META for microservices
Getting your enterprise migration in motion

Matt McLarty, Chief Architect, Islay Bank
@mattmclartybc
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 the jettisoning of PC businesses in favor of investing more heavily in software, where it sees better potential for growth. Google plans to buy the cellphone handset maker Motorola Mobility. Both moves surprised us both, as is in line with a trend I’ve observed, one that makes me optimistic about the future of the American and world economies, despite the recent turmoil on Wall Street.

In short, software is eating the world.

More than 50 years after the peak of the 1960s, a dozen or so now Internet companies like Google, Twitter are sparking controversy in Silicon Valley, rapidly growing private market valuations, an influential, successful IPO. With news from the Web 2.0 and Pets.com still fresh in the investors’ minds, are we asking, “Isn’t this just a dangerous new bubble, along with others, have been argving the old case? I am a co-founder and general partner of Andreessen-Horowitz, which has invested...

**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 adopt right now will soon find themselves obsolete.

---

**Forbes**

*Now Every Company Is A Software Company*

By Marc Andreessen

In an interview with Daniel K. Eakle, Durling, Grant and Lincolin investor Marc Andreessen notes that the

---

**Disrupting the Public Sector**

By William D. Eggers and Ruben Gonzalez

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

- Data Digitization
- Process Optimization
- Information Operationalization
Digital Transformation

- Data Digitization
- Process Optimization
- Information Operationalization

API
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 an order of magnitude improvement within a decade in productivity, in reliability, in simplicity.

Abstract

All software construction involves essential tasks, the fleshing of the complex conceptual structures that compose the abstract software entity, and accidental tasks, the representation of those abstract entities in programming languages and the mapping of those 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 incredibly 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 fleshing 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 strategy 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 designs of the rising generation.

Introduction

Of all the monsters that fill the nightmares of our fathers, none terrify them more than座席者, because they transform unexpectedly from the familiar into horrors. For these, we seek bullets of silver to cast magically lay them to rest.

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

- David Parnas, *On the Criteria To Be Used in Decomposing Systems into Modules*, 1972
"Modularity … is to a technological economy what the division of labor is to a manufacturing one."

- W. Brian Arthur, *The Nature of Technology*
“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
“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
“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

Service Perspective
Microservices Design Thinking
Microservices Design Thinking

- Ecosystem Perspective
- System Perspective
- Service Perspective
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

- Program Design
- System Design
- Foundation Design
- Practice Design
- Service Design
META at a Glance

5 Design Disciplines, 15 Processes, 9 Memes

**Program Design**
- Create the case for change
- Establish momentum-oriented oversight
- Prime the program

**System Design**
- Define the target system scope
- Determine the functional domains
- Determine the non-functional domains

**Service Design**
- Sketch the service
- Design and the interface
- Decide the implementation approach

**Foundation Design**
- Assess technological capabilities
- Select capability implementations
- Outline standards and guidelines

**Practice Design**
- Align with core processes
- Address organizational blind spots
- Evolve processes & culture toward adaptability

**Memes**
- Com4
- AIM
- Bedlam
- Dharma
- Microservice Design Canvas
- Star
- Poised
- Jet
- Cate3
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
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.

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

I’ll retire to BEDLAM!
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

• 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
“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

Customer and Card Management

Customer Identity

Customer Information

Consumer Payments & Transactions

Self-Service Banking

Customer Identity

Deposit Accounts

Investment Accounts

Deposit Accounts

PLC Accounts

Credit Cards

Mortgages

Online Banking

Point of Sale

Retail Lending and Credit

Mortgages
“SingleMalt” with BEDLAM
“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**

- **Self-Service Banking**
  - Online Banking
  - Point of Sale

- **Customer and Card Management**
  - Customer Identity
  - Customer Information
  - Consumer Payments & Transactions

- **Investment Banking**
  - Investment Accounts
- **Deposit Accounts**
  - Deposit Accounts
- **Retail Lending and Credit**
  - PLC Accounts
  - Credit Cards
- **Mortgages**

**Queries**
- Q: Provide option to sign up and present preference options
- Q: Retrieve the customer's products
- C: Authenticate the customer
- C: Sign the customer up and configure payment preferences
“SingleMalt” Interactions
Customer changes preferences

Queries
Commands
Events

Self-Service Banking
Customer and Card Management
Investment Banking
Investment Accounts
Deposit Accounts
Deposit Accounts
Retail Lending and Credit
PLC Accounts
Credit Cards
Mortgages
Mortgages

Online Banking
Customer Identity

Point of Sale
Customer Information

Q: Authenticate the customer
C: Change payment preferences

Q: Provide current customer preferences
C: Change payment preferences

Q: Retrieve the customer's products
“SingleMalt” Interactions

Customer opts out

Self-Service Banking

Customer and Card Management

Customer Identity

Customer Information

Investment Banking

Investment Accounts

Deposit Accounts

Deposit Accounts

Retail Lending and Credit

PLC Accounts

Credit Cards

Mortgages

Mortgages

Online Banking

Point of Sale

Consumer Payments & Transactions

Q: Provide current customer preferences and signup status

C: Authenticate the customer

C: Opt out of payment service

Q:

C:
"SingleMalt" Interactions
Customer requests authorization

Customer and Card Management
Customer Identity
Customer Information
Consumer Payments & Transactions

Self-Service Banking
Online Banking
Point of Sale

C: Authenticate the customer
C: Request payment authorization
C: Request payment authorization

Investment Banking
Investment Accounts
Deposit Accounts
Deposit Accounts
Retail Lending and Credit
PLC Accounts
Credit Cards
Mortgages
Mortgages
“SingleMalt” Interactions
Transaction gets fulfilled

- Self-Service Banking
- Online Banking
- Point of Sale

- Customer and Card Management
- Customer Identity
- Customer Information

- Consumer Payments & Transactions

- Investment Banking
  - Investment Accounts
- Deposit Accounts
  - Deposit Accounts
- Retail Lending and Credit
  - PLC Accounts
  - Credit Cards
- Mortgages
  - Mortgages

E: Post transaction to product service
“SingleMalt” Interactions
Activity on customer product
"SingleMalt" Interactions
Customer requests authorization

Self-Service Banking
Online Banking
Point of Sale

Customer and Card Management
Customer Identity
Customer Information
Consumer Payments & Transactions

C: Authenticate the customer
C: Request payment authorization
E: Notification of customer sign-in
Q: Retrieve customer activity analysis

Investment Banking
Investment Accounts
Deposit Accounts
Deposit Accounts
Retail Lending and Credit
PLC Accounts
Credit Cards
Mortgages
Mortgages

oreillysacon.com/ny
#OREillySACon
"SingleMalt" Interactions Overlay – Emerging Services

### Self-Service Banking
- Customer and Card Management
  - Deposit Accounts
  - Retail Lending and Credit
- Investment Banking
  - Investment Accounts
- Deposit Accounts
- Retail Lending and Credit
  - PLC Accounts
- Mortgages
- Online Banking
  - C: Authenticate the customer
  - E: Notification of customer sign-in
  - Q: Provide option to sign up and present preference options
  - C: Sign the customer up and configure payment preferences
  - C: Change payment preferences
  - C: Opt out of payment service
  - C: Request payment authorization
  - Q: Provide current customer preferences
  - Q: Provide current customer preferences and signup status
  - Q: Retrieve the customer's products
  - Q: Retrieve customer activity analysis
  - C: Change payment preferences
  - Q: Retrieve current customer preferences and signup status
  - C: Opt out of payment service
  - C: Request payment authorization
  - Q: Provide current customer preferences and signup status

### Consumer Payments & Transactions
- Customer Information
  - Customer Identity
  - Consumer Payments & Transactions
  - C: Authenticate the customer
  - E: Notification of customer sign-in
  - Q: Provide option to sign up and present preference options
  - C: Sign the customer up and configure payment preferences
  - C: Change payment preferences
  - C: Opt out of payment service
  - C: Request payment authorization
  - Q: Provide current customer preferences
  - Q: Provide current customer preferences and signup status
  - Q: Retrieve the customer’s products
  - Q: Retrieve customer activity analysis
  - C: Change payment preferences
  - Q: Retrieve current customer preferences and signup status
  - C: Opt out of payment service
  - C: Request payment authorization
  - Q: Provide current customer preferences and signup status

### Queries
- C: Authenticate the customer
- E: Notification of customer sign-in
- Q: Provide option to sign up and present preference options
- C: Sign the customer up and configure payment preferences
- C: Change payment preferences
- C: Opt out of payment service
- C: Request payment authorization
- Q: Provide current customer preferences
- Q: Provide current customer preferences and signup status
- Q: Retrieve the customer’s products
- Q: Retrieve customer activity analysis
- C: Change payment preferences
- Q: Retrieve current customer preferences and signup status

### Commands
- E: Notification of customer sign-in
- Q: Provide option to sign up and present preference options
- C: Sign the customer up and configure payment preferences
- C: Change payment preferences
- C: Opt out of payment service
- C: Request payment authorization
- Q: Provide current customer preferences
- Q: Provide current customer preferences and signup status
- Q: Retrieve the customer’s products
- Q: Retrieve customer activity analysis
- C: Change payment preferences
- Q: Retrieve current customer preferences and signup status

### Events
- E: Notification of Account/Product Activity
- E: Post transaction to product service

---

oreillysacon.com/ny
#OReillySACon
“SingleMalt” Services
Context Map

- Self-Service Banking
  - Online Banking Web App
  - Mobile Banking App
  - POS Networks (3rd Party)

- Customer and Card Management
  - Customer Authentication Service
  - Customer Information Service
  - Customer Activity Analysis Service
  - Transaction Posting Service
  - Customer-centric Payments Management Service
  - Customer-centric Payments Authorization Service

- Investment Banking
  - Investment Account Service

- Deposit Accounts
  - Deposit Account Service

- Retail Lending and Credit
  - Personal Lending Service
  - Credit Card Service

- Mortgages
  - Mortgages Service

oreillysacon.com/ny
#OReillySACon
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

<table>
<thead>
<tr>
<th>Service Name:</th>
<th>Description:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consumer Tasks</td>
<td>Interface</td>
</tr>
<tr>
<td>Consumer ... • Task list...</td>
<td>Queries</td>
</tr>
<tr>
<td>.</td>
<td>Commands</td>
</tr>
<tr>
<td>.</td>
<td>Event Subscriptions</td>
</tr>
<tr>
<td>Consumer ... • Task list...</td>
<td>Event Publications</td>
</tr>
<tr>
<td>.</td>
<td>Data</td>
</tr>
</tbody>
</table>

### Consumer Tasks
- Consumer ...
  - Task list...
- ...
- ...
- Consumer ...
  - Task list...

### Interface
- Queries
- Commands
- Event Subscriptions
- Event Publications

### Dependencies
- Service ...
  - Task list...
- .
- .
- Service ...
  - Task list...
- .
- .
Microservice Design Canvas

<table>
<thead>
<tr>
<th>Service Name:</th>
<th>Description:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consumer Tasks</td>
<td></td>
</tr>
<tr>
<td>Consumer ...</td>
<td>Task list...</td>
</tr>
<tr>
<td>Consumer ...</td>
<td>Task list...</td>
</tr>
<tr>
<td>Quality:</td>
<td></td>
</tr>
<tr>
<td>Interface:</td>
<td></td>
</tr>
<tr>
<td>Dependencies:</td>
<td></td>
</tr>
</tbody>
</table>

Enumerating the consumers of the service along with the tasks they need to perform helps to crystallize the purpose of the service and provides the material inputs needed to design the interface.
Consumer tasks can be broken down into interactions with the service interface. Classifying interactions according to patterns—queries, commands, events—will help shape the underlying service implementation.
In addition to the tasks and interactions for the service—what it does—we must also consider the non-functional aspects of the service—what it is. Identifying qualities such as availability and performance levels, extensibility approaches, and security expectations help further consumers’ understanding of the service and also influence its implementation.
Taken together, the consumer tasks, interface and qualities define the “surface” of the service.
<table>
<thead>
<tr>
<th>Consumer Tasks</th>
<th>Interface</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Commands</td>
<td>Service ...</td>
</tr>
<tr>
<td></td>
<td>Event Subscriptions</td>
<td>Task list…</td>
</tr>
<tr>
<td></td>
<td>Event Publications</td>
<td>Service ...</td>
</tr>
<tr>
<td></td>
<td>Logic/Rules</td>
<td>Task list…</td>
</tr>
<tr>
<td></td>
<td>Data</td>
<td>Service ...</td>
</tr>
</tbody>
</table>

The “Logic/Rules” and “Data” boxes provide a place for service designers to document key considerations in these areas. Resist the temptation to go too deep at this stage.
Finally, service dependencies are listed in order to call out what tasks the service requires. For task-heavy microservices featuring a fair amount of business logic, it is natural to require interactions with more data-oriented services. However, in the spirit of microservice architecture, the goal is to minimize these dependencies.
## 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

<table>
<thead>
<tr>
<th>Banking Customer using Online Banking Web App</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Sign up for payments service</td>
</tr>
<tr>
<td>• Opt out of payments service</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Branch CSR using Branch Banking Desktop App</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Sign customer up for payments service</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Marketing Web App</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Identify customers for payments promotion</td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>Service Name:</th>
<th>Description:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer-centric Payments Authorization Service</td>
<td>The Customer-centric Payments Authorization Service allows consumers to authorize transfers or payments using the “Customer-centric Payments” product.</td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>Reuse existing service?</th>
<th>Evolve an existing service?</th>
<th>Build a net new service?</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Does a semantically equivalent service already exist?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Does it meet the qualities of service?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Is the interface sufficient?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Does a related service already exist that can be extended?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• What needs to be done to the protocol, interface, and QoS to make it usable?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Are you sure there is no reasonable starting point with an existing service or asset?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Implementation Decisions - SingleMalt

## Service Design - Determining the Service Implementation

<table>
<thead>
<tr>
<th>Reuse</th>
<th>Evolve</th>
<th>Build</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Customer Authentication Service</td>
<td>• Transaction Posting Service</td>
<td>• Customer-centric Payments Management Service</td>
</tr>
<tr>
<td>• Customer Information Service</td>
<td>• Investment Account Service</td>
<td>• Customer-centric Payments Authorization Service</td>
</tr>
<tr>
<td></td>
<td>• Deposit Account Service</td>
<td>• Customer Activity Analysis Service</td>
</tr>
<tr>
<td></td>
<td>• Personal Lending Service</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Credit Card Service</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Mortgages Service</td>
<td></td>
</tr>
</tbody>
</table>

---

**Image:**
- [IsB](https://isb.com) logo

**Source:**
- oreillysacon.com/ny
- #OReillySACon

**Software Architecture**
- O'Reilly
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*:

<table>
<thead>
<tr>
<th>Shared Capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Hardware services</td>
</tr>
<tr>
<td>• Code management, testing, and deployment</td>
</tr>
<tr>
<td>• Data stores</td>
</tr>
<tr>
<td>• Service orchestration</td>
</tr>
<tr>
<td>• Security and identity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Local Capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>• General tooling</td>
</tr>
<tr>
<td>• Runtime configuration</td>
</tr>
<tr>
<td>• Service discovery</td>
</tr>
<tr>
<td>• Request routing</td>
</tr>
<tr>
<td>• System observability</td>
</tr>
</tbody>
</table>
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

- Platform Capabilities
- Observation Capabilities
- Microservices
- Security Capabilities
- Interop Capabilities
- Engineering Capabilities
- Deployment Capabilities
POISED: A Technological Capability Foundation

Foundation Design

Microservices
POISED: A Technological Capability Foundation

Foundation Design

<table>
<thead>
<tr>
<th>Platform</th>
<th>Observation</th>
<th>Interop</th>
<th>Security</th>
<th>Engineering</th>
<th>Deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container Runtime</td>
<td>Logging</td>
<td>Routing</td>
<td>Access Control</td>
<td>API Design &amp;</td>
<td>CI/CD Pipelines</td>
</tr>
<tr>
<td>Container</td>
<td>Correlation/Tracing</td>
<td>Transcoding</td>
<td>Identity Management</td>
<td>Definition</td>
<td>Configuration</td>
</tr>
<tr>
<td>Orchestration</td>
<td>Analytics</td>
<td>Service Discovery</td>
<td>PKI/Key Management</td>
<td>IDEs</td>
<td>Management/Version</td>
</tr>
<tr>
<td>IaaS/PaaS/FaaS</td>
<td>Anomaly Detection</td>
<td>(Runtime)</td>
<td>Network Management</td>
<td>Test Automation</td>
<td>Control</td>
</tr>
<tr>
<td>Infrastructure</td>
<td>Event Triggering</td>
<td>Service Level</td>
<td>Segmentation</td>
<td>Service Discovery</td>
<td>Release Management</td>
</tr>
<tr>
<td>Provisioning</td>
<td>(Ops)</td>
<td>Management</td>
<td>Threat Mitigation</td>
<td>(Design Time)</td>
<td>Change Management</td>
</tr>
<tr>
<td>DB/Data Management</td>
<td></td>
<td>Service Coordination</td>
<td>Privacy Management</td>
<td>Container Creation</td>
<td></td>
</tr>
<tr>
<td>Storage</td>
<td></td>
<td>Runtime Policy</td>
<td>Vulnerability</td>
<td>Build Automation</td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td></td>
<td>Enforcement/Filtering</td>
<td>testing</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Message Delivery</td>
<td>Code analysis</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

O'REILLY®
Software Architecture

oreillysacon.com/ny
#OReillySACon
## POISED Worksheet

**Foundation Design**

<table>
<thead>
<tr>
<th>Category:</th>
<th>Capability</th>
<th>Requirements</th>
<th>Current State</th>
<th>Action Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>List the POISED capabilities that are highest priority to address for your organization</td>
<td>Note what the organization requires or is hoping to accomplish specifically with each capability</td>
<td>State where the organization currently stands with respect to this capability</td>
<td>List what the organization needs to do to close the gap between the Requirements and the Current State</td>
<td></td>
</tr>
</tbody>
</table>
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

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

### Category: Engineering

<table>
<thead>
<tr>
<th>Capability</th>
<th>Requirements</th>
<th>Current State</th>
<th>Action Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>API Design &amp; Definition</strong></td>
<td>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</td>
<td>Most API definitions are generated from the implementation code, and changes to code can result in breaking interface changes for consumers</td>
<td>Take “API First” approach with net new services in SingleMalt initiative</td>
</tr>
<tr>
<td><strong>Test Automation</strong></td>
<td>Need tooling to support move to automated testing as the default for most stages in the development lifecycle</td>
<td>Resistance from QA teams to migrating practices has led to limited adoption of automated testing tools</td>
<td>Raise priority of changing testing practices (see Practice Design below) and investigate automated test tooling for load testing, regression testing, more</td>
</tr>
<tr>
<td><strong>Build Automation</strong></td>
<td>Must ensure software builds provide as little friction as possible in development lifecycle</td>
<td>Fairly widespread use of build automation tooling across the organization, but lacking consistency</td>
<td>Assess impact of inconsistency, but no immediate action required</td>
</tr>
<tr>
<td><strong>Service Discovery (Design Time)</strong></td>
<td>Critical to be able to identify services that can be used as building blocks for future initiatives</td>
<td>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”</td>
<td>Iteratively work to publish services on API developer portal, provided they meet minimum requirements around protocol, documentation and design</td>
</tr>
</tbody>
</table>
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
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 $\rightarrow$ design $\rightarrow$ capabilities $\rightarrow$ 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**

<table>
<thead>
<tr>
<th><strong>Service Registry (e.g. Consul)</strong></th>
<th>• Service Discovery (Runtime)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>API Gateway</strong></td>
<td>• Runtime Policy Enforcement</td>
</tr>
<tr>
<td></td>
<td>• Service Level Management</td>
</tr>
<tr>
<td><strong>Message Bus (e.g. Apache Kafka)</strong></td>
<td>• Message Delivery</td>
</tr>
<tr>
<td><strong>API Design Studio (e.g. SwaggerHub)</strong></td>
<td>• API Design</td>
</tr>
<tr>
<td><strong>Automated Testing Platform (e.g. Blazemeter)</strong></td>
<td>• Test Automation</td>
</tr>
<tr>
<td><strong>API Developer Portal</strong></td>
<td>• Service Discovery (Design Time)</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Proposition</th>
<th>Type</th>
<th>Justification</th>
<th>Education</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short description of the proposed standard or guideline</td>
<td>Is this a standard or a guideline?</td>
<td>Why is this standard or guideline important for the organization?</td>
<td>How will people in the organization learn about this standard or guideline?</td>
<td>What tools will be provided to help people in the organization comply?</td>
</tr>
</tbody>
</table>
## JET Stream Worksheet
### Foundation Design

<table>
<thead>
<tr>
<th>Proposition</th>
<th>Type</th>
<th>Justification</th>
<th>Education</th>
<th>Tools</th>
</tr>
</thead>
</table>
| 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 specification  
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

Accelerate – Measuring Delivery Performance

Practice Design

- Deployment Frequency
- Lead Time for Changes
- Mean Time to Recovery (MTTR)
- Change Failure Rate

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...
The Result
## Summary: Microservice-based Enterprise Transformation Approach (META)

<table>
<thead>
<tr>
<th>Design</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Program Design</strong></td>
<td>• Define the goals and align the organization in order to point your microservice adoption efforts in the right direction</td>
</tr>
<tr>
<td><strong>System Design</strong></td>
<td>• Decompose the system in order to ensure its independent evolvability over time</td>
</tr>
<tr>
<td><strong>Service Design</strong></td>
<td>• Design the services in order to maximize interoperability and extensibility of the system components</td>
</tr>
<tr>
<td><strong>Foundation Design</strong></td>
<td>• Determine the needed capabilities in order to optimize and focus the developer experience</td>
</tr>
<tr>
<td><strong>Practice Design</strong></td>
<td>• Adapt core practices in order to remove cultural inertia and build sustainable momentum</td>
</tr>
</tbody>
</table>
For More Information…

<table>
<thead>
<tr>
<th>Fowler and Lewis on Microservices</th>
<th><a href="https://www.martinfowler.com/articles/microservices.html">https://www.martinfowler.com/articles/microservices.html</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuous API Management</td>
<td><a href="http://shop.oreilly.com/product/0636920201755.do">http://shop.oreilly.com/product/0636920201755.do</a></td>
</tr>
<tr>
<td>Domain Driven Design Book</td>
<td><a href="https://domainlanguage.com/ddd/">https://domainlanguage.com/ddd/</a></td>
</tr>
<tr>
<td>Securing Microservice APIs from O’Reilly Media</td>
<td><a href="https://www.apiacademy.co/resources/books/securing-microservice-apis">https://www.apiacademy.co/resources/books/securing-microservice-apis</a></td>
</tr>
<tr>
<td>Microservice Design Canvas</td>
<td><a href="http://www.apiacademy.co/the-microservice-design-canvas/">http://www.apiacademy.co/the-microservice-design-canvas/</a></td>
</tr>
</tbody>
</table>
Thank you!