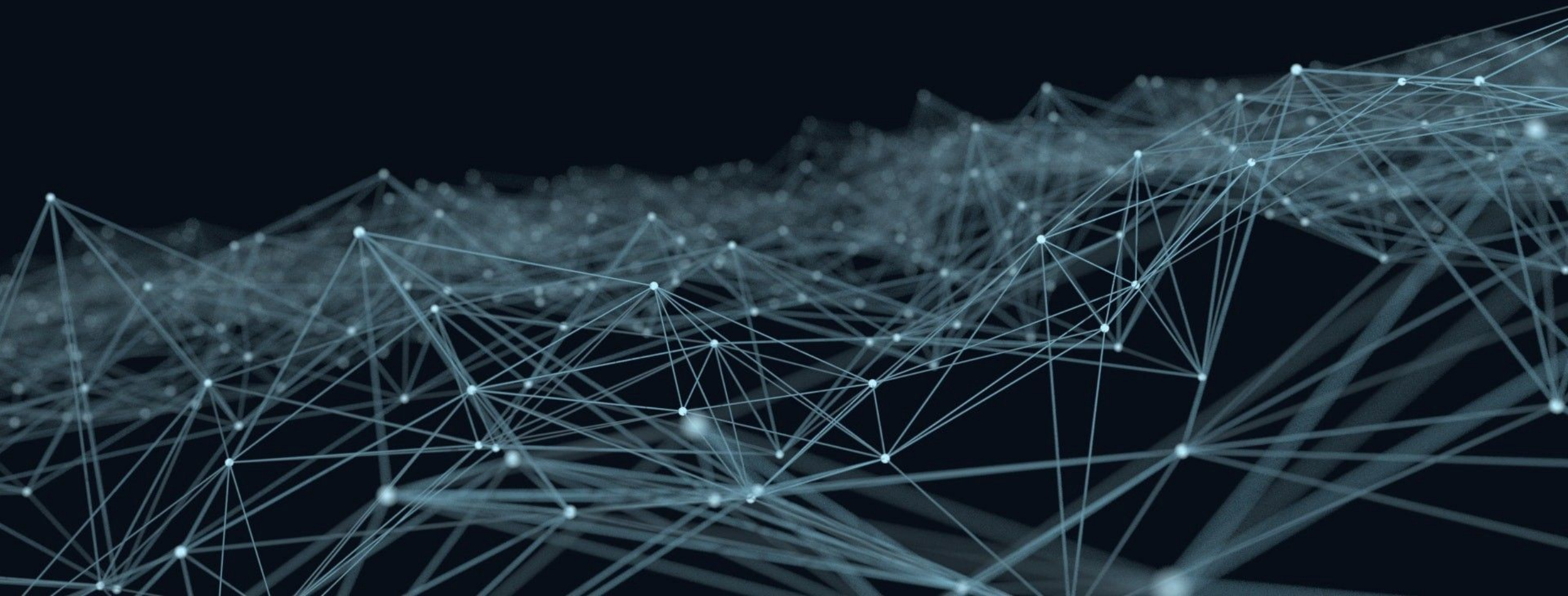
Microservices

Introduction to Microservices



Presenter – Rod Davison

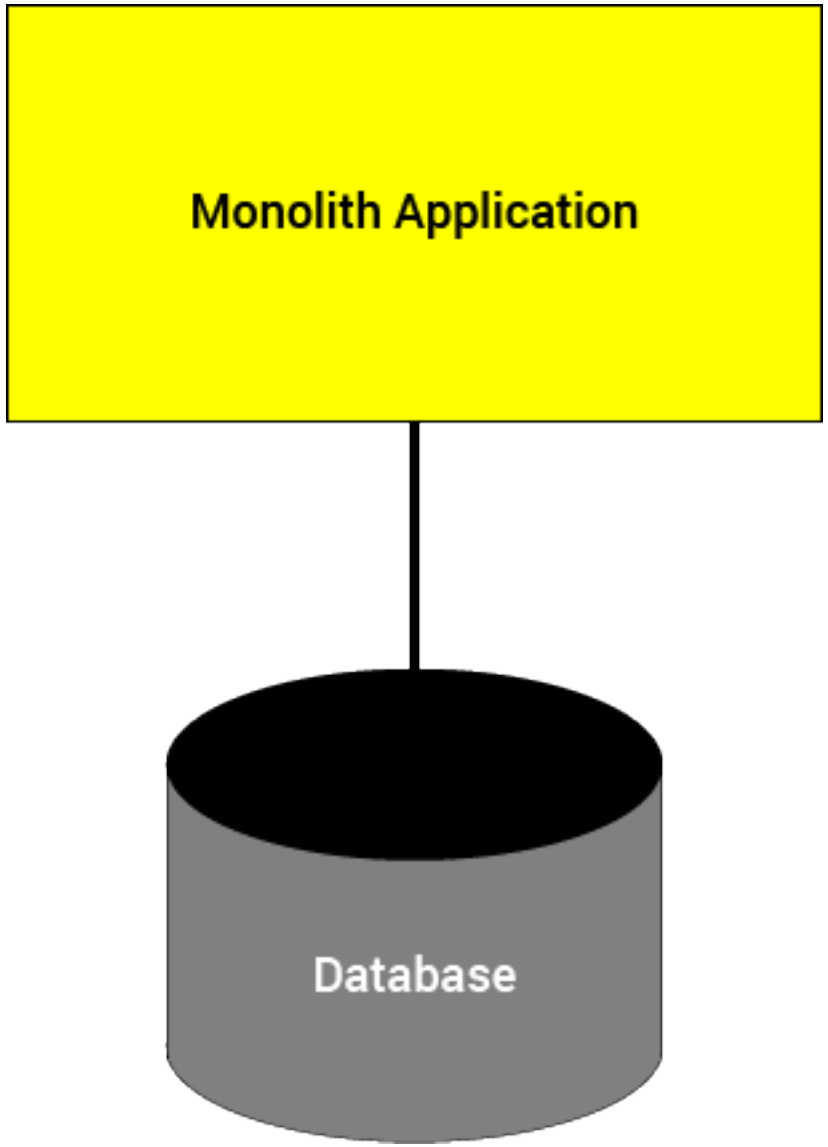
- 50+ years experience
  - Academia (math, linguistics, cognitive science)
  - Artificial Intelligence R&D
  - Software Development
  - Data Analytics – Social Research
  - Market Research
  - Project Manager
  - Quality and Testing
  - Business Analysis
  - Consulting and Training

Introduction

- Microservices are a software architectural pattern
- There are specific challenges that they solve effectively
  - These are generally issues around scaling in both the development and operations
  - Microservices are not a “magic” bullet, sometime they are the wrong choice
- Deploying microservices requires
  - Supporting technologies in operations – for example: Kubernetes, Kafka, Docker
  - Analysis and design techniques for building a component based software architecture
  - Code and application design techniques to make code “microservices ready”
  - Production techniques to support the successful deployment of a microservice

Monolith Application

- Characterized by a single code base
  - May be modular at the programming language level
- Integrated with a single database
  - All code uses a common schema
- “Monolith” means
  - If a change to the code base or to the data schema
  - Then the entire application needs to be redeployed

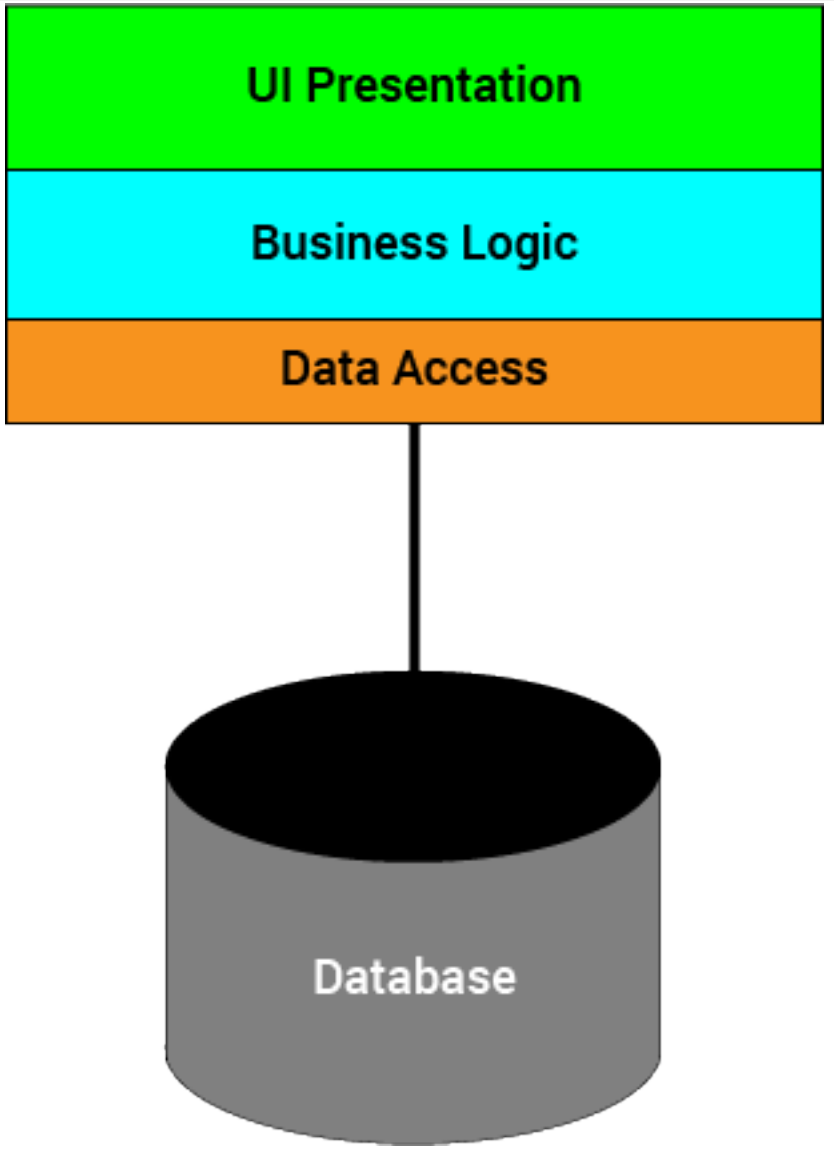


Business Analogy

- Start-up businesses are monoliths
- There are a few people who do everything
- The enterprise is small enough that this model works
  - It’s actually counterproductive to have a highly structured departmental organization with just a few employees
- Early versions of applications are similar
  - Simple enough that all of the code is manageable
  - Single code base for the whole app
  - Flat or minimal architectural structure

Modular Monolith Application

- Code is modularized
- Organized along job descriptions
  - Front end dev has their modules
  - Programmers have their modules
  - Data engineers have their modules
- Shows up historically as a n-tier architecture

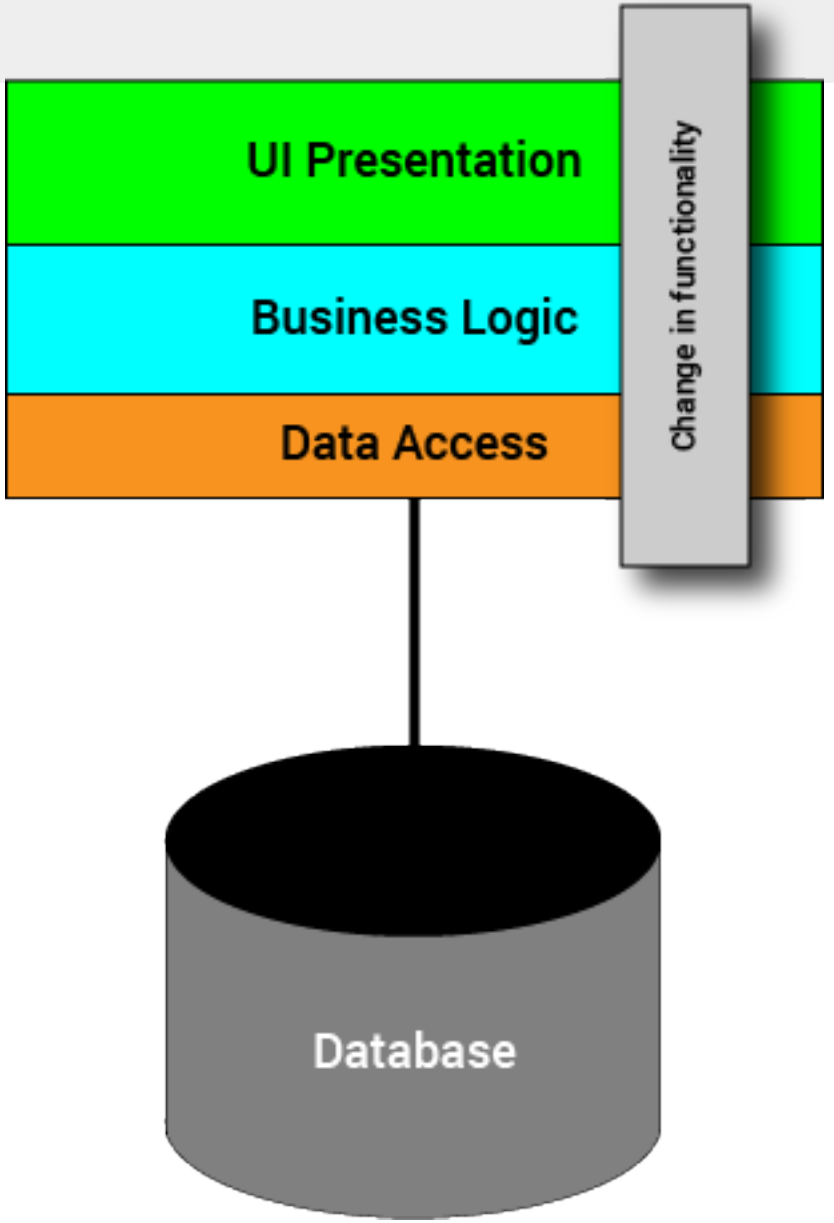


Business Analogy

- As start-up grows, it adds more people
- The original organization starts to become ineffective
  - The entrepreneurial “all hands-on deck” model doesn’t scale well
  - Processes become chaotic
  - Difficult to manage
  - Productivity stalls
- At some point the business must modularize
  - Usually by creating specialized departments
  - Accounting, sales, HR, etc.

Modular Monolith Application

- The application is still a monolith
- A change in functionality
  - Affects each layer because related changes have to be made in each layer
- Interacts with the data model
  - The data model may constrain what changes can be made
  - Changing the data model might break other parts of the app
- The modules and the database often show high coupling
  - Due to how the modules are defined



Pros and Cons of Monoliths

Pros

- Simple to develop
- Simple to test
- Simple to deploy
- Simple to scale horizontally
  - Run multiple copies behind a load balancer
- Although persistence is problematic when using horizontal scaling

Cons

- The size of the application can slow down the start-up time
- The entire application must be redeployed on each update
- Monolithic applications can also be challenging to scale vertically
- Reliability – easy to crash the whole application
- Difficult to migrate to new technologies

Legacy Monolithic Issues

- Codebase is enormous
  - Little or no documentation
  - Coupled to legacy technology
- Eg. Internal Revenue Service
  - Running 1960s vintage code
  - Application highly complex
  - Attempts to replace it have failed
  - Over $15 billion so far
- The IRS is not unique
  - Migrating legacy monolith to a modern monolith is not an option

Scaling in Systems

- Scaling is an increase in size or quantity along some dimension
- Can take place in the development or operations space
- Development scaling
  - Increase in complexity, functionality or volume of code
  - These dimensions are often related
  - Business analogy is a company increasing the range of services and products they offer or expanding into different markets (like a Canadian company expanding into Europe)
- Operational scaling
  - Increase in the amount of activity of a system
  - Throughput, load, transaction time, simultaneous users, etc
- Traditional monoliths tend to be scalable only to a limited degree
  - There is a certain level of complexity after which they become unmaintainable

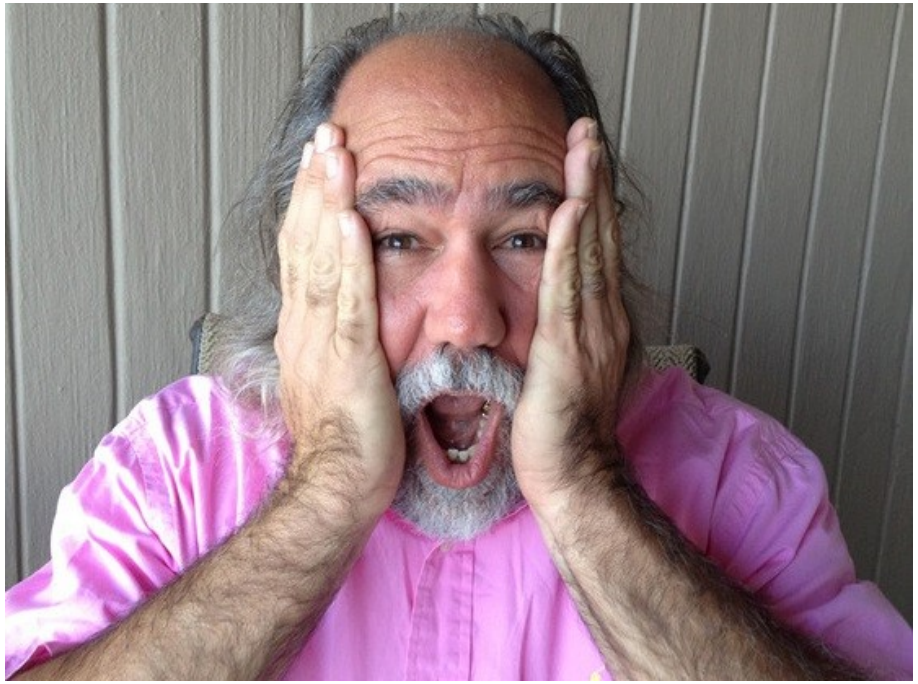
Mission Critical Industrial Strength Software

Mission critical software tends to have a long lifespan, and over time, many users come to depend on their proper functioning. In fact, the organization becomes so dependent on the software that it can no longer function in its absence. At this point, we can say the software has become industrial-strength.

The distinguishing characteristic of industrial strength software is that it is intensely difficult, if not impossible, for the individual developer to comprehend all of the subtleties of its design.

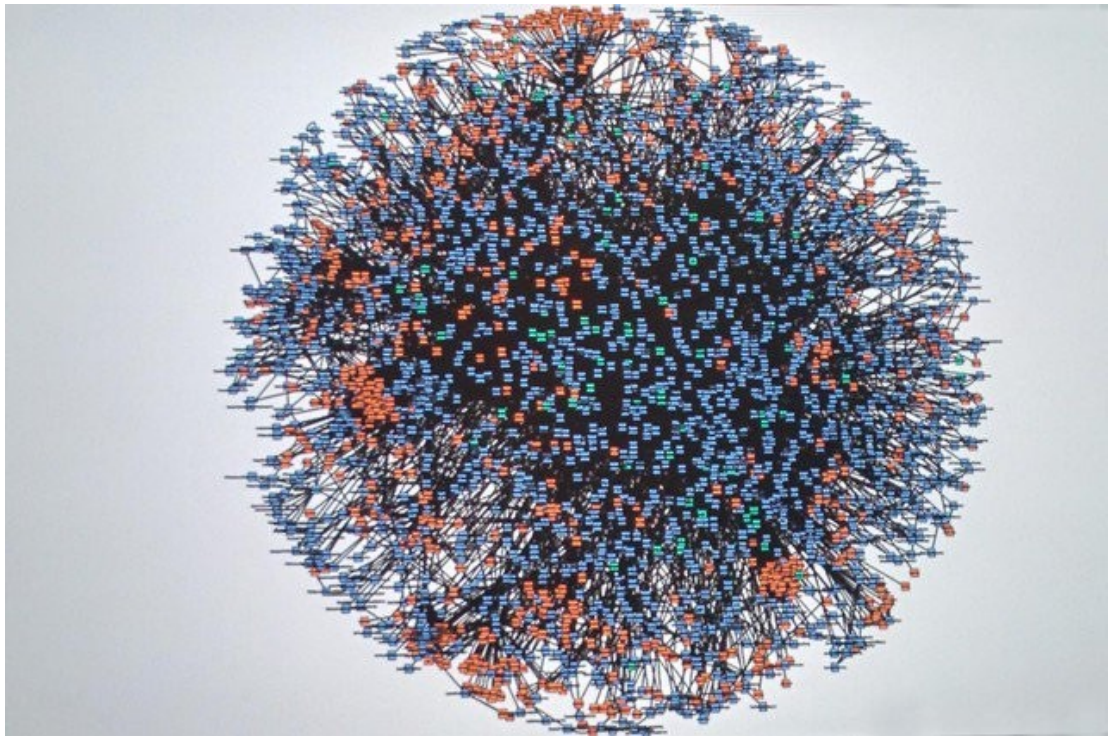
Stated in blunt terms, the complexity of such systems exceeds the human intellectual capacity. Alas, this complexity we speak of seems to be an essential property of all large software sys--tems. By 'essential' we mean that we may master this complex-ity, but we can never make it go away.

Grady Booch



Operational Complexity

- Modern applications have to deal with
  - Petabytes of streaming data
  - Billions of transaction
  - Mission critical fault tolerance
- Non-functional requirements
  - Throughput, response time, loading, stress
  - Disaster recovery, transaction time
- Results in a “death star” architecture
  - Image: Amazon in 2008
  - The complexity of the operational architecture is now industrial strength



IT Failures and Complexity

The United States is losing almost as much money per year to IT failure as it did to the [2008] financial meltdown. However the financial meltdown was presumably a onetime affair. The cost of IT failure is paid year after year, with no end in sight. These numbers are bad enough, but the news gets worse. According to the 2009 US Budget [02], the failure rate is increasing at the rate of around 15% per year.

Is there a primary cause of these IT failures? If so, what is it?.... The almost certain culprit is complexity.... Complexity seems to track nicely to system failure.

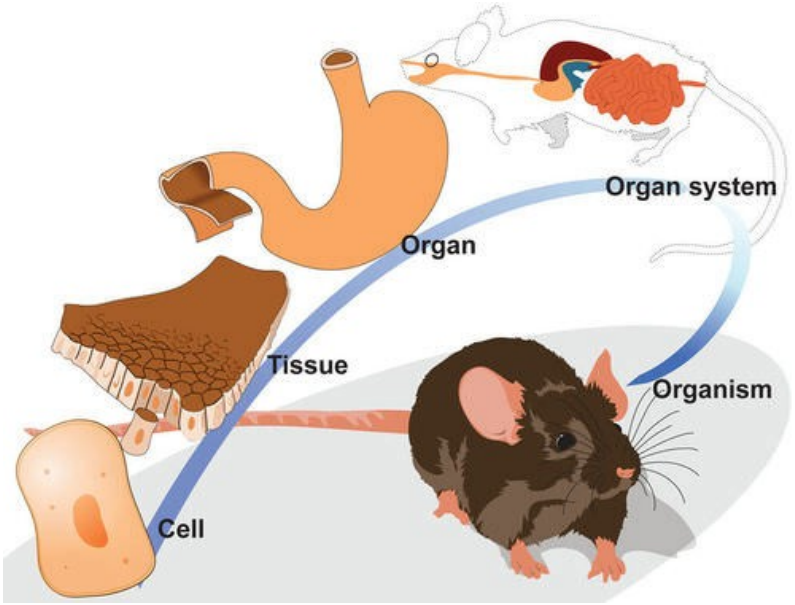
Once we understand how complex some of our systems are, we understand why they have such high failure rates.

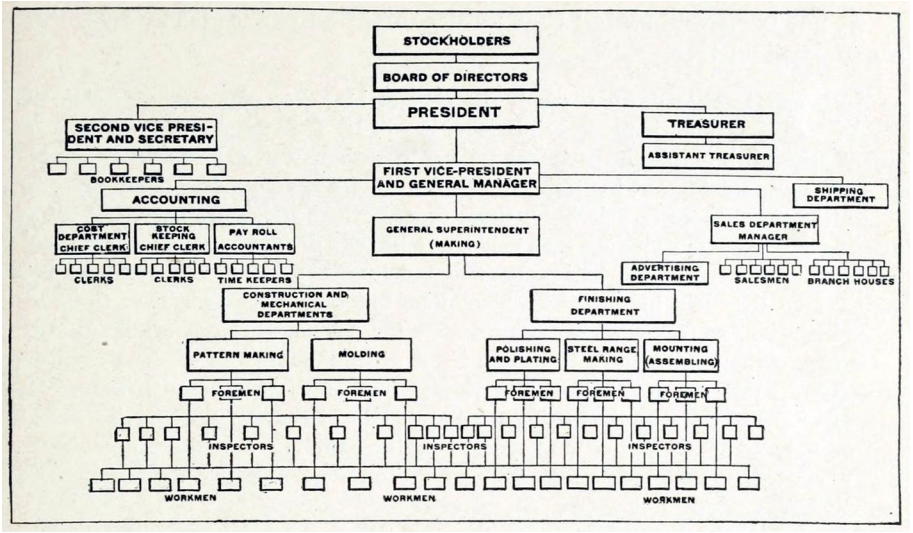
We are not good at designing highly complex systems. That is the bad news. But we are very good at architecting simple systems. So all we need is a process for making the systems simple in the first place.

Roger Sessions

Natural Systems Organization

- Naturally occurring systems scale successfully
  - Biological processes, social organization etc.
  - They all show similarities in organization regardless of domain
- Natural occurring systems tend to be
  - Recursive in structure
  - Hierarchical or layered
  - Modular at each layer
  - Loosely coupled and highly cohesive





Operational Complexity in Practice

- In production, systems have to scale operationally
- Consider a diner
  - During the lunch and dinner hour rush, the number of servers and cooks have to scale up
  - There has to be clear boundaries on what each role does
  - During non-peak hours, the amount of staff can be scaled down

Operational Complexity in Practice

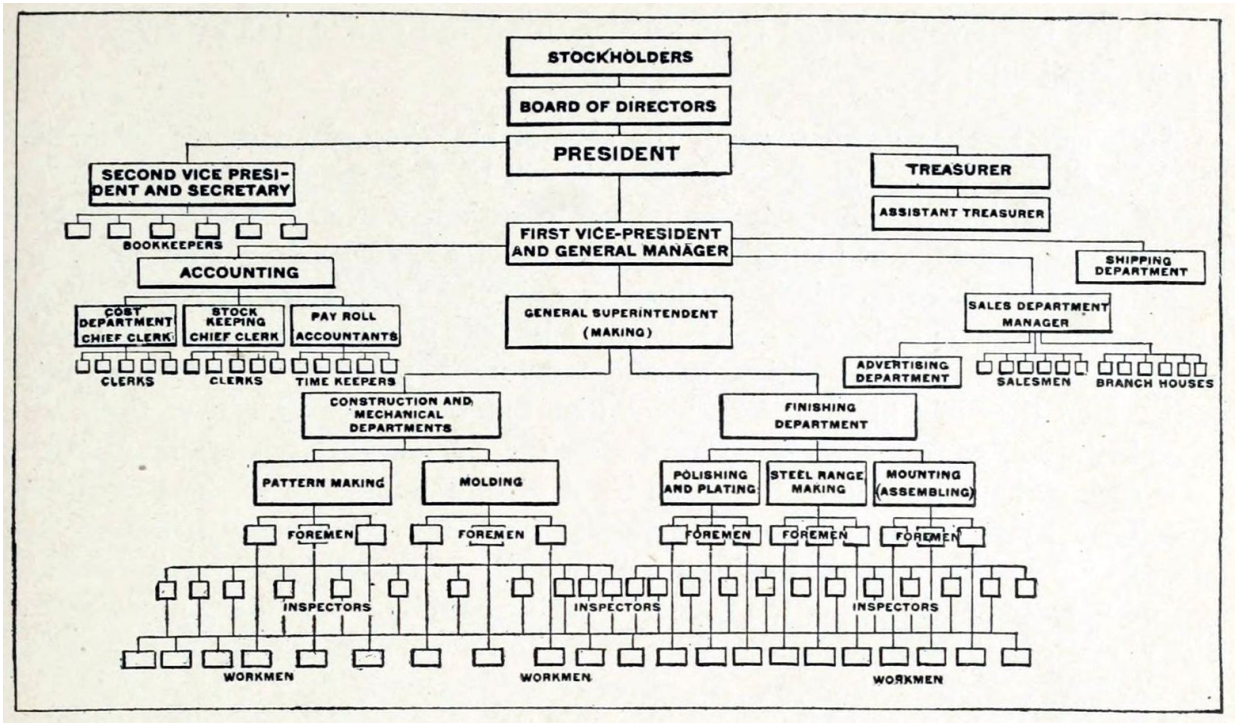
- Operations are specialized
- Tasks are broken down into sub-tasks, and sub-tasks broken down into sub-sub-tasks, and so on
- At some point
  - Specialized systems or agents perform an individual sub-task
  - These are all coordinated
- For example, the specialized positions on a sports team

Complex Systems

Frequently, complexity takes the form of a hierarchy, whereby a complex system is composed of interrelated subsystems that have in turn their own subsystems, and so on, until some lowest level of elementary components is reached

Courtois

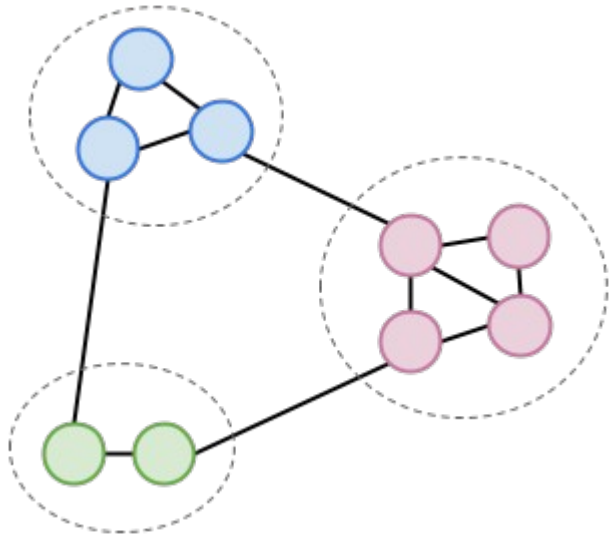
On Time and Space Decomposition of Complex Structures
Communications of the ACM, 1985, 28(6)



Complex Systems

Intra-component linkages are generally stronger than inter-component linkages. This fact has the effect of separating the high frequency dynamics of the components – involving the the internal structure of the components – from the low frequency dynamics – involving the interaction among components

Simeon



Complex Systems

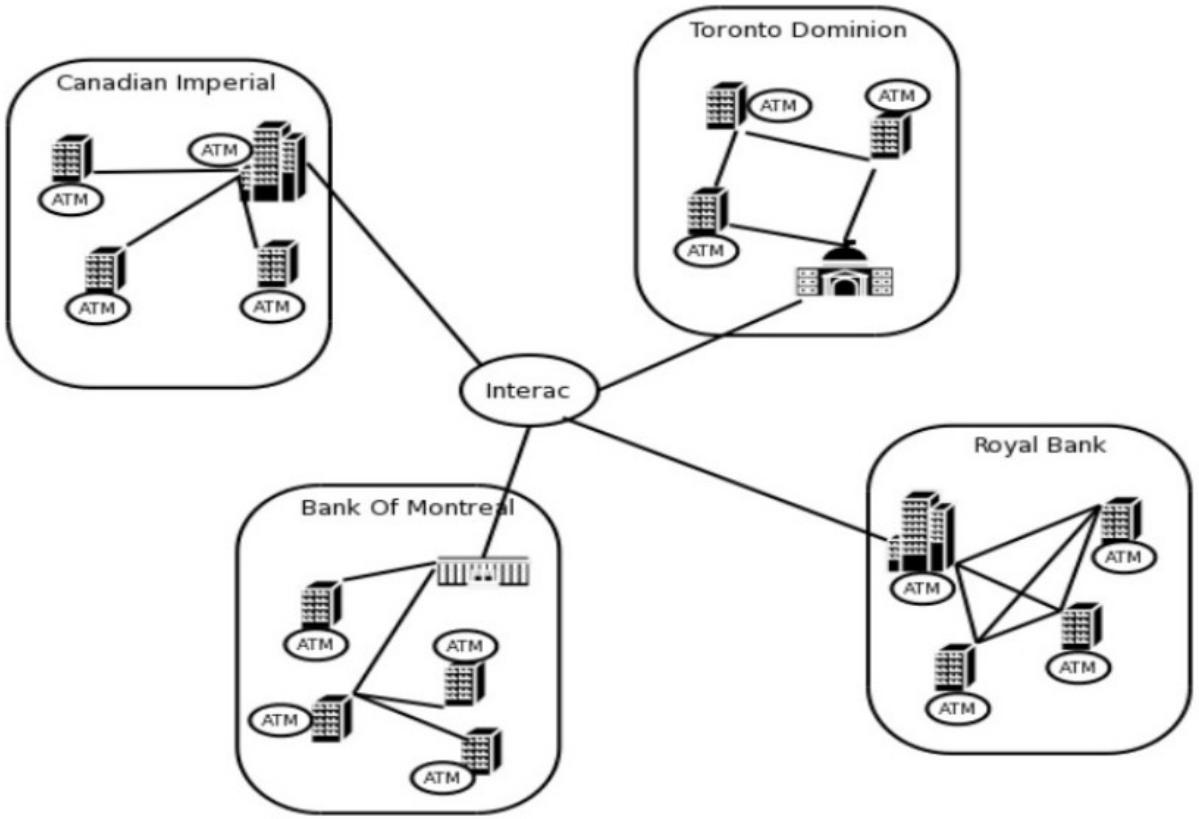
Hierarchical systems are usually composed of only a few different kinds of sub-systems in various combinations and arrangements

Simeon

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 can never be patched up to make it work. You have to start over, beginning with a simple working system.

John Gall

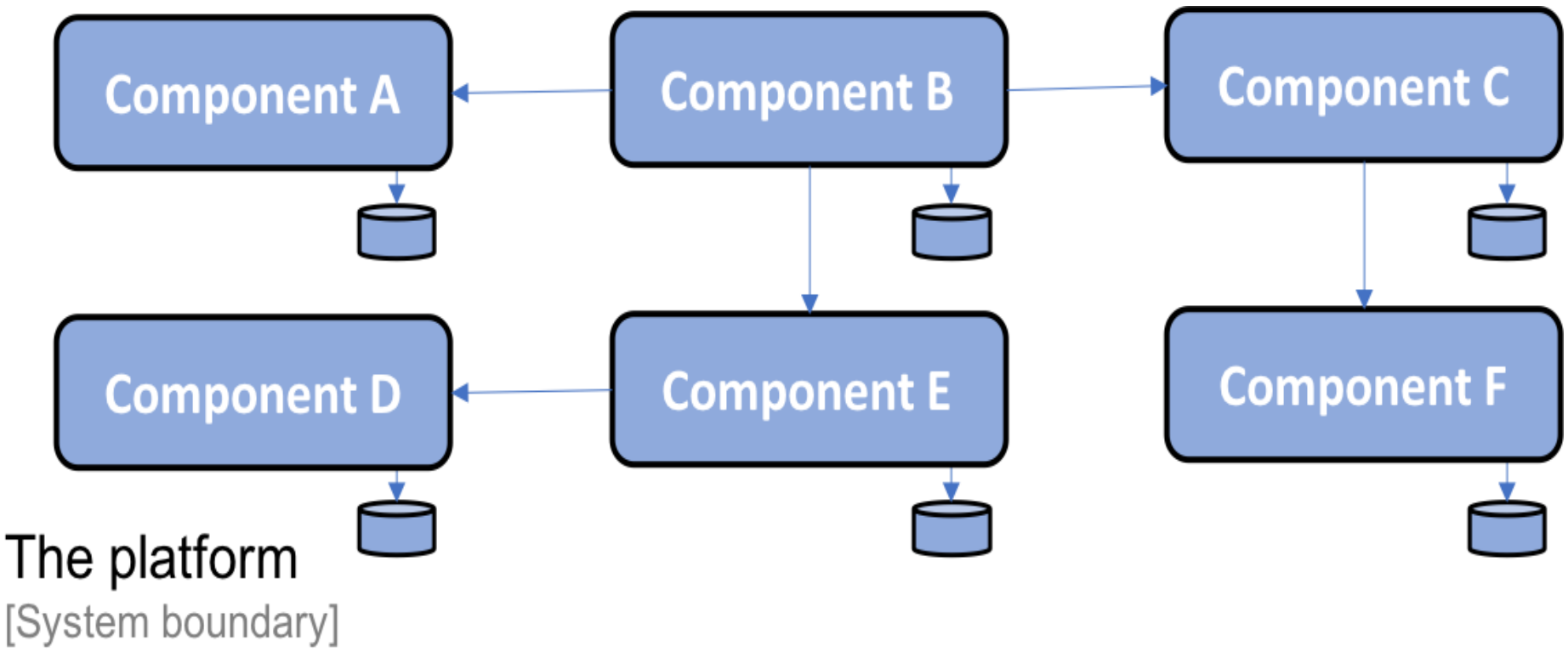
Systemantics: How Sytems Really Work an How They Fail
1975



Microservices

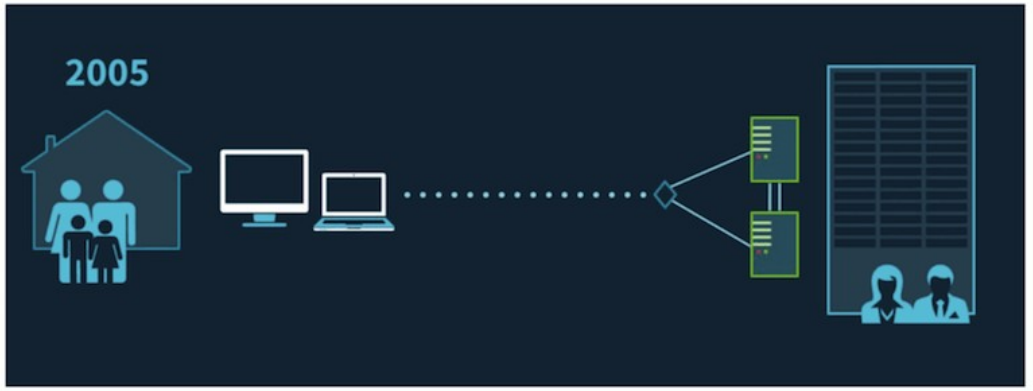
“The Microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.”

Microservices

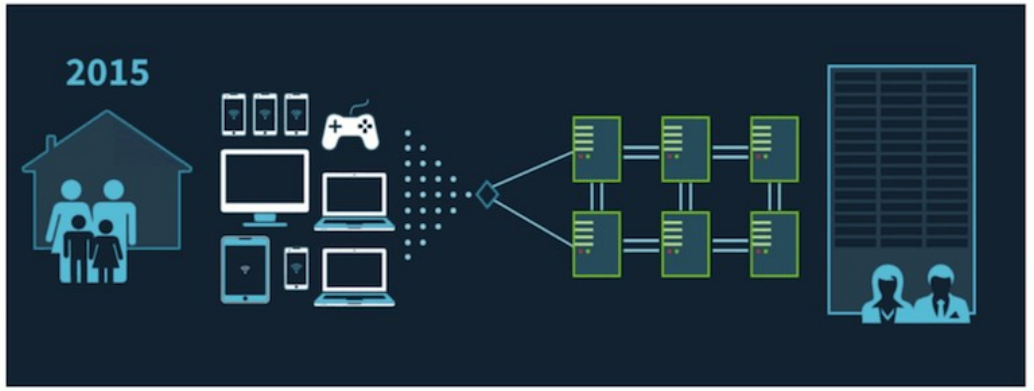


Microservices

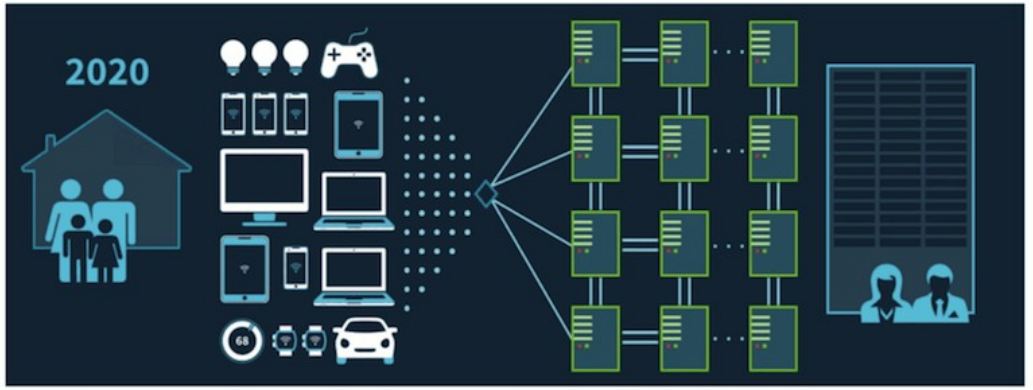
2005 ARCHITECTURE



2015 ARCHITECTURE



2020 ARCHITECTURE



THE WORLD BY 2020

» 4 billion connected people

» 25+ million apps

» 25+ billion embedded systems

» 40 zettabytes (40 trillion gigabytes)

» 5,200 GB of data for every person on Earth

Microservices Drivers

Modeling Microservices

- A core principle of microservices is that they model a business domain or a domain of activity
- These domains are often complex and wide ranging in scope
- Very often, they have evolved a domain structure that is
  - A hierarchy of sub-domain layers
  - The structure is recursive
  - Each layer is made up of modular components
  - The components work together by sending well-defined messages
  - Messages are sent through interfaces
  - Components are cohesive and loosely coupled
- This is a sort of idealized version of services “in the wild”
  - There are many factors that can make this organization disfunctional

In Real Life

- A hospital provides a health care service
- Made up of individual microservices
  - Laboratory
  - X-Ray
  - Pharmacy
  - Medical Staffing
- Each microservice:
  - Specializes in a specific domain activity
  - Only that microservice performs that domain activity (the pharmacy doesn't do x-rays for example)
  - Each microservice operates autonomously

In Real Life

- Microservices request services from each other
  - They do not need to know the internal workings of the other microservice
- Requests are made through interfaces
  - Called APIs in software engineering
  - These are often paper based in the real world
  - Requisitions and official forms and paperwork used to make requests of a service
  - Response to a request is often a report of some kind
- Microservices make sense intuitively

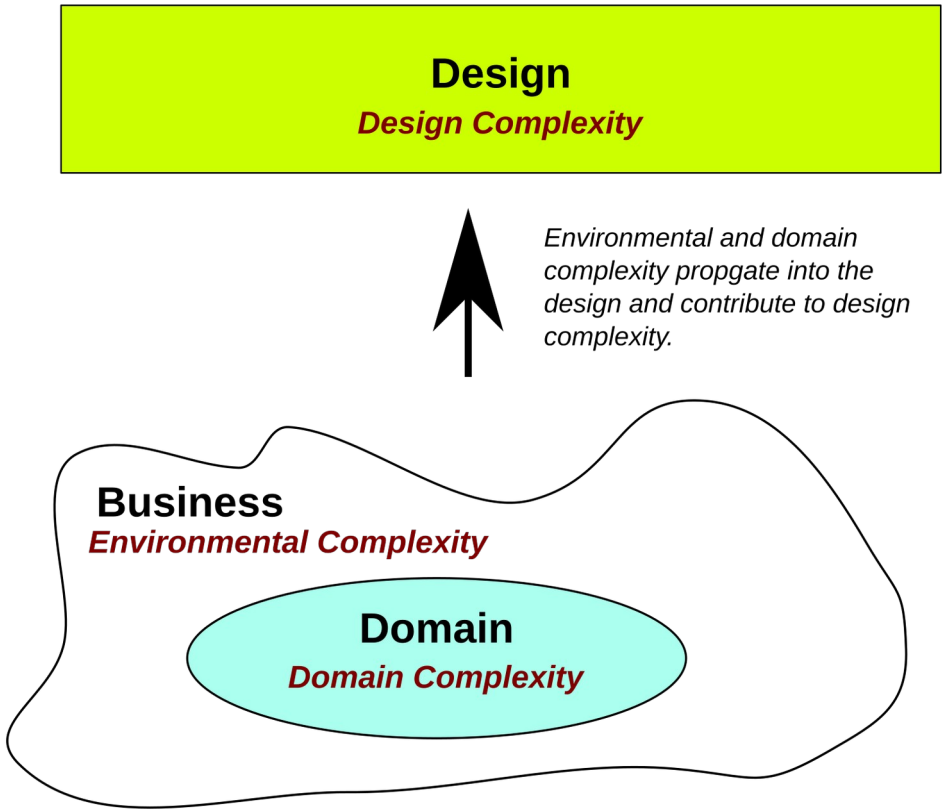
COVID-19 VIRUS LABORATORY TEST REQUEST FORM[1]

Submitter information			
NAME OF SUBMITTING HOSPITAL, LABORATORY, or OTHER FACILITY*			
Physician			
Address			
Phone number			
Case definition:[2]	☐ Suspected case ☐ Probable case		
Patient info			
First name		Last name	
Patient ID number		Date of Birth	Age:
Address		Sex	☐ Male ☐ Female ☐ Unknown
Phone number			
Specimen information			
Type	☐ Nasopharyngeal and oropharyngeal swab ☐ Bronchoalveolar lavage ☐ Endotracheal aspirate ☐ Nasopharyngeal aspirate ☐ Nasal wash ☐ Sputum ☐ Lung tissue ☐ Serum ☐ Whole blood ☐ Urine ☐ Stool ☐ Other:		
All specimens collected should be regarded as potentially infectious and you must contact the reference laboratory before sending samples. All samples must be sent in accordance with category B transport requirements.			
Please tick the box if your clinical sample is post mortem ☐			
Date of collection		Time of collection	
Priority status			
Clinical details			
Date of symptom onset:			
Has the patient had a recent history of travelling to an affected area?	☐ Yes ☐ No	Country Return date	
Has the patient had contact with a confirmed case?		☐ Yes ☐ No ☐ Unknown ☐ Other exposure:	
Additional Comments			

The Complexity Problem

- There are three kinds of complexity we have always been concerned with
  - **Domain complexity** - trying to automate a domain that is inherently complex
  - **Design complexity** - Designs that are not elegant (simple and effective)
  - **Environmental complexity** – Resulting from a disorganized and poorly functioning business
- We can eliminate a lot of design complexity with good design and engineering practices
- We cannot make domain complexity go away, we can only manage it

Controlling complexity is the essence of computer programming. Brian Kernigan

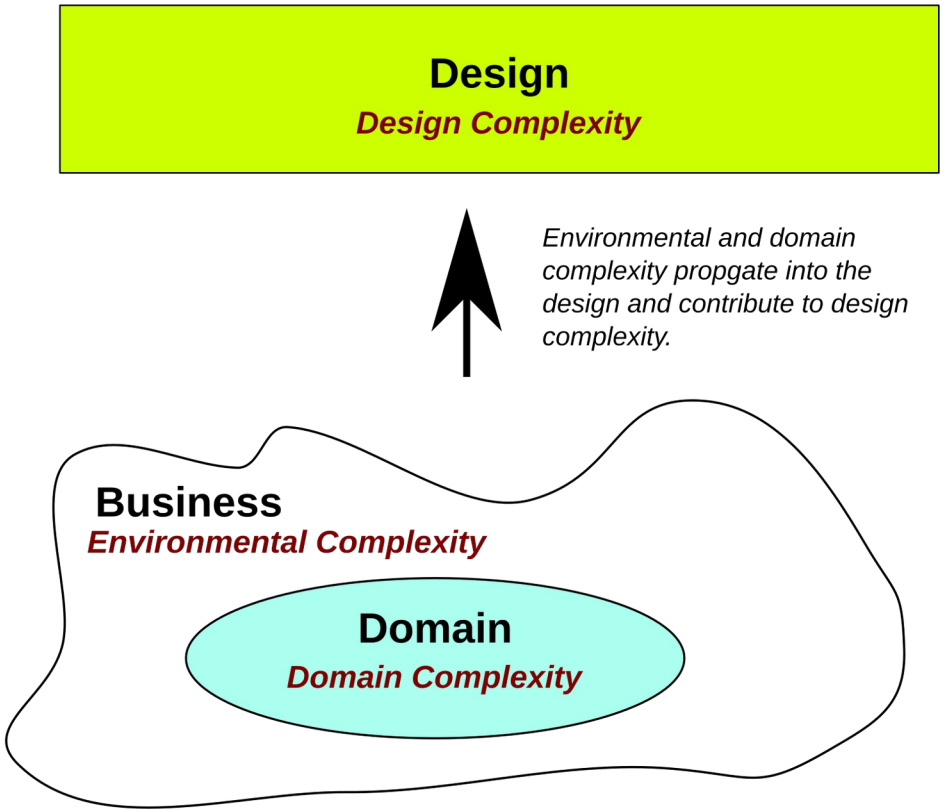


Propagation of Complexity

- Not properly managed, environmental and domain complexity directly propagate into design complexity
- We should not confuse environmental complexity with domain complexity
  - We cannot deal with it, so we have to ignore it in understanding the domain

The most important single aspect of software development is to be clear about what you are trying to build.
Edsger Dijkstra

First, solve the problem. Then, write the code
Donald Knuth



Microservices Essential Principles

- Single Responsibility
- Discrete
- Carries its own data
- Transportable
- Ephemeral

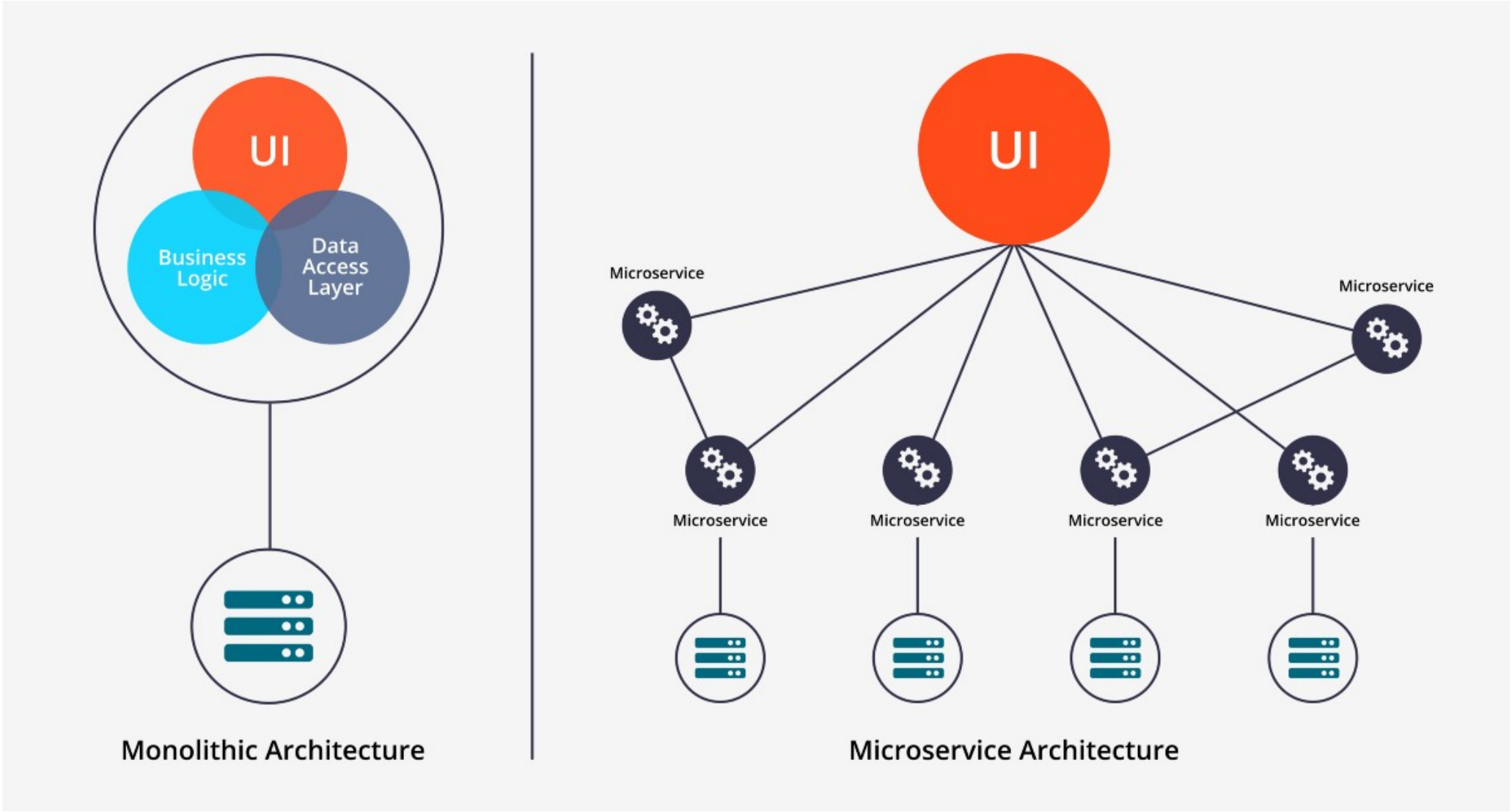
Microservices Core Concepts

- Microservices are small, independent, composable services
- Each service can be accessed by way of a well-known API format
  - Like REST, GraphQL, gRPC or in response to some event notifications
- Breaks large business processes into basic cohesive components
  - These basic process actions are implemented as microservices
  - The business process is executed by making calls to these microservices
- Each microservice provides one cohesive service
  - Microservices do not exist in isolation
  - They are part of a larger organization framework

Microservices Core Concepts

- Microservices coordinate with other microservices to accomplish the tasks normally handled by a monolithic application
- Microservices communicate with each other synchronously or asynchronously
- Microservices make application components easier to develop and maintain
- BUT A LOT HARDER TO MANAGE

Monolith to Microservice



Characteristics of Microservice Deployments

- Light-weight, independent, and loosely-coupled
- Each has its own codebase
- Responsible for a single unit of business functionality
- Uses the best technology stack for its use cases
- Has its own DevOps plan for test, release, deploy, scale, integrate, and maintain independent of other services
- Deployed in a self-contained environment
- Communicates with other services by using well-defined APIs and simple protocols like REST over HTTP
- Responsible for persisting its own data and keeping external state

Business Drivers

- Business Agility
  - Speed of change is faster with a more modular architecture
  - React quicker to feature and enhancement requirements
  - Easy integration with new technology, processes (Mobile, Cloud, Continuous DevOps)
- Composability
  - Allows for reuse of capability and functionality
  - Allows for integration with other internal and external services
  - Reduces technical debt and replication
- Robustness and Migration
  - A single microservice failures does not bring down an application
  - The business functionality is always available
  - Updated functionality can be rolled out in a controlled and safe manner
  - Smaller components reduces risk exposure

Business Drivers

- Scalability
  - Services can scale up to handle peak loads, or down to save costs
- Enable Polyglot Development
  - Different technologies can be used for different services
  - No lock-in to a single technology across the whole business

Pros and Cons of Microservices

Pros

- Easy to develop individually
- Easy to understand individually
- Easy to deploy idividually
- Easy to monitor each service
- Flexible Release Schedule
- Use standard APIs (JSON, XML)

Cons

- Requires retooling
- Requires more deployments
- Requires translation (JSON, XML)
- Requires more monitoring
- Operations configuration can be very complex
- System can be very complex

End of Module