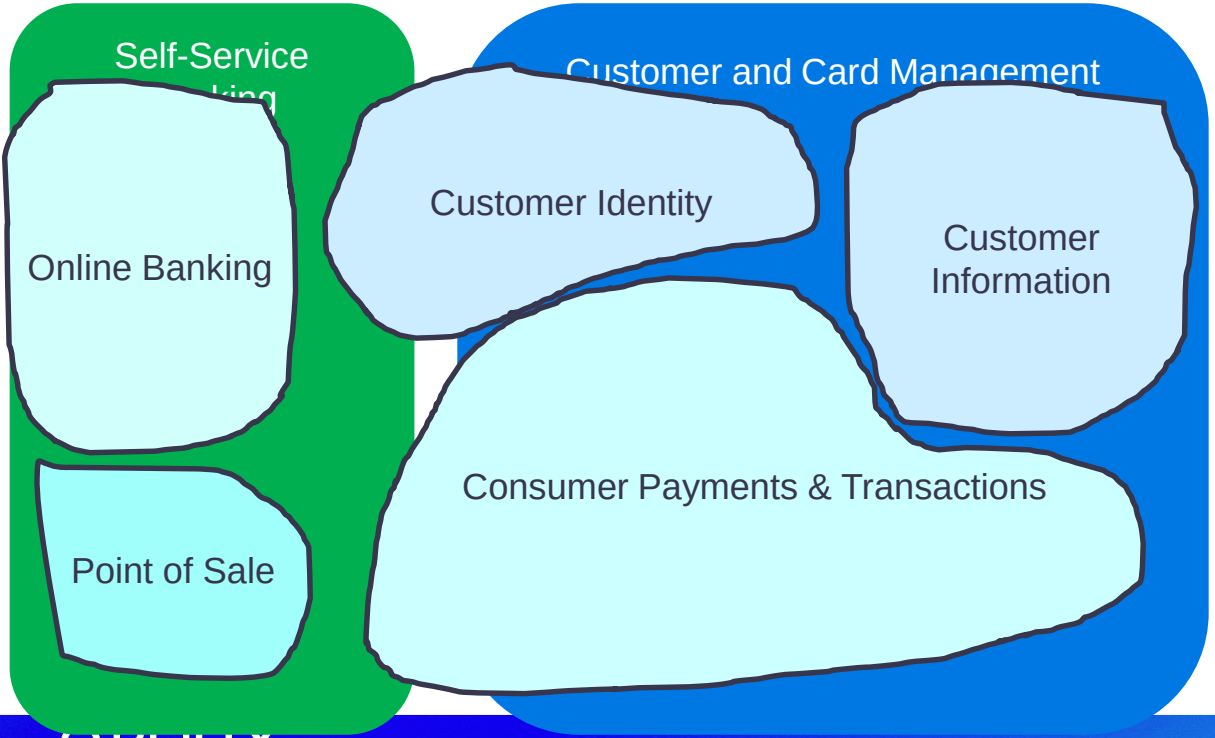
Investment Banking

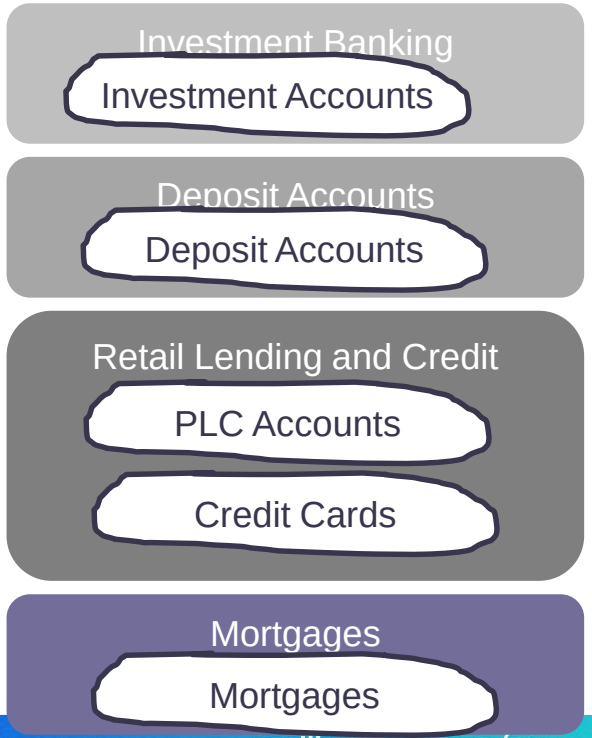
Deposit Accounts

Retail Lending and Credit

Mortgages

“SingleMalt” Bounded Contexts





“SingleMalt” with BEDLAM

Bottom Up

“SingleMalt” Events

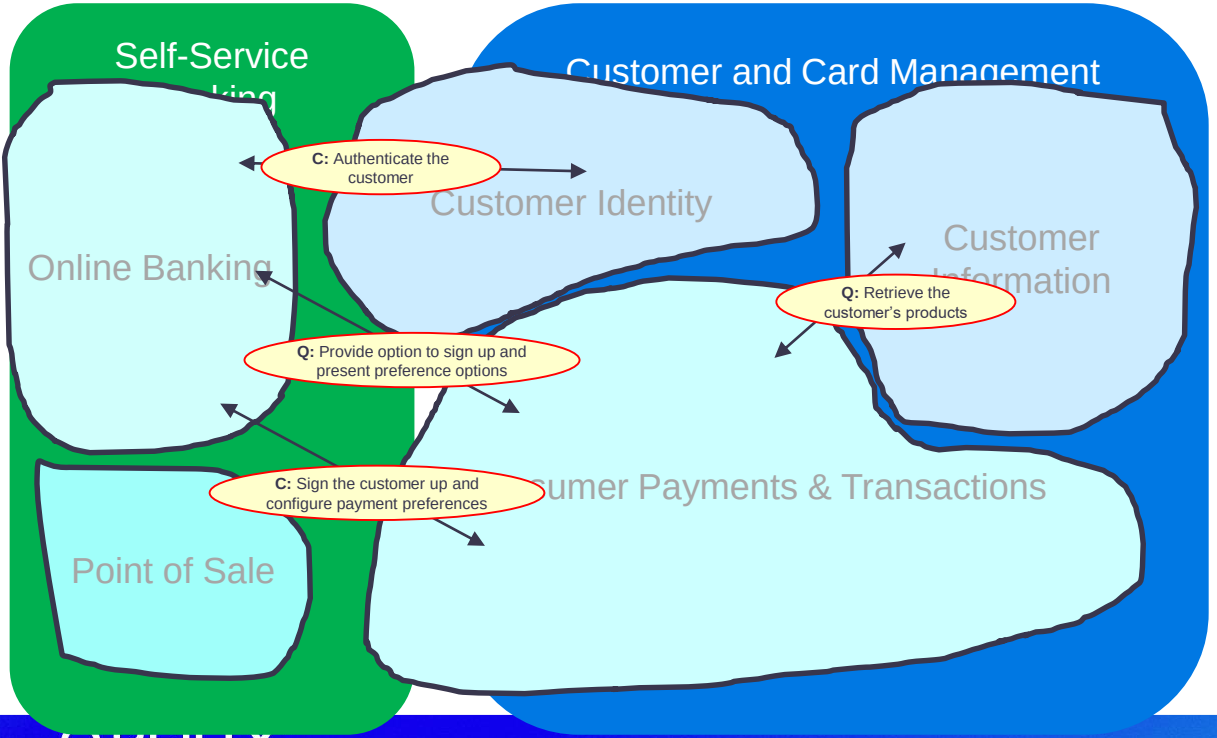
Customer signs up	Customer changes preferences	Customer opts out
Customer requests authorization	Transaction gets fulfilled	Activity on customer product

“SingleMalt” Interactions

Customer signs up

Queries
Commands
Events

IsB



Investment Banking
Investment Accounts

Deposit Accounts
Deposit Accounts

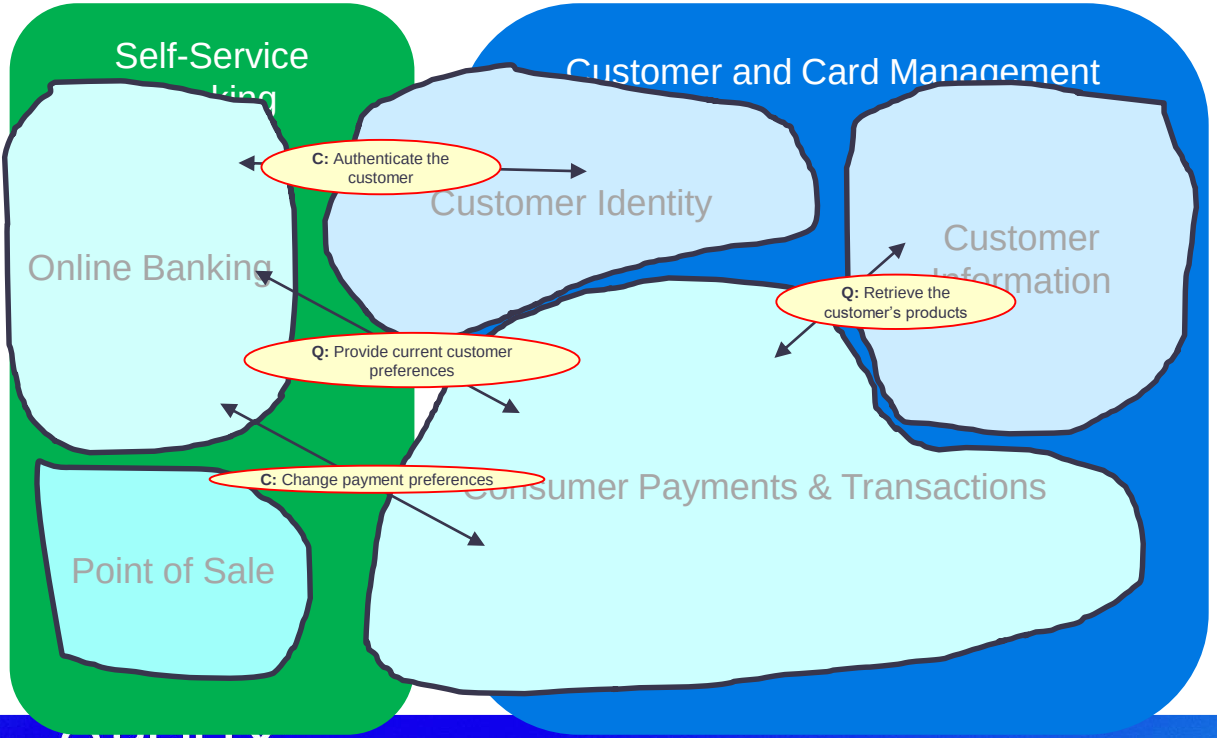
Retail Lending and Credit
PLC Accounts
Credit Cards

Mortgages
Mortgages

“SingleMalt” Interactions

Customer changes preferences

Queries
Commands
Events



IsB

Investment Banking
Investment Accounts

Deposit Accounts
Deposit Accounts

Retail Lending and Credit
PLC Accounts
Credit Cards

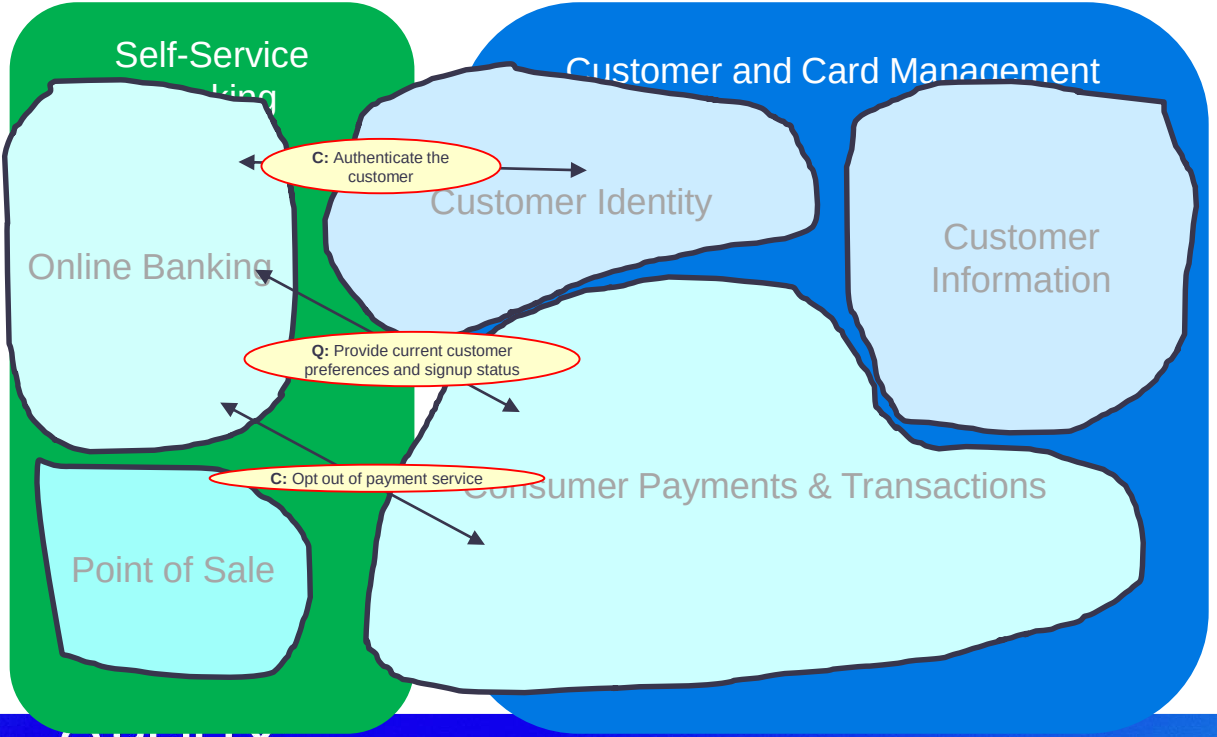
Mortgages
Mortgages

“SingleMalt” Interactions

Customer opts out

Queries
Commands
Events

IsB



Investment Banking
Investment Accounts

Deposit Accounts
Deposit Accounts

Retail Lending and Credit
PLC Accounts
Credit Cards

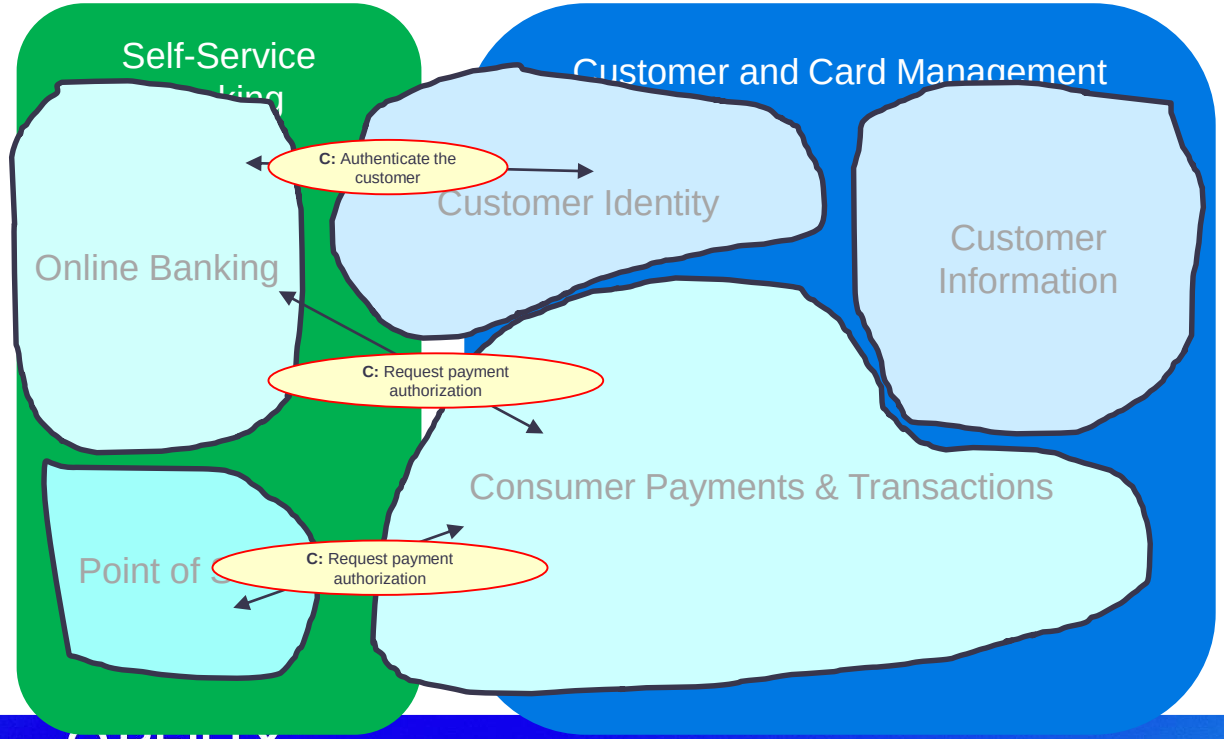
Mortgages
Mortgages

“SingleMalt” Interactions

Customer requests authorization

Queries
Commands
Events

IsB



Investment Banking
Investment Accounts

Deposit Accounts
Deposit Accounts

Retail Lending and Credit
PLC Accounts
Credit Cards

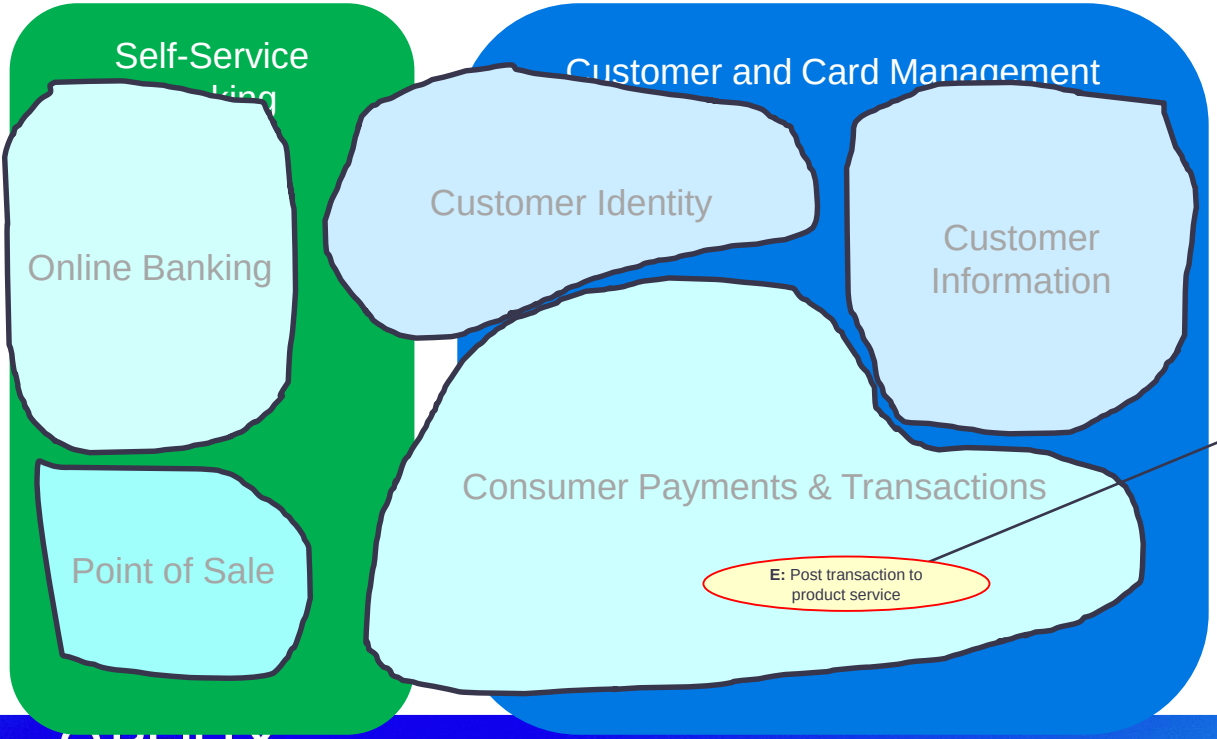
Mortgages
Mortgages

“SingleMalt” Interactions

Transaction gets fulfilled

Queries
Commands
Events

IsB



Investment Banking
Investment Accounts

Deposit Accounts
Deposit Accounts

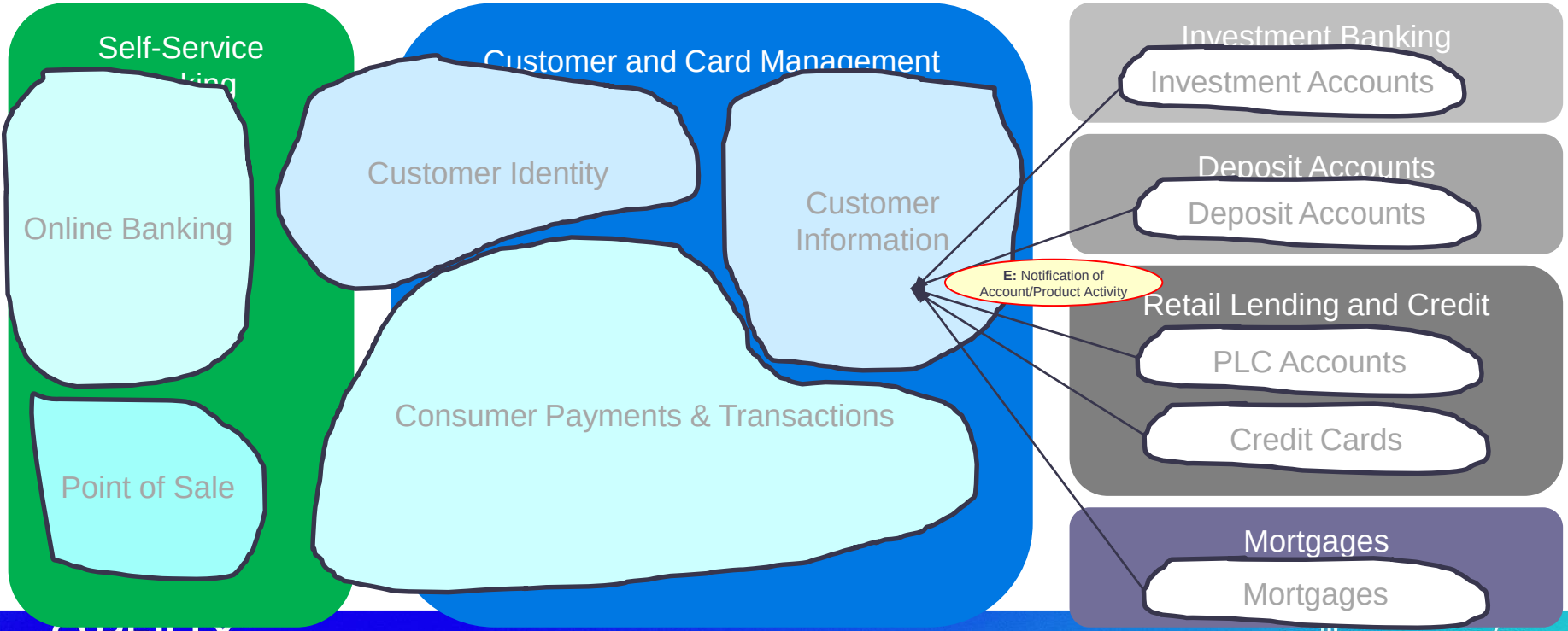
Retail Lending and Credit
PLC Accounts
Credit Cards

Mortgages
Mortgages

“SingleMalt” Interactions

Activity on customer product

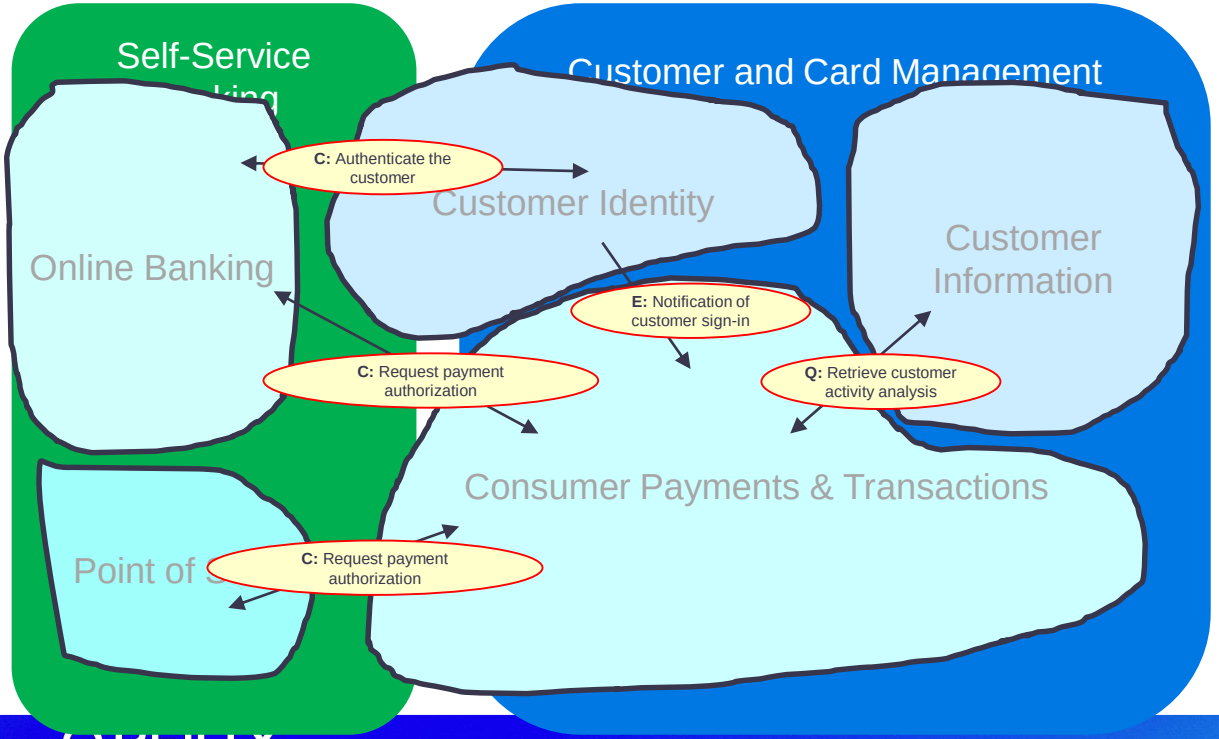
Queries
Commands
Events



“SingleMalt” Interactions

Customer requests authorization

Queries
Commands
Events



IsB

Investment Banking
Investment Accounts

Deposit Accounts
Deposit Accounts

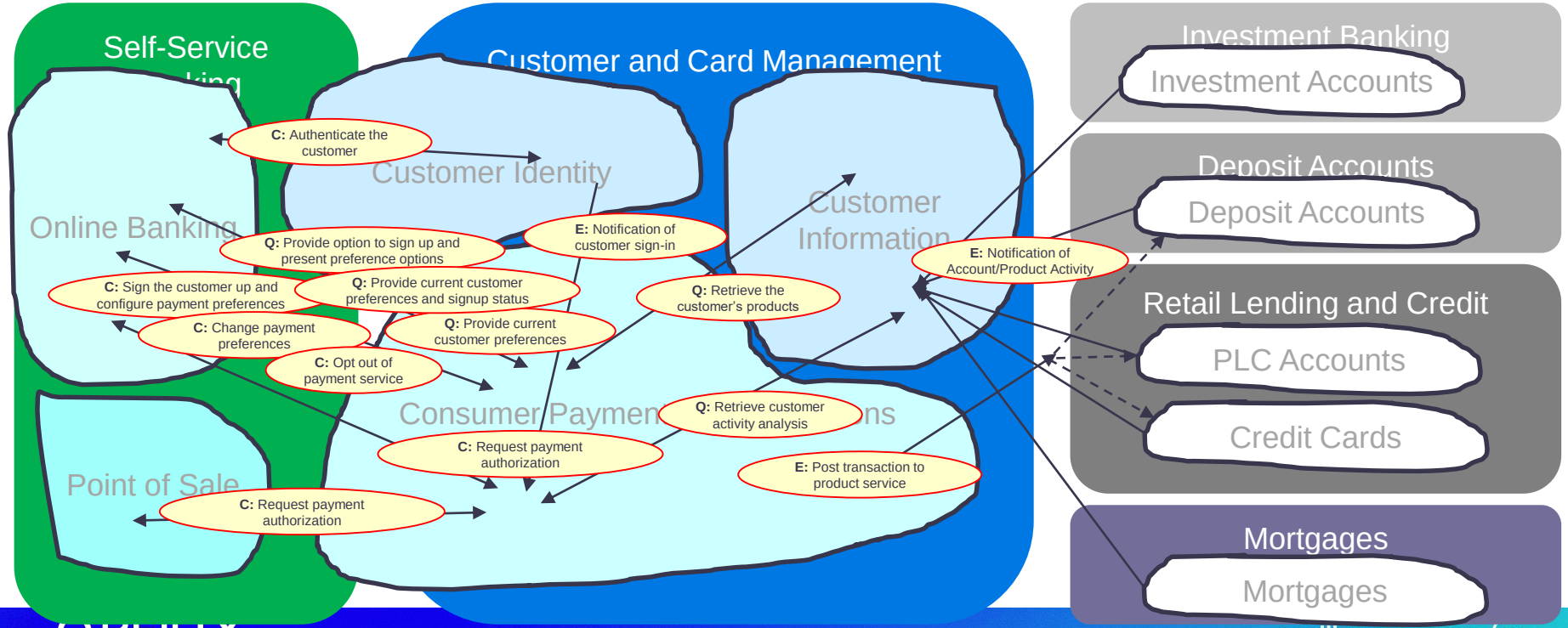
Retail Lending and Credit
PLC Accounts
Credit Cards

Mortgages
Mortgages

“SingleMalt” Interactions
Overlay

Queries
Commands
Events

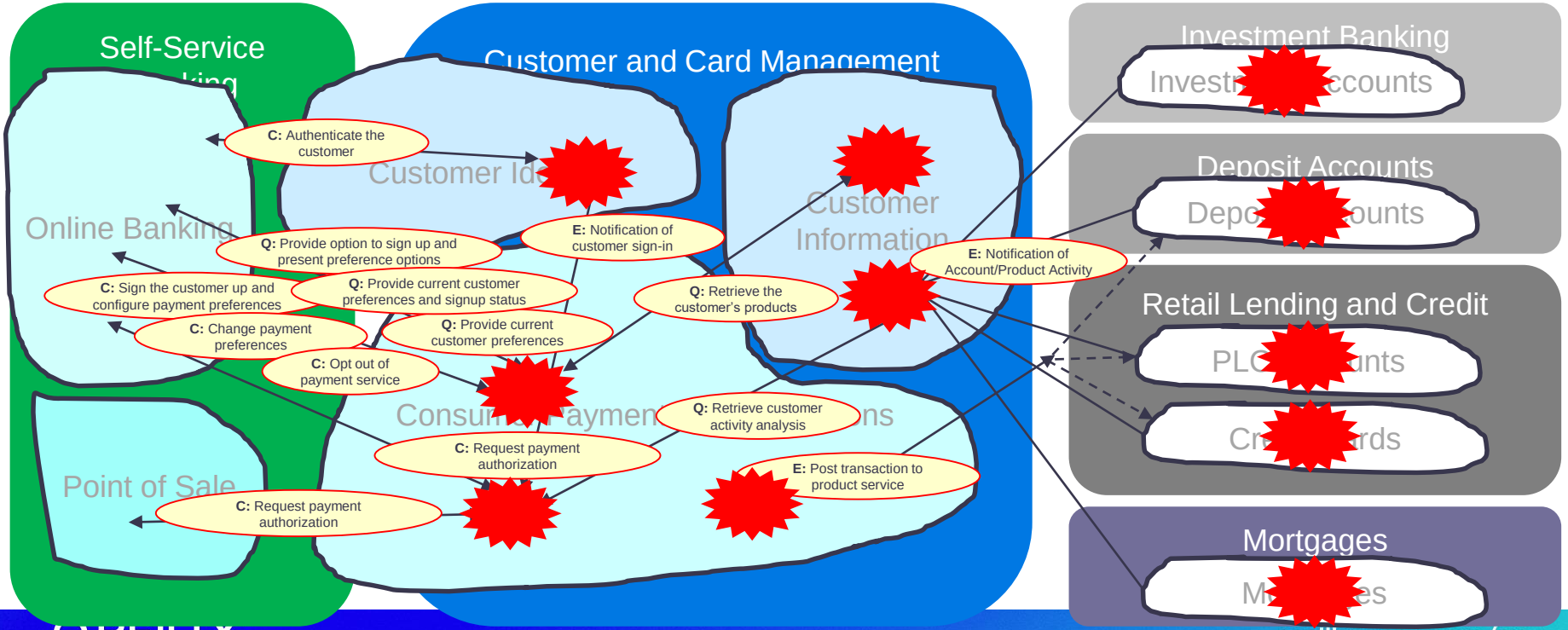
IsB



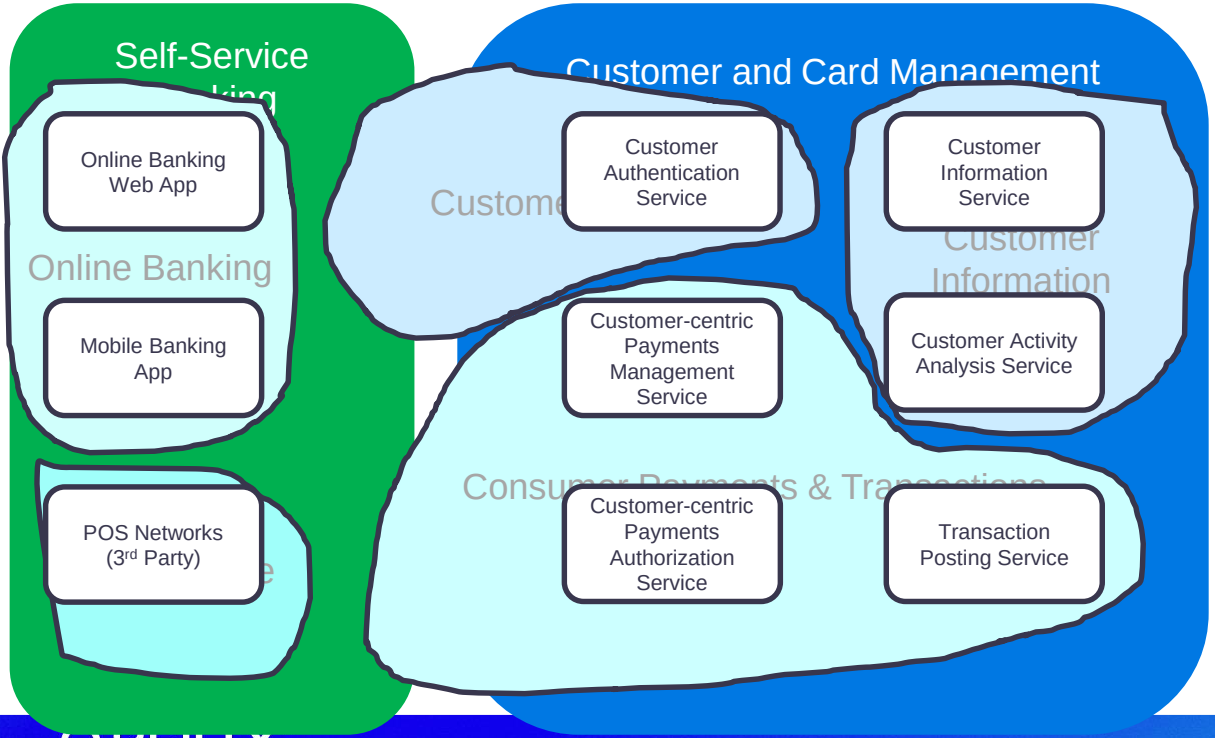
“SingleMalt” Interactions

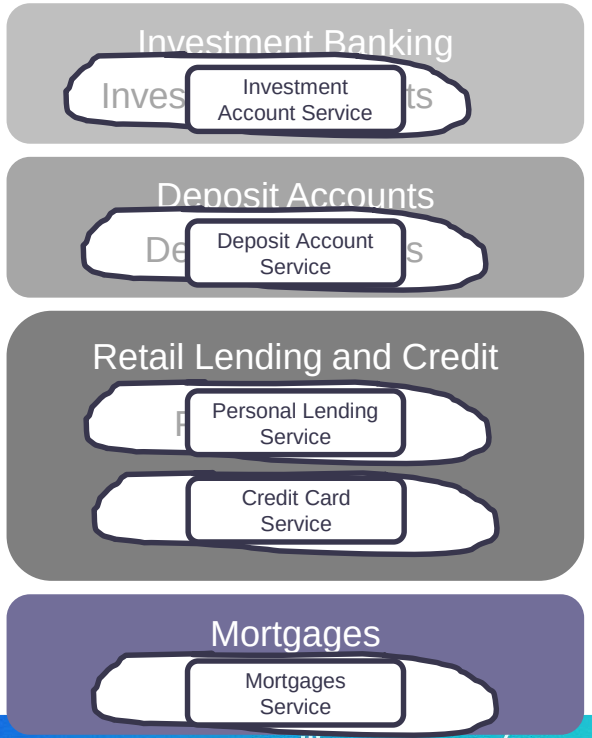
Overlay – Emerging Services

Queries
Commands
Events



“SingleMalt” Services
Context Map





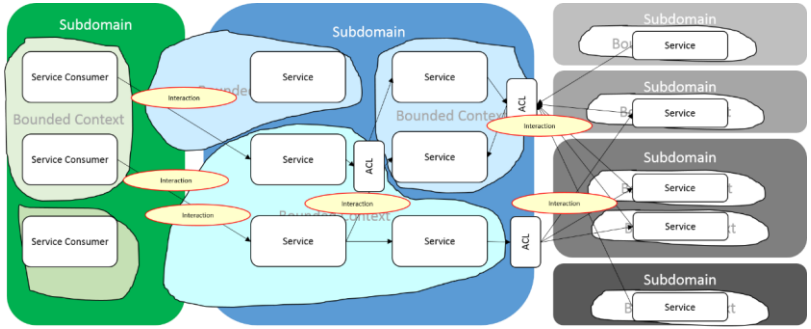
Reality Check

- Retail Banking is a very complex domain!
  - This example was chosen in order to illustrate complex scenarios
  - You may want to start with a smaller scope and iterate in order to learn
- You may not know the big picture…
  - But that won't be important at the beginning
  - Expect to redraw some boundaries
  - Much better to get started and fill in the blanks than to expect to create the ideal service topology before you start implementing

Exercise: Applying BEDLAM

Time: 15 minutes

- The purpose of this exercise is to practice establishing service boundaries through subdomains, bounded contexts and interactions
- Create a context map based on a real life scenario
- Helpful steps
  - Define domains and bounded contexts
  - Brainstorm service consumers
  - List tasks for each service consumer



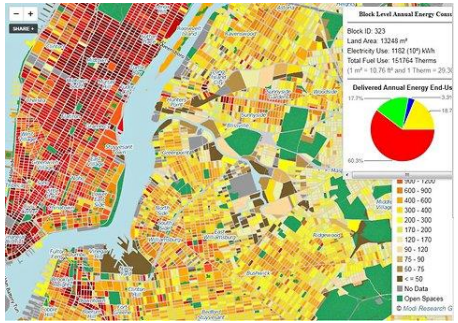
Testing the BEDLAM System Design

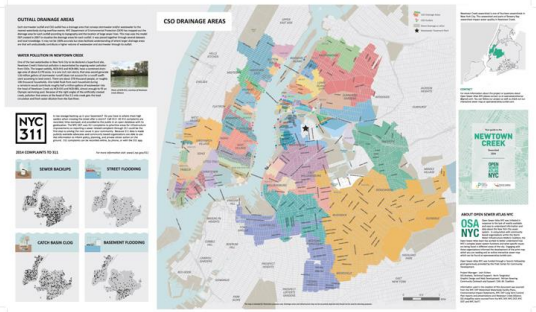
- If the system scope aligns with an existing application…
  - Review the feature backlog
  - See how many services each feature would impact
  - Measure and adjust coordination effort

Planes and Domains

- Functional domains exist on the **data plane**
  - Where components of the system interact to fulfill the core business purpose of the system
  - This is the focus of DDD
- System design must also address the various **control planes**
  - Security, observability, service level (availability, performance, reliability)

The Always Helpful City Planning Analogy





Determine Non-Functional Domains

System Design

- “Non-Functional” areas are mostly the “-ities”
  - Security, Availability, Reliability, Scalability, Operability
- Differences in these areas may impact how you organize the system
- Considerations for determining non-functional domains:
  - Do some services need to be grouped based on security, access and privacy?
  - Is there a difference in service level expectations (availability, hours of operation, etc.) for some services versus others?
  - Are there different tiers of performance and scalability for services?
- Non-functional domains will help determine required capabilities
- Security merits special attention…

Service Design

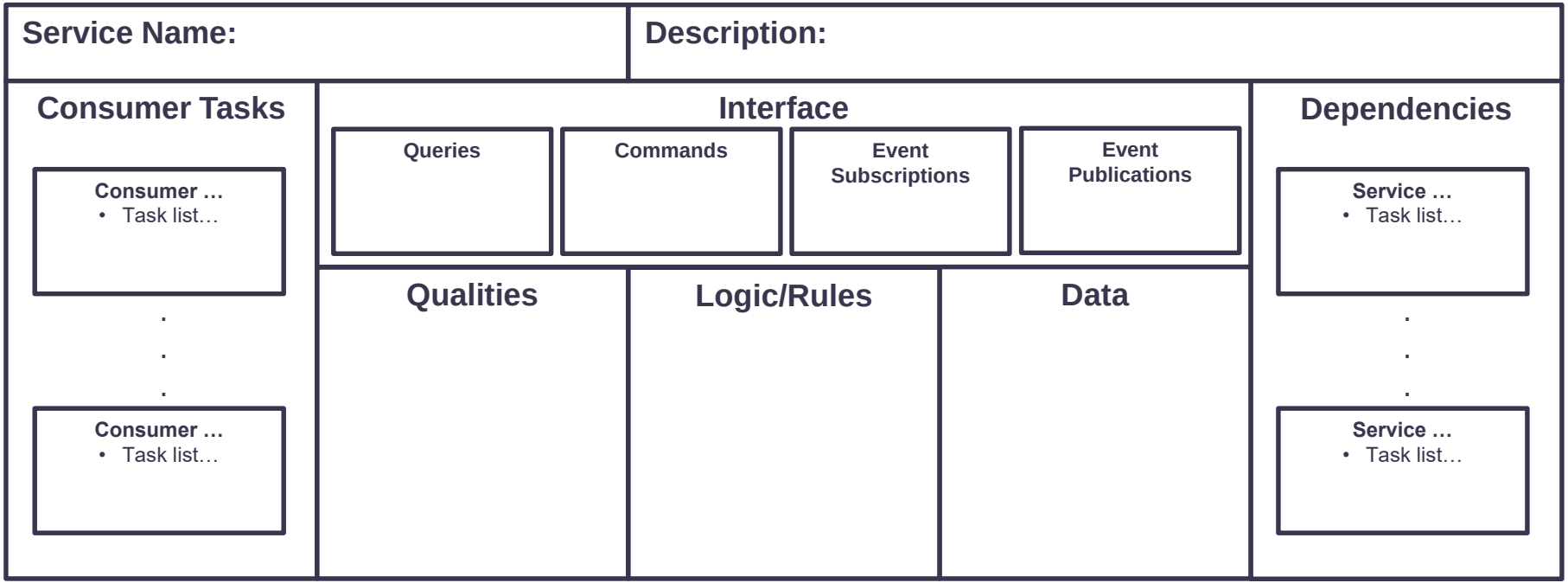
Service Design

- **Sketch** the service
- Design the **interface**
  - API and contract
  - Prototype it
- Determine the **implementation**
  - Find a reusable service
  - Evolve an existing service
  - Develop net new service

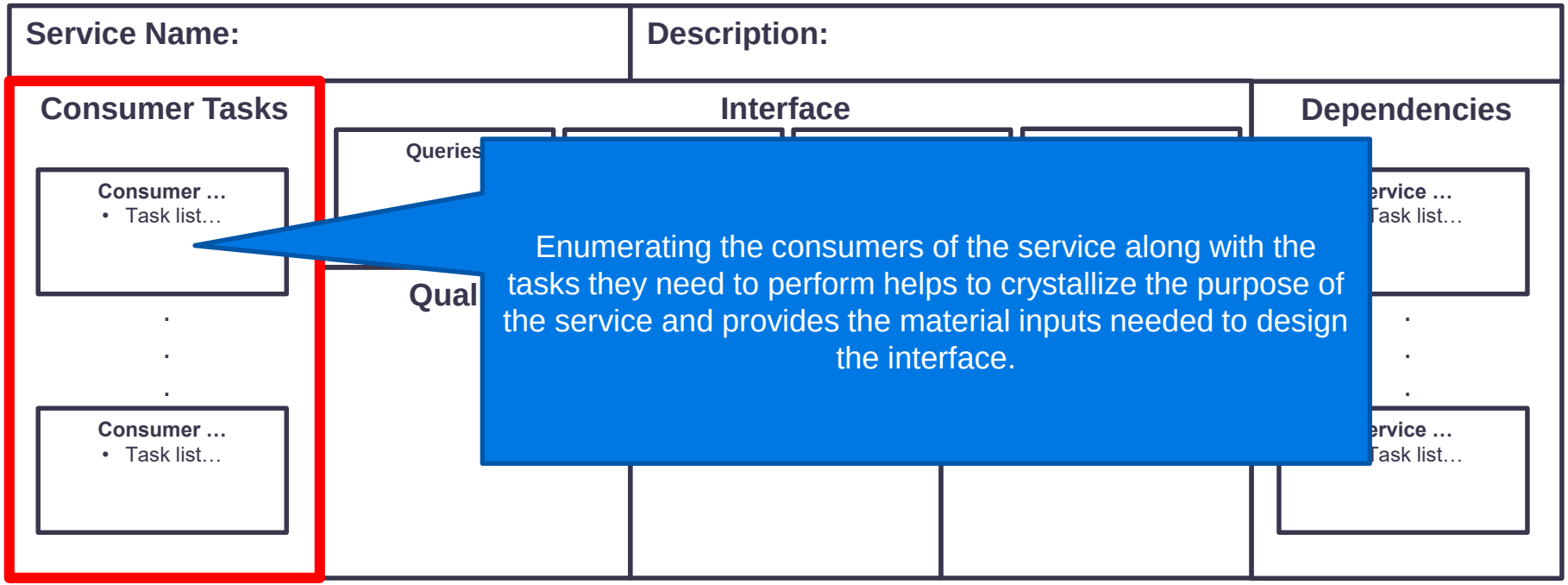
Service Design: Sketch the Service

- Take an “outside in” approach
- Start with external concerns
  - Service consumers and associated tasks
  - Interface/API (vocabulary, interaction patterns, resources, task mapping)
  - Qualities (SLA’s, NFR’s, security policy, versioning approach)
  - External service dependencies
- Let the external concerns drive and hide the service’s internals
  - Core logic, rules and data

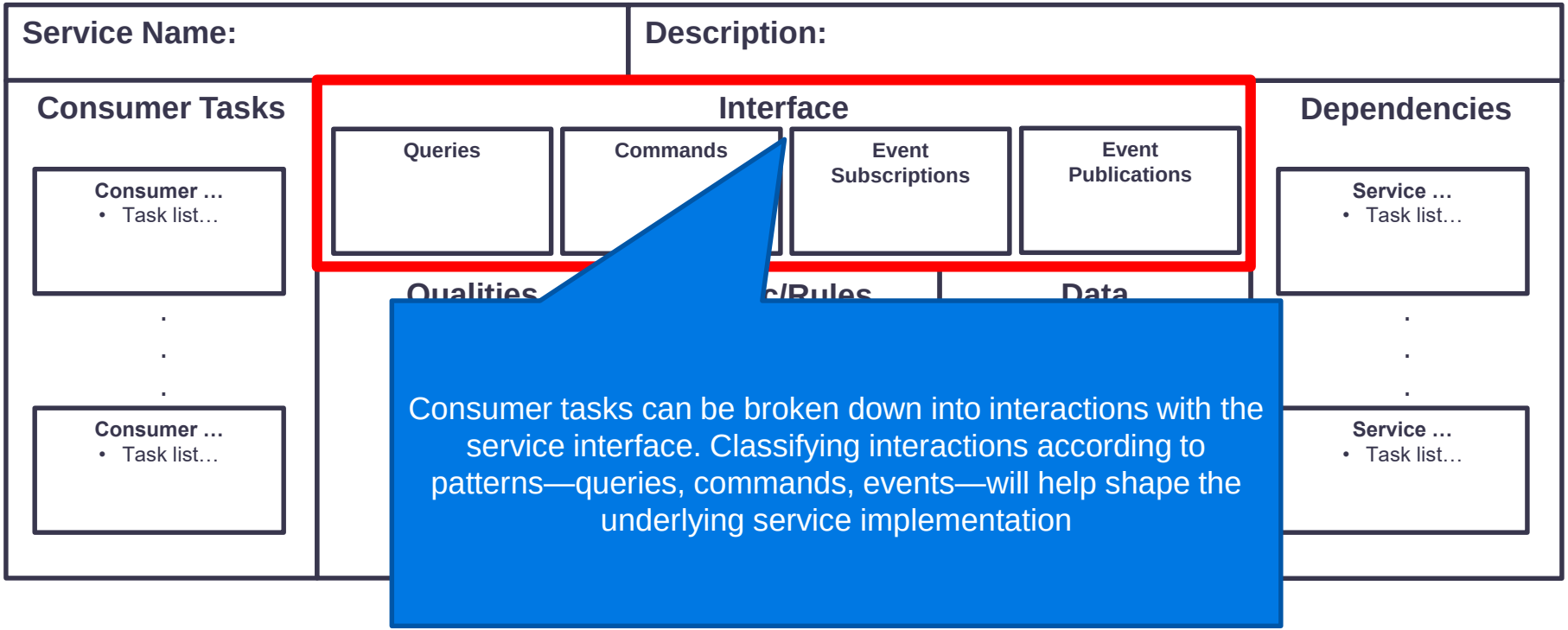
Microservice Design Canvas



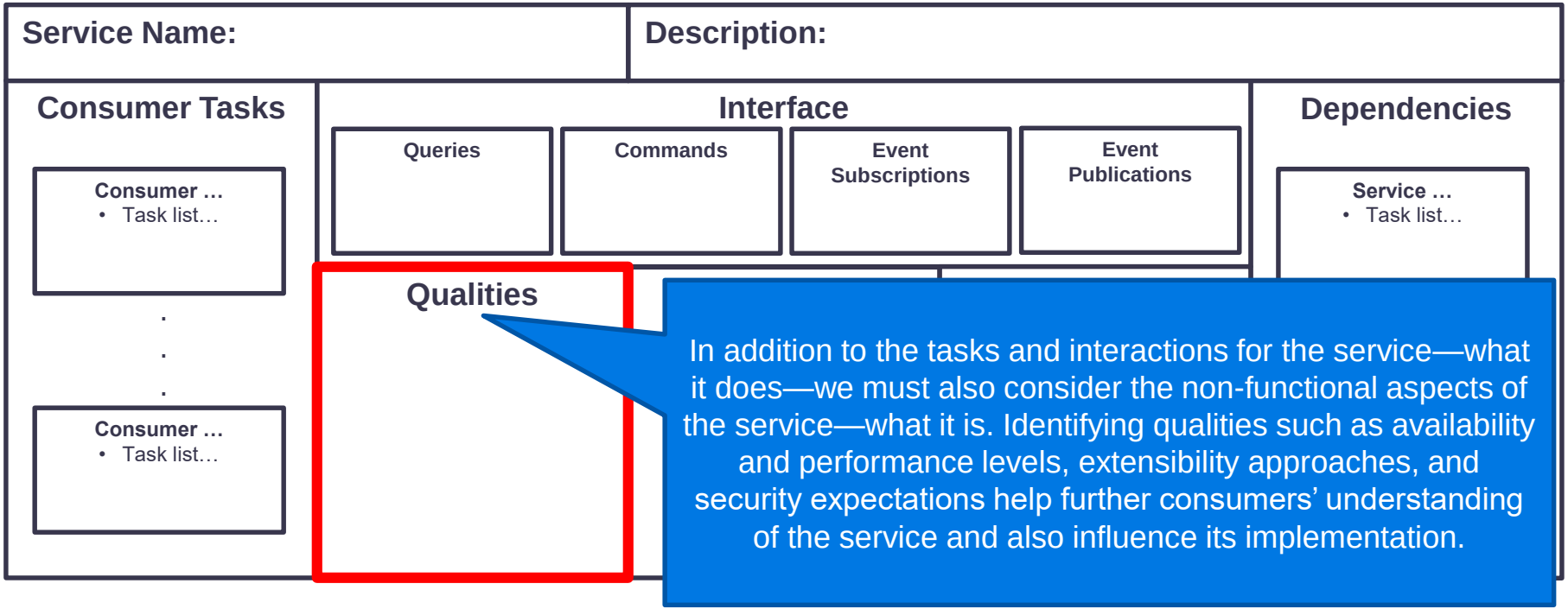
Microservice Design Canvas



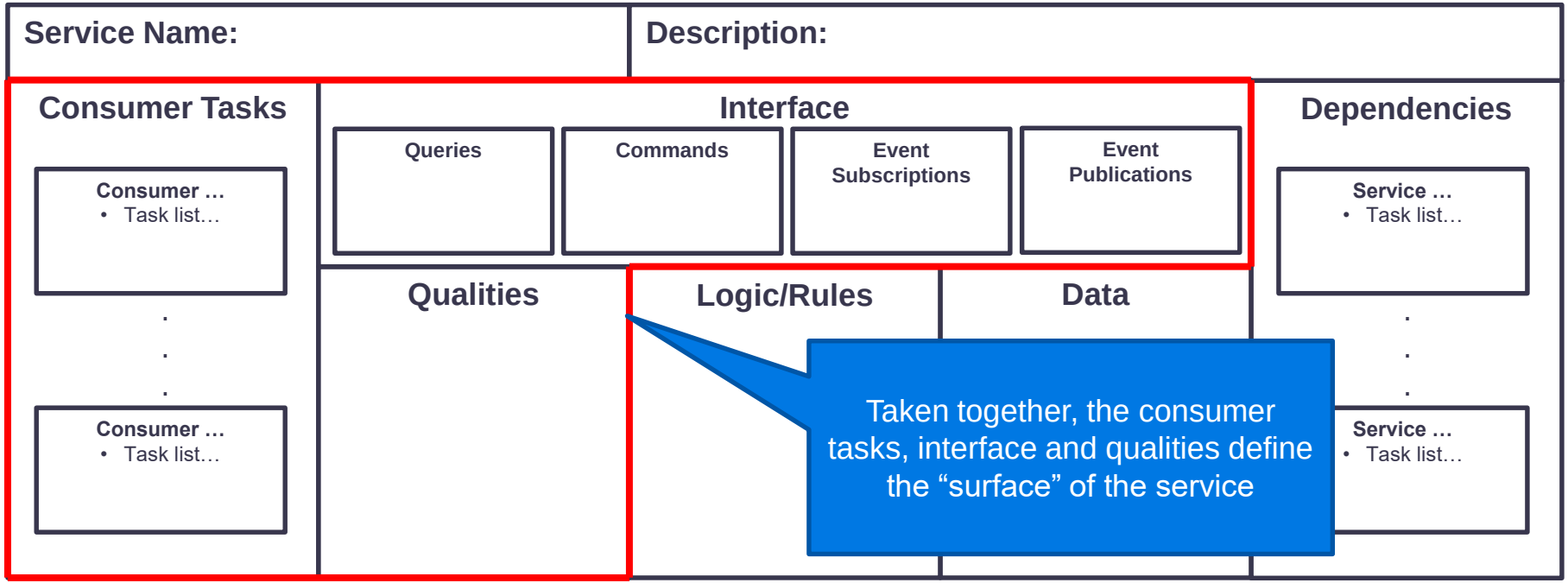
Microservice Design Canvas



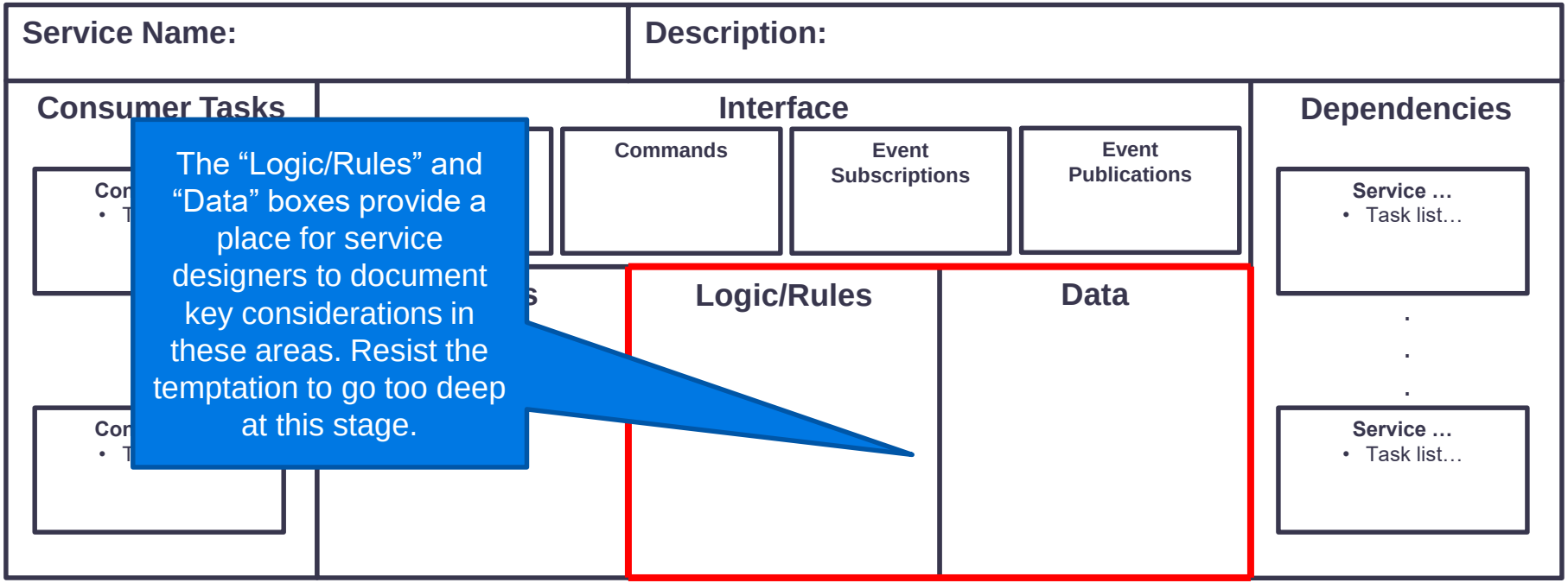
Microservice Design Canvas



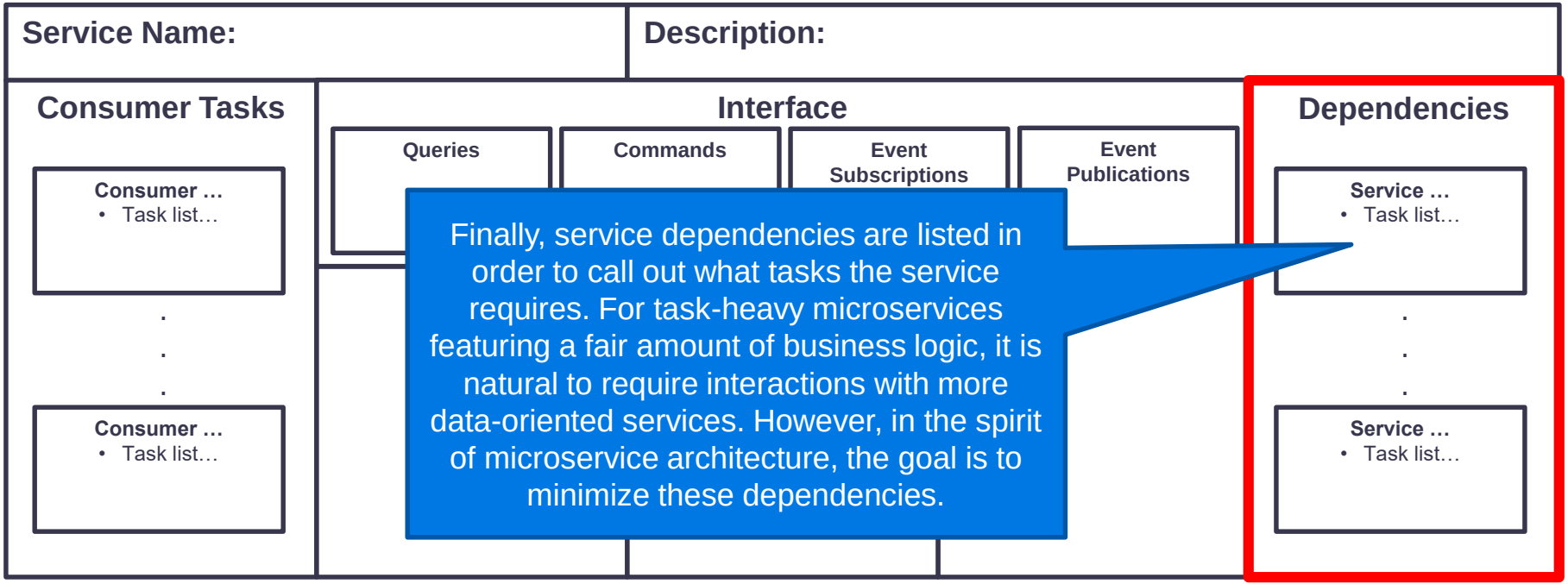
Microservice Design Canvas



Microservice Design Canvas



Microservice Design Canvas



Microservice Design Canvas

Service Name: Customer-centric Payments Management Service

Description: The Customer-centric Payments Management Service allows consumers to sign up and administer the preferences for the "Customer-centric Payments" product.

Consumer Tasks

Banking Customer using Online Banking Web App
- Sign up for payments service
- Opt out of payments service

Branch CSR using Branch Banking Desktop App
- Sign customer up for payments service

Marketing Web App
- Identify customers for payments promotion

Interface

Queries
- Query customer payments status and preferences

Commands
- Opt in
- Opt out
- Update preferences

Event Subscriptions

Event Publications

Qualities

- Audited
- Low volume
- Non-critical
- Delegated authorization
- Backward compatibility for interface versions

Logic/Rules

- Minimum accounts/products required for signup
- Role-based permissions

Data

- Customer signup status for customer-centric payments
- Customer preferences for customer-centric payments

Dependencies

Customer Information Service
- Obtain list of customer accounts and products

Microservice Design Canvas

Service Name: Customer-centric Payments Authorization Service

Description: The Customer-centric Payments Authorization Service allows consumers to authorize transfers or payments using the “Customer-centric Payments” product.

Consumer Tasks

Banking Customer using Online Banking Web or Mobile App
- Make a payment

Banking Customer using POS device
- Make a payment

Interface

Queries
- Query customer payments status

Commands
- Request payment authorization

Event Subscriptions
- Customer preparing to make customer-centric payment

Event Publications
- Post payment transaction

Qualities

- Audited
- High volume
- Mission critical
- Direct customer authentication (strong)
- Backward compatibility for interface versions

Logic/Rules

- Build and cache customer authorization profile from customer-centric payment preferences and customer activity analysis
- Authorize or decline based on authorization profile
- Select posting account based on customer-centric payment preferences

Data

- Customer authorization profile

Dependencies

Customer-centric Payments Service
- Obtain customer preferences

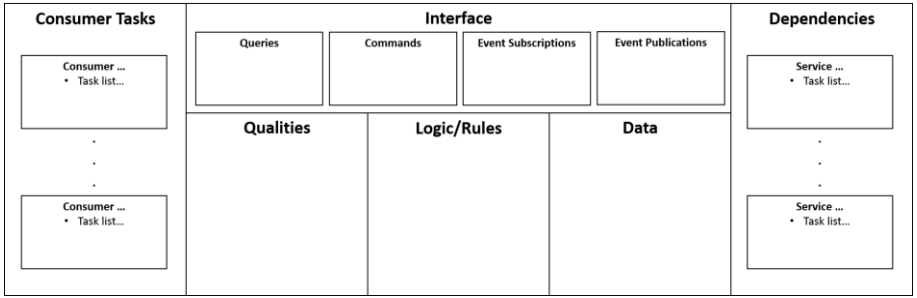
Customer Activity Analysis Service
- Obtain customer payment analytics

Transaction Posting Service
- Post payment transaction

Exercise: Microservice Design Canvas

Time: 15 minutes

- The purpose of this exercise is to practice defining services from a consumer-focused, task-based perspective in order to drive the service's attributes, qualities and other details
- Use the Microservice Design Canvas to define one of the services from the Context Mapping Exercise



Context Maps and Service Canvases

- Responsibility for System Design and Service Design should be loosely coupled
- However, the two are closely linked, especially context maps and service canvases
- It’s a good idea to revisit the context map when the services have been sketched using the design canvas

Designing the Interface

Service Design

- There is confusion in the industry…
  - “API Design” conventional wisdom actually focuses on “API Definitions”
  - “REST” may often refers to CRUD operations on resources (that’s not REST)
- Separate API design from API definition and implementation
  - Semantics, not syntax
- Also different types of APIs
  - Control plane vs. data plane

API Design Methodology

Service Design – Designing the Interface

1. Select the Service Canvas
2. Complete the Interface Details (args, responses)
3. Draw a State Diagram
4. Normalize Names
5. Describe the Interface (ALPS)
6. Publish your Interface Elements (Profile, Rels, etc.)

Determining the Service Implementation

Service Design

- Context maps and service canvases are **implementation- and technology-agnostic**
  - For example, all services in a context map could be implemented in the same monolithic app
- It's useful to design the system and the services without implementation **assumptions** and **constraints**
  - But once those design artifacts are created, it's time to look at the implementation approach

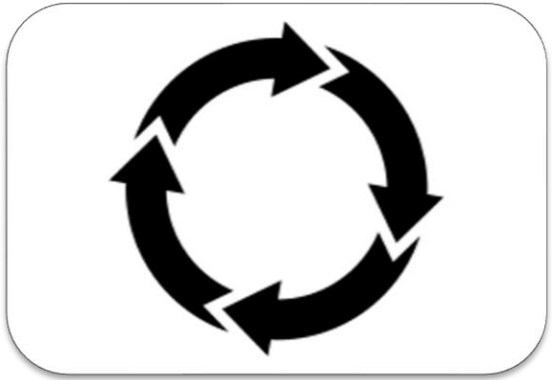
Discovering Services

Service Design - Determining the Service Implementation

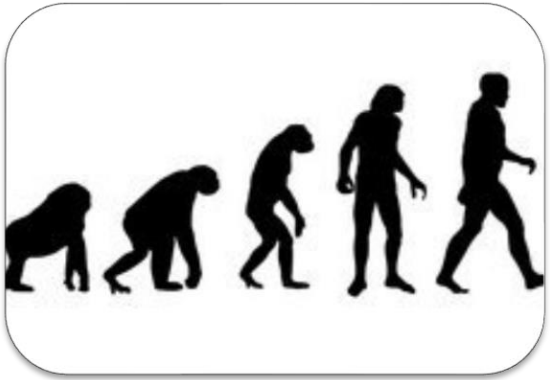
- “Service Discovery” is an overloaded term in the microservice architecture landscape
- Typically refers to runtime discovery of service instances
  - A capability provided by tools like Eureka (Netflix), Consul (Hashicorp), Zookeeper (Apache), etcd (CoreOS)
- But what about design time discovery?
  - What services already exist in your organization’s ecosystem?
  - What assets exist that might be building blocks for services?

Implementation Options

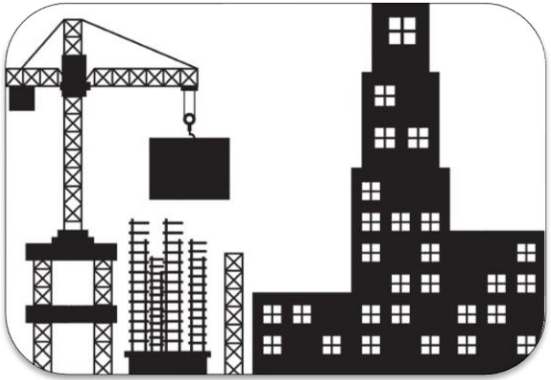
Service Design - Determining the Service Implementation



Reuse existing service



Evolve an existing service



Build a net new service

Implementation Option Decision Process

Service Design - Determining the Service Implementation

Reuse existing service?

- Does a semantically equivalent service already exist?
- Does it meet the qualities of service?
- Is the interface sufficient?

Evolve an existing service?

- Does a related service already exist that can be extended?
- What needs to be done to the protocol, interface, and QoS to make it usable?

Build a net new service?

- Are you sure there is no reasonable starting point with an existing service or asset?

Implementation Decisions - SingleMalt

Service Design - Determining the Service Implementation

Reuse

- Customer Authentication Service
- Customer Information Service

Evolve

- Transaction Posting Service
- Investment Account Service
- Deposit Account Service
- Personal Lending Service
- Credit Card Service
- Mortgages Service

Build

- Customer-centric Payments Management Service
- Customer-centric Payments Authorization Service
- Customer Activity Analysis Service

Service Design - Summary

- **Sketch** the service
- Design and define the **interface**
  - API and contract
  - Prototype it
- Determine the **implementation**
  - Find a reusable service?
  - Evolve an existing service?
  - Develop net new service?

Foundation Design

Foundation Design

- Assess technological **capabilities**
  - For the system and services
- Determine capability **implementations**
  - Platforms and tools
- Define **standards** and **guidelines**
  - From principles and capabilities (weigh the incentives)

Assess Technological Capabilities

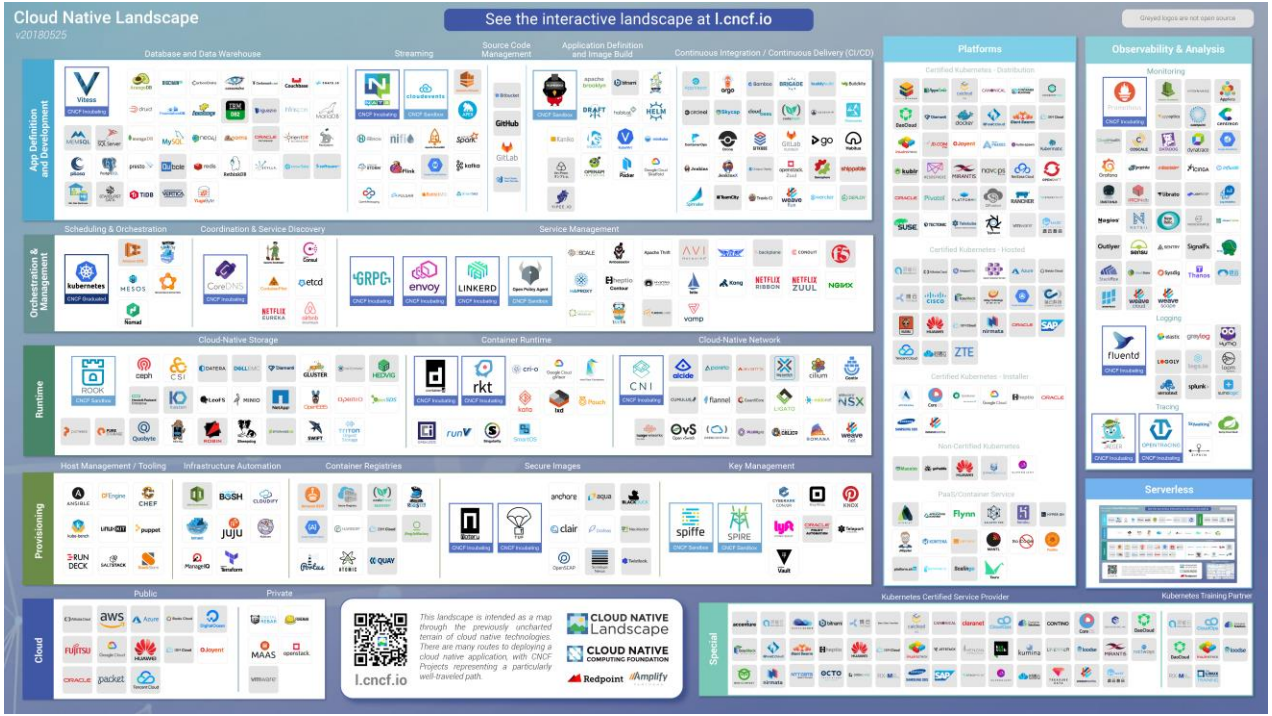
Foundation Design

- Functionality provided by enabling technologies (platforms, tools)
- Identify needed capabilities that support objectives, system needs (functional and non-functional), service qualities
- Focus first on the capability, then the underlying tool
- Capabilities examples from ***Microservice Architecture***:

Shared Capabilities	Local Capabilities
• Hardware services • Code management, testing, and deployment • Data stores • Service orchestration • Security and identity	• General tooling • Runtime configuration • Service discovery • Request routing • System observability

Cloud Native Computing Foundation (CNCF)

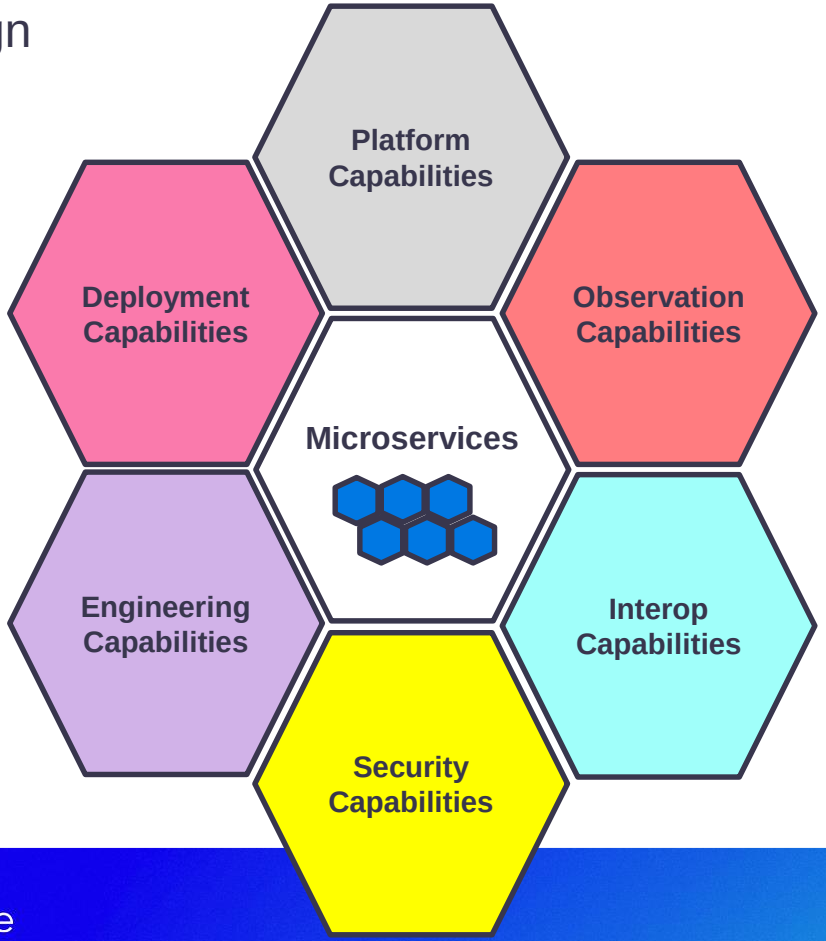
Foundation Design



From https://raw.githubusercontent.com/cncf/landscape/master/landscape/CloudNativeLandscape_latest.png

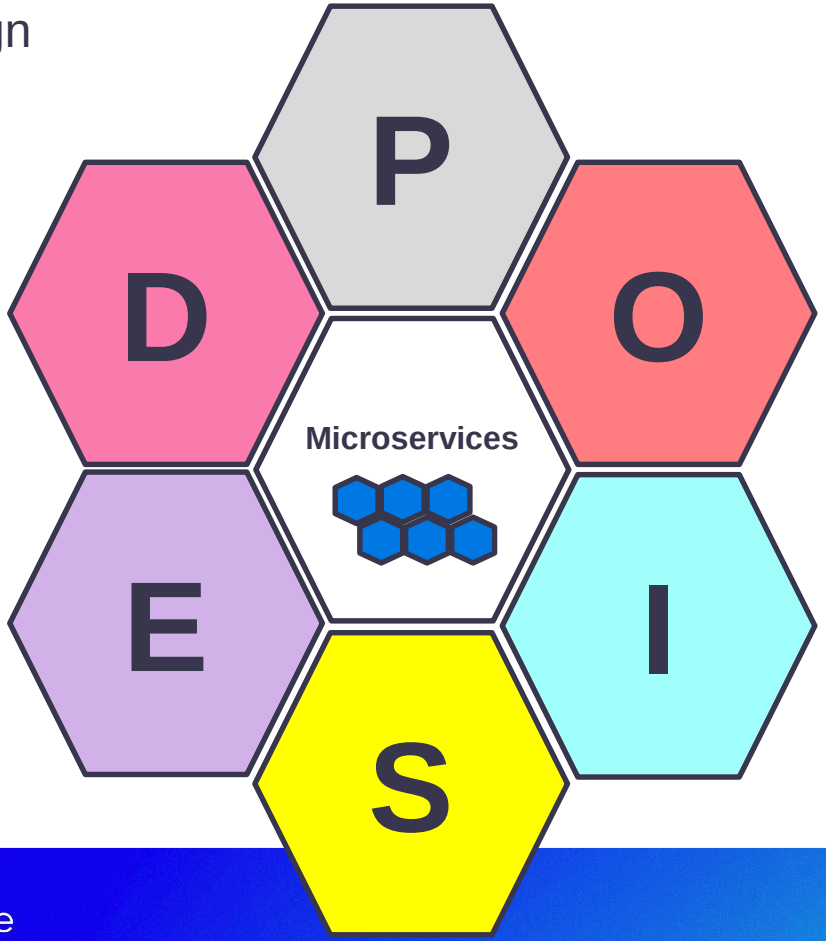
POISED: A Technological Capability Foundation

Foundation Design



POISED: A Technological Capability Foundation

Foundation Design



POISED: A Technological Capability Foundation

Foundation Design

Platform

- Container Runtime
- Container Orchestration
- IaaS/PaaS/ FaaS
- Infrastructure Provisioning
- DB/Data Management
- Storage
- Network

Observation

- Logging
- Correlation/ Tracing
- Analytics
- Anomaly Detection
- Event Triggering (Ops)

Interop

- Routing
- Transcoding
- Service Discovery (Runtime)
- Service Level Management
- Service Coordination
- Runtime Policy Enforcement/ Filtering
- Message Delivery

Security

- Access Control
- Identity Management
- PKI/Key Management
- Network Segmentation
- Threat Mitigation
- Privacy Management
- Vulnerability testing
- Code analysis

Engineering

- API Design & Definition
- IDEs
- Test Automation
- Service Discovery (Design Time)
- Container Creation
- Build Automation

Deployment

- CI/CD Pipelines
- Configuration Management/ Version Control
- Release Management
- Change Management

POISED Worksheet

Foundation Design

Category:

Capability	Requirements	Current State	Action Plan

List the POISED capabilities that are highest priority to address for your organization

Note what the organization requires or is hoping to accomplish specifically with each capability

State where the organization currently stands with respect to this capability

List what the organization needs to do to close the gap between the Requirements and the Current State

Using the POISED Worksheet

Foundation Design

- Use one page per capability category if needed
- Helpful inputs:
  - Non-functional domains
  - Design Canvas “qualities”
- Pick 1 or 2 categories and a few (5-8) key capabilities to start with
- Assess current state of capabilities and action plan to meet the requirements

POISED Worksheet

Foundation Design

Category: Interop			
Capability	**Requirements**	**Current State**	**Action Plan**
Service Discovery (Runtime)	Need the ability to locate healthy instances of services at runtime for high scale, ephemeral services	Relying on DNS, significant latency	Implement service registry/discovery capability initially for Transaction Posting service; use for others if successful
Runtime Policy Enforcement	Need the ability to enforce system wide policies for security and monitoring	Inconsistent implementation, many duplicate code libraries that are a nightmare to change across the board when a system wide policy changes	Globalize security and monitoring policy enforcement into components outside the core service logic
Service Level Management	Need to be able to protect service qualities through interventive approach	Rate limits and traffic shaping mostly done through proprietary service code if at all	Externalize service level management from services
Message Delivery	Need high scale streaming for event distribution	Most asynchronous messaging done using legacy, transactional message queues that are low scale and brittle	Introduce stream-friendly message distribution capability for pushing product events to Customer Activity Analysis Service

POISED Worksheet

Foundation Design

Category: Engineering

Capability	Requirements	Current State	Action Plan
API Design & Definition	Need the ability to model API specifications prior to coding in order to provide contracts to consumers early, and to encourage loose coupling of interface and implementation	Most API definitions are generated from the implementation code, and changes to code can result in breaking interface changes for consumers	Take “API First” approach with net new services in SingleMalt initiative
Test Automation	Need tooling to support move to automated testing as the default for most stages in the development lifecycle	Resistance from QA teams to migrating practices has led to limited adoption of automated testing tools	Raise priority of changing testing practices (see Practice Design below) and investigate automated test tooling for load testing, regression testing, more
Build Automation	Must ensure software builds provide as little friction as possible in development lifecycle	Fairly widespread use of build automation tooling across the organization, but lacking consistency	Assess impact of inconsistency, but no immediate action required
Service Discovery (Design Time)	Critical to be able to identify services that can be used as building blocks for future initiatives	Some newer services available in homegrown API portal, but most either in ESB metadata (hard to access) or legacy services that are “not in the census”	Iteratively work to publish services on API developer portal, provided they meet minimum requirements around protocol, documentation and design

Homework Exercise: POISED Worksheet

- The purpose of this exercise is to understand how technological capabilities enable enterprise-wide and initiative-specific goals
- Using the following…
  - Objectives and principles from Program Design
  - Service qualities from Design Canvases
  - Non-functional domains
  - POISED Capabilities
- …complete the worksheet with the critical enabling capabilities for some of your group's goals, principles and services

Category:			
Capability	Requirements	Current State	Action Plan

Determine Capability Implementations

Foundation Design

- **Don’t start here!**
  - Many enterprise change initiatives start by acquiring a “shiny new toy” and trying to work back to what it does, why that’s valuable, and how it aligns with the organization’s strategy
- Find the platforms and tools that **fit**
  - “Fit” means strategy → design → capabilities→ technologies
- Important to consider capability **intersections**
  - Implementation components need to work well together

Sample Tools that Enable POISED Capabilities

Foundation Design – Determine Capability Implementations

- **API Gateway**
  - Provides Dynamic Routing, Service Aggregation, Security Policy Enforcement, SLA Management (e.g. Rate Limiting)
- **Application Performance Management & Monitoring**
  - Provides System Monitoring, Event Monitoring, App/API Monitoring, Operational Analytics
- Capability-named products and tools
  - **Service Discovery**/Registry
  - Containers and **Container Orchestration**
  - **Infrastructure-as-a-Service** (compute, storage, network)

Tools to Support POISED Capabilities

Foundation Design – Determine Capability Implementations

Service Registry (e.g. Consul)	• Service Discovery (Runtime)
API Gateway	• Runtime Policy Enforcement • Service Level Management
Message Bus (e.g. Apache Kafka)	• Message Delivery
API Design Studio (e.g. SwaggerHub)	• API Design
Automated Testing Platform (e.g. Blazemeter)	• Test Automation
API Developer Portal	• Service Discovery (Design Time)

On Enterprise Standards

Foundation Design

- It is tempting to standardize technologies in every category of functionality
  - Many enterprise architecture teams started out with a mission to institute enterprise standards
  - The goals are often “consistency” or “alignment”, but how do those objectives help the enterprise?
- Resist the temptation!
- Instead, use more carrot and less stick
  - Standardize only on items required for interoperability of the system
  - Provide guidelines in other areas
  - Incentivize choices in these areas through tools, support and accessibility

The JET Model for Standards

Foundation Design

- Minimize standards
  - Make sure there is a strong justification for each standard
  - Otherwise make it a guideline
- For both standards and guidelines
  - **Justification:** Communicate their purpose
  - **Education:** Provide education on how to follow them
  - **Tools:** Provide tools to help people follow them

JET Stream for Standards and Guidelines

JET Stream for Standards and Guidelines

Justification

- Should this be a standard, or a guideline?
- Why is it needed?

Education

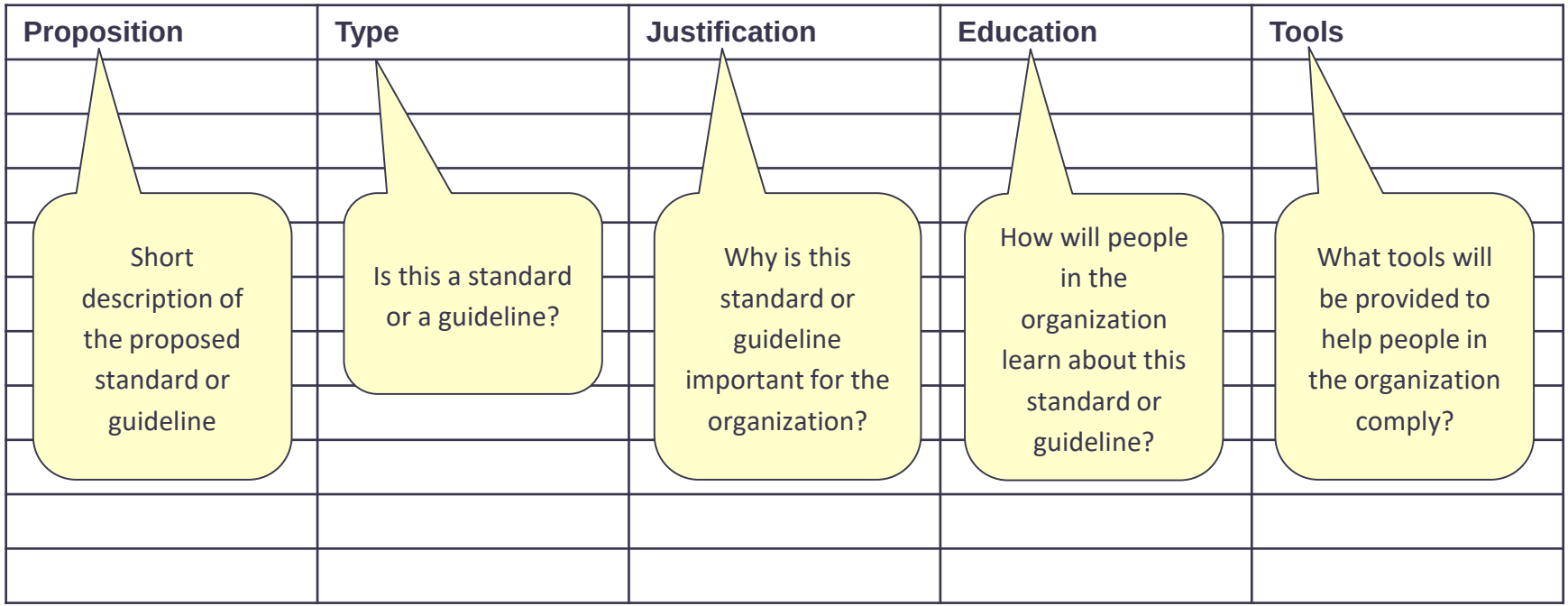
- How will people become aware of this standard/guideline?
- How can people learn how to apply it?

Tools

- What tools will be available to help people implement this standard/guideline?

JET Stream Worksheet

Foundation Design



JET Stream Worksheet

Foundation Design

Proposition	Type	Justification	Education	Tools
Web APIs (HTTP-based) will be used for integration between UI systems and microservices	Standard	Web APIs provide maximum operability, loose coupling, mature security, ubiquitous language and framework support	Web API Design Guidelines Document API Special Interest Group (Guild) On Demand API Training	Helper libraries for Java, .NET, JavaScript API Gateway for legacy adaptation
Web APIs should support more than one data format	Guideline	Consumers may require different formats, and supporting multiple formats helps providers avoid the implementation bleeding into the interface	Documentation on decoupling interface and implementation Decision criteria for selecting data formats (XML, JSON, hypermedia formats, Protobuf, etc)	Libraries for supporting multiple HTTP media types (XML, JSON, HAL) API Gateway policies for transcoding formats
Individual microservices should supply discoverable metadata through a web accessible API	Guideline	To get an accurate view of what is running in the system, metadata must be collected from the system itself, not stale and disconnected documentation	Training on OpenAPI spefication Collaborative work on formulating metadata specifications for IsB	Plug-in components on multiple platforms (Docker-based, API Gateway-based) to provide metadata
Docker will be used as the standard container runtime	Standard	Container runtime a commoditized capability, with Docker dominating the adoption	Training on using Docker in all contexts (image creation, deployment, orchestration)	Docker toolset, container orchestration toolsets, sample Compose files, etc
Services should have unique identities	Guideline	In a distributed system, all components should be identifiable in order to track and factor into authorization	API and microservice security basics course	SPIFFE framework, SPIRE runtime, SVID certificates

Foundation Design - Summary

- Assess technological **capabilities**
  - For the system and services
- Determine capability **implementations**
  - Platforms and tools
- Define **standards** and **guidelines**
  - From principles and capabilities (weigh the incentives)

Practice Design

The Human Side of Microservices

Practice Design

- An organization's ecosystem of microservices is intrinsically linked to…
  - …the processes and methodologies used for delivery and management
  - …the people doing the work and how they are organized
  - …the cultural norms that incent and constrain behavior
- These are the human dimensions of software systems, and…
- **Change is always harder when people are involved!**

Observed Microservices Best Practices

Processes & Methodologies

- Agile software development
- Continuous integration/continuous delivery (build & deployment automation)
- Test automation
- Operational automation

Organizational Practices

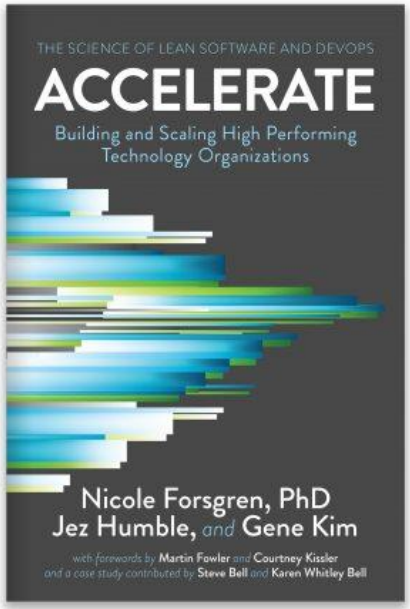
- Small team size
- Business alignment
- Cross-organizational supporting teams
- “Guilds”
- Organizational design

Cultural Practices

- Stated principles
- Team autonomy and empowerment
- Two way communication
- Change tolerance
- Experimental and iterative approach
- Toolmaking and enablement (vs. “governance”)

Accelerate – The Science of Lean Software & DevOps

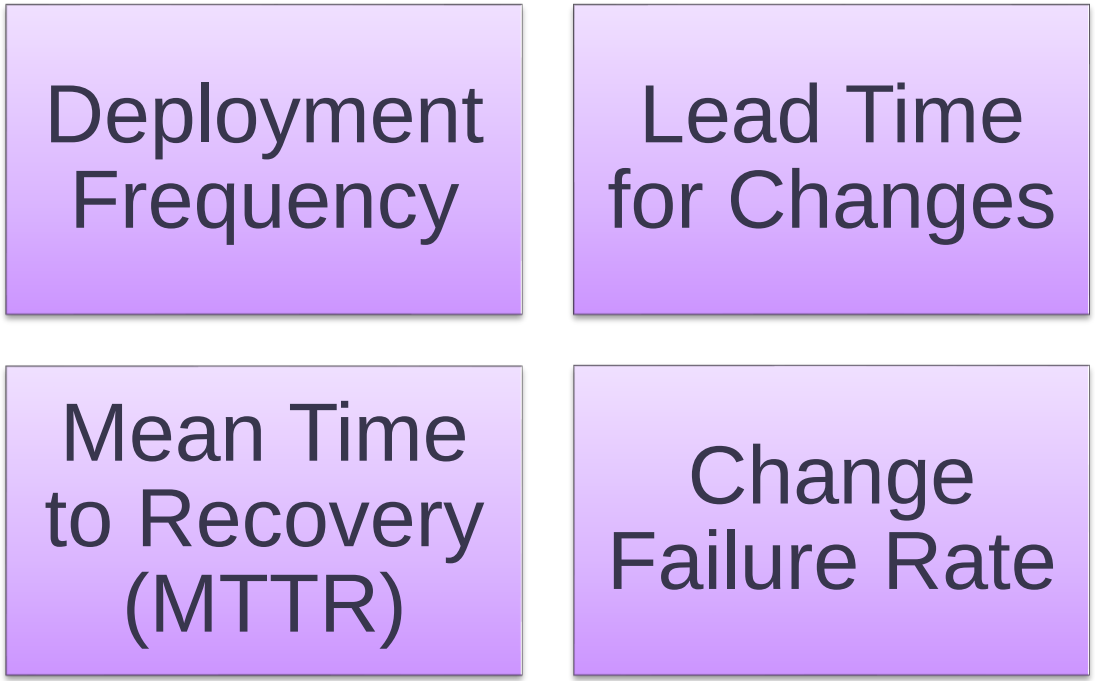
Practice Design



https://itrevolution.com/book/accelerate/

Accelerate – Measuring Delivery Performance

Practice Design



From https://itrevolution.com/book/accelerate/

Accelerate – Fundamental Organizational Capabilities

Practice Design

Continuous Delivery Capabilities

- Version control
- Deployment automation
- Continuous integration
- Trunk-based development
- Test automation
- Test data management
- Shift left on security
- Continuous delivery (CD)

Architecture

- Loosely coupled architecture
- Empowered teams

Lean Management and Monitoring Capabilities

- Change approval processes
- Monitoring
- Proactive notification
- WIP limits
- Visualizing work

Product and Process Capabilities

- Customer feedback
- Value stream
- Working in small batches
- Team experimentation

Cultural Capabilities

- Westrum organizational culture
- Supporting learning
- Collaboration among teams
- Job satisfaction
- Transformational leadership

From https://itrevolution.com/book/accelerate/

Must Have Organizational Capabilities for Microservices

Practice Design

Test Automation

CI/CD/Deployment Automation

Supporting Learning

More Human Considerations

Practice Design

- Who is responsible for designing the overall system of microservices?
- What about maintaining the system?
- How is the organization structured? How well does the structure align with the target system design?
- How effective is communication across organizational boundaries?
- What is the tolerance for change in the organization? Does it vary?
- How will skill gaps be addressed?

Human Considerations at IsB

Practice Design

- Retail Banking architecture team (embedded LOB architects plus enterprise architects) responsible for final say on domain architecture
- Retail Banking Platform organization responsible for system concerns
- Retail Banking CIO evaluating organizational change from horizontal technology teams to cross-function business-aligned teams
  - Will include cross-Retail platform and architecture team with assigned ownership for the service landscape
- IsB CIO starting “Innovation Ingrained” program with associated awards
- For a real world story…
  - Holger Reinhardt, “The automated monolith” - http://transform.ca.com/API-Microservices-Best-Practices-API360-Summit-Videos.html

The Result

Photo credit: https://blog.thewhiskyexchange.com/core/wp-content/uploads/2016/10/Bowmore-...

Summary: Microservice-based Enterprise Transformation Approach (META)

Program Design	• **Define the goals and align the organization** in order to point your microservice adoption efforts in the right direction
System Design	• **Decompose the system** in order to ensure its independent evolvability over time
Service Design	• **Design the services** in order to maximize interoperability and extensibility of the system components
Foundation Design	• **Determine the needed capabilities** in order to optimize and focus the developer experience
Practice Design	• **Adapt core practices** in order to remove cultural inertia and build sustainable momentum

For More Information…

Fowler and Lewis on Microservices	• https://www.martinfowler.com/articles/microservices.html
Microservice Architecture from O'Reilly Media	• http://shop.oreilly.com/product/0636920050308.do
Continuous API Management	• http://shop.oreilly.com/product/0636920201755.do
Domain Driven Design Book	• https://domainlanguage.com/ddd/
Securing Microservice APIs from O'Reilly Media	• https://www.apiacademy.co/resources/books/securing-microservice-apis
Microservice Design Canvas	• http://www.apiacademy.co/the-microservice-design-canvas/

Thank you!