SE 577
Software Architecture

Intro to Software Architecture

What is Software Architecture?

- The fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.

ANSI/IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems.

What is Software Architecture? (cont'd)

A software system architecture comprises:

- A collection of software and system components, connections, and constraints.
- A collection of system stakeholders' need statements.
- A rationale which demonstrates that the components, connections, and constraints define a system that, if implemented, would satisfy the collection of system stakeholders' need statements.

Boehm, et al., 1995

What is Software Architecture? (cont'd)

- An architecture is the set of significant decisions about the organization of a software system, the selection of the structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaborations among those elements, the composition of these structural and behavioral elements into progressively larger subsystems, and the architectural style that guides this organization---these elements and their interfaces, their collaborations, and their composition

P. Kruchten: The Rational Unified Process.

What is Software Architecture? (cont'd)

- As the size and complexity of software systems increases, the design problem goes beyond the algorithms and data structures of the computation: designing and specifying the overall system structure emerges as a new kind of problem.

  Structural issues include gross organization and global control structure; protocols for communication, synchronization, and data access; assignment of functionality to design elements; physical distribution; composition of design elements; scaling and performance; and selection among design alternatives.

  This is the software architecture level of design.

  Garlan and Shaw, 1999

Every system has an architecture, regardless if it was planned or not planned

- Every system comprises elements and relations among them to support some type of reasoning.
- But the architecture may not be known to anyone.
  - Perhaps all of the people who designed the system are long gone
  - Perhaps the documentation has vanished (or was never produced)
  - Perhaps the source code has been lost (or was never delivered)
- An architecture can exist independently of its description or specification.
- Documentation is critical.

But why do we need architecture?

Purpose Built Computers

1940s

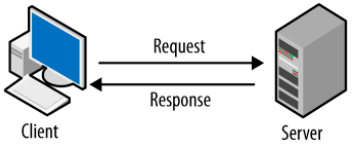
Batch Business Processing on Mainframes

1960s

monolythic applications on a desktop computer

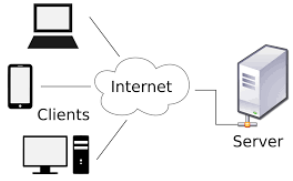
1980s

Basic Client/Server



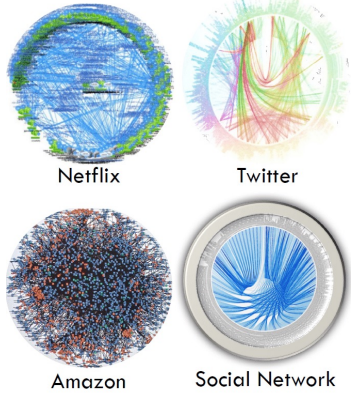
1990s

Basic Web Applications



2000s

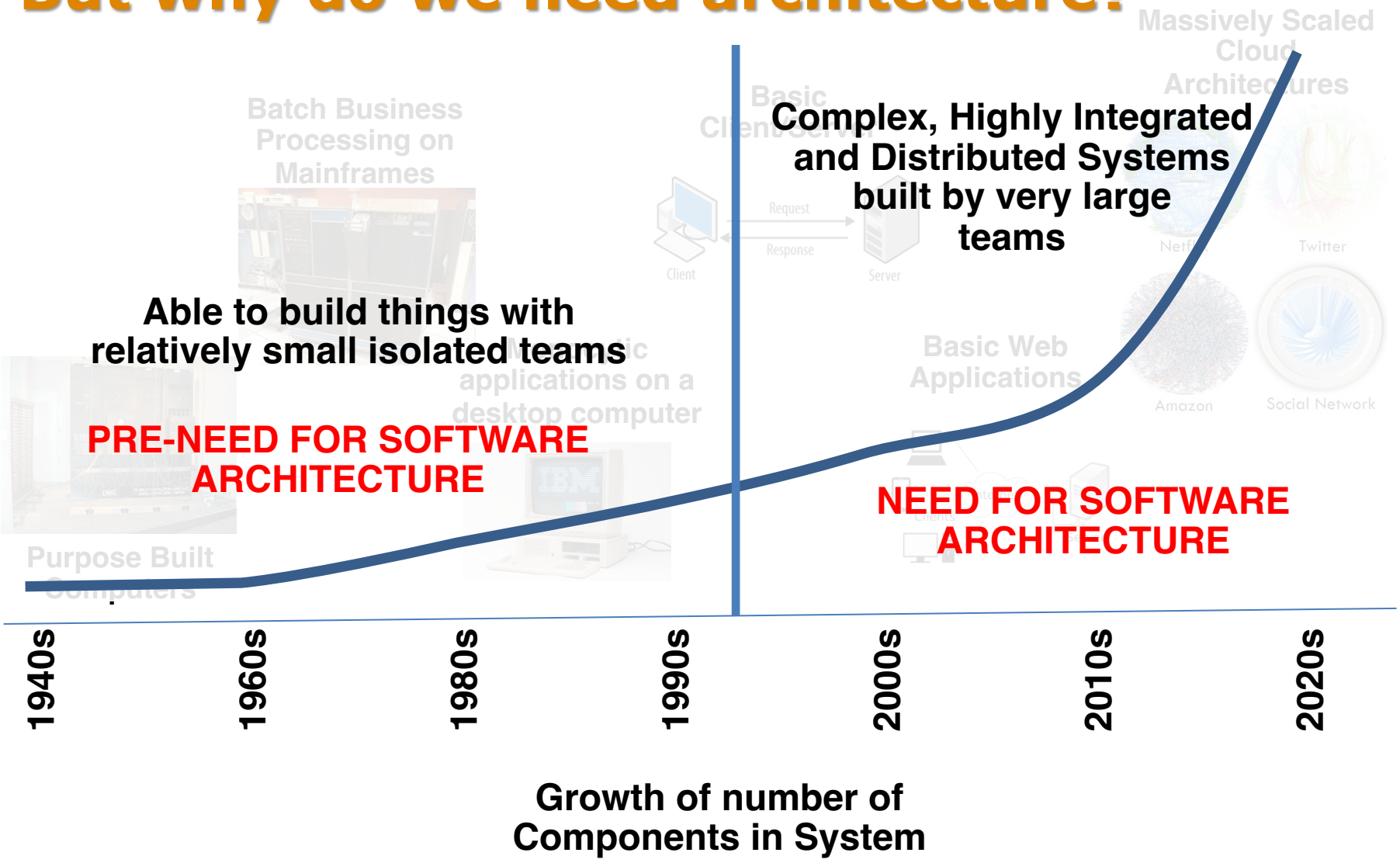
Massively Scaled Cloud Architectures



2010s

2020s

But why do we need architecture?



But why do we need architecture?

An Introduction to Software Architecture

David Garlan and Mary Shaw
January 1994

CMU-CS-94-166

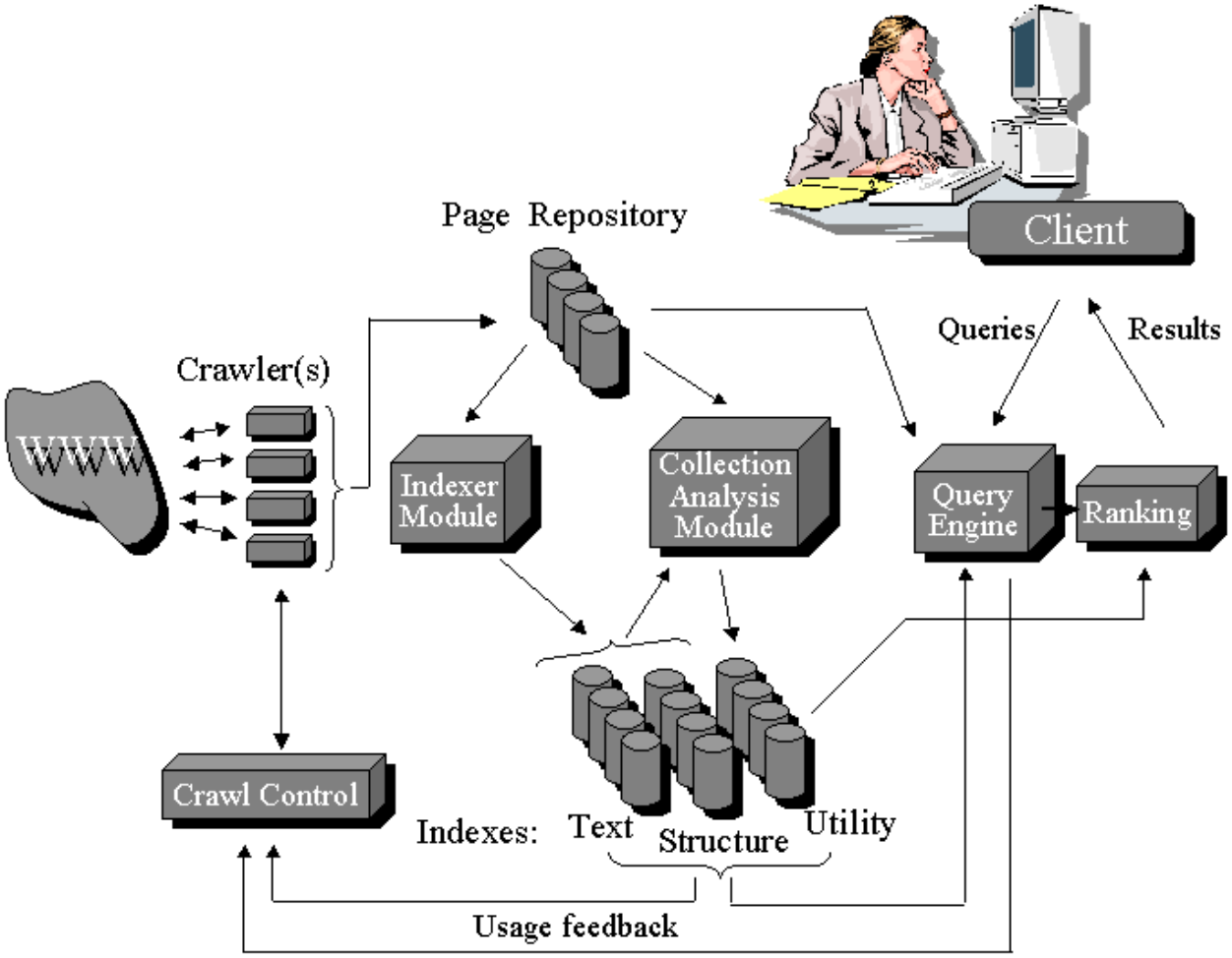
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Also published as "An Introduction to Software Architecture," *Advances in Software Engineering and Knowledge Engineering, Volume I,* edited by V.Ambriola and G.Tortora, World Scientific Publishing Company, New Jersey, 1993.

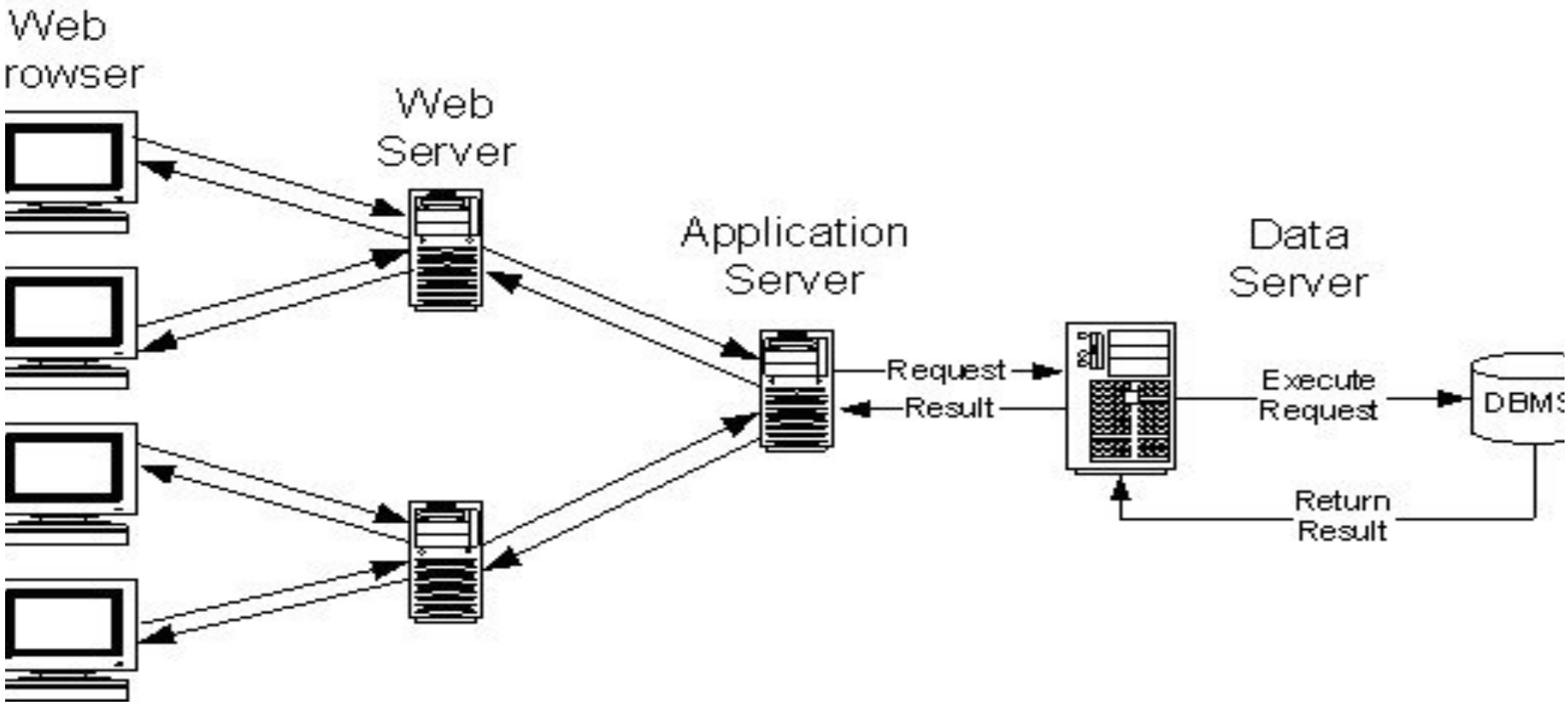
Also appears as CMU Software Engineering Institute Technical Report CMU/SEI-94-TR-21, ESC-TR-94-21.

The Seminal Paper Describing the Need for Software Architecture

General Search Engine Architecture (Arvind)



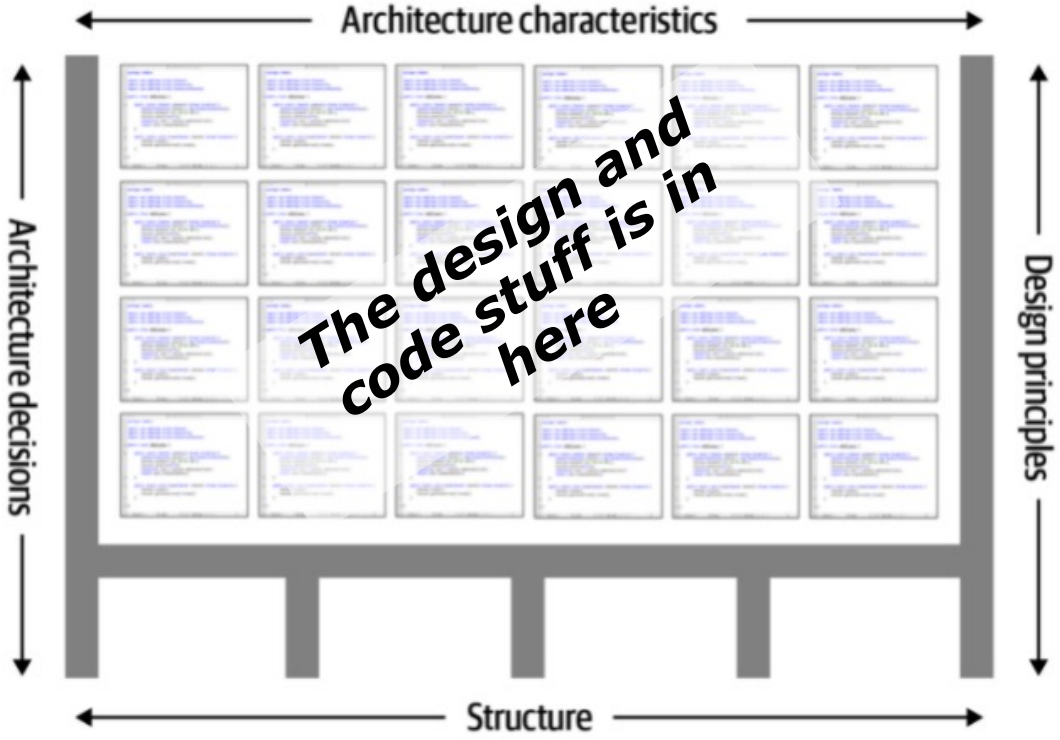
Multi-tier Architecture



Software Architecture as a Design Activity

- It's about software design
  - All architecture is software design, but not all design is software architecture
  - Part of the design process
- Simply, architecture focuses on issues that will be difficult/impossible to change once the system is built
  - Quality attributes like security, performance
  - Non-functional requirements like cost, deployment hardware
  - More on these later in this course
- Generally architecture is about the effective management of constraints on the to be built system

Software Architecture in the Context of Design



Architecture is key to constraining and structuring the design of systems

Ref: Fundamentals of Software Architecture – N. Ford & M. Richards

Architecture is Expressed using Models

- Software architecture emerged when it was becoming obvious that designs are too complicated to develop from scratch (circa early 1990s)
- Good designs tend to be build using models…
  - 1) Abstract different views of the system
  - 2) Build models using precise notations (e.g., UML)
  - 3) Verify that the models satisfy the requirements
  - 4) Gradually add details to transform the models into the design
- And such models can be derived from proven/established architecture patterns – more on this later

Architecture Abstractions

There are different techniques to perform software architecture analysis…

- Reference Architecture – focus is on the broad domain
- Solution Architecture – focus on the solution space
- Software Architecture – focus on the key technical abstractions

As well as a diversity of use cases where this information can be useful…

- New product definition
- Monitoring design quality (trends)
- Redocumenting and/or extending the lifespan of existing systems

The "art" of architecture is to pick the correct abstractions based on the desired objectives or outcomes

Architecture Framing

When thinking about a reference-, solution-, or software architecture focus on:

- Who is the constituent – aka, who am I trying to talk to or influence
- What do I want them to understand – aka, why am I talking about architecture
- What are the benefits, constraints, or tradeoffs associated with the architecture I am describing

If you cant answer any/all of the above questions the impact of your architectural work will likely be sub-optimal

We will look at documentation approaches later, but I'm not necessarily a big fan of standard notations – more on that later.

Architecture Done Poorly …

Get it Right the First Time !

Why Study Architecture?

- It is an integral aspect of structuring software so that it can be built, understood and maintained:
  - Client / Server
  - Layered systems
  - Cloud based systems
  - etc
- Architectural choices are critical in determining SW quality:
  - Performance
  - Scalability
  - Reuse
  - … and other non-functional properties

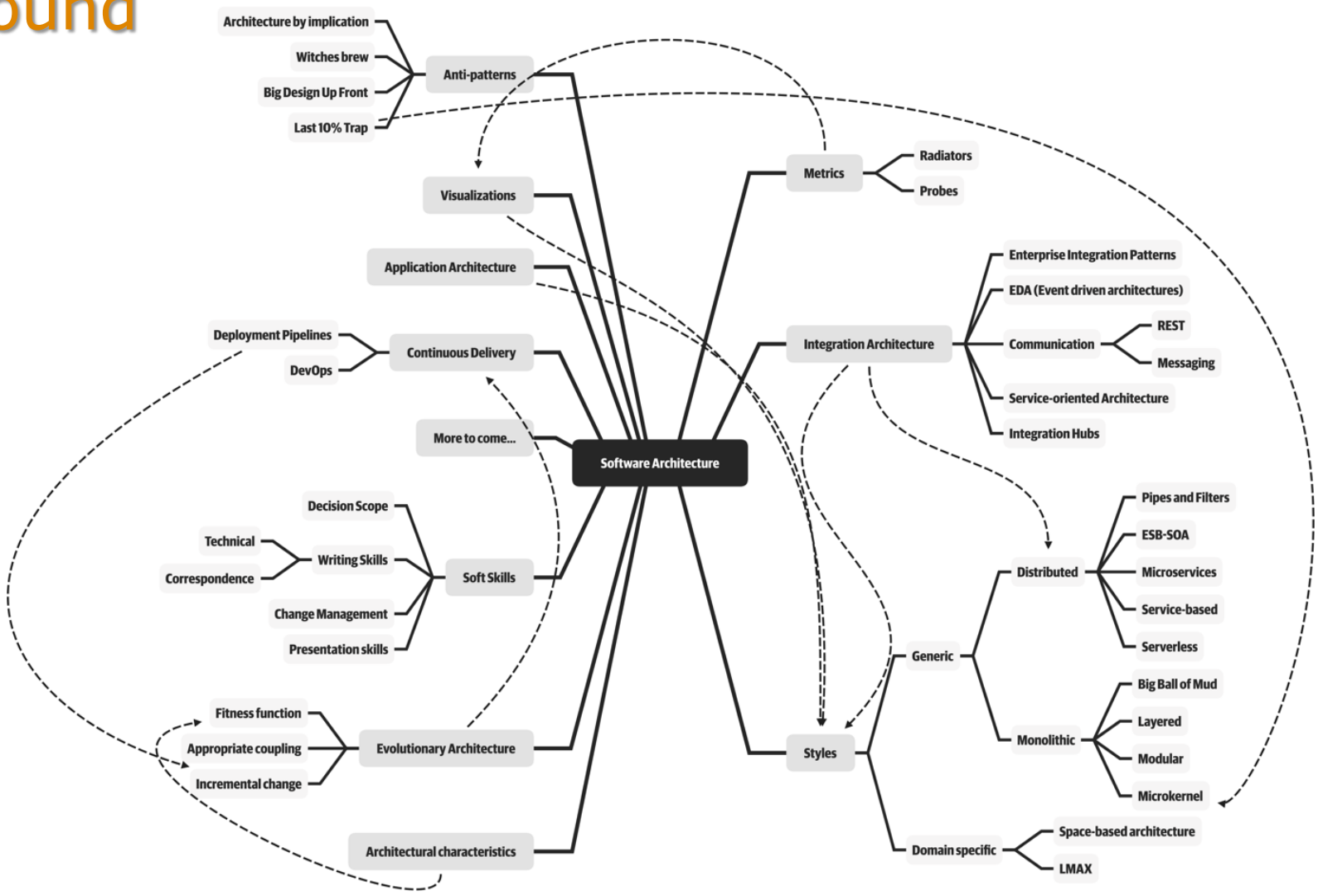
Why Study Architecture? (cont'd)

- As a architect, you need to:
  - Effectively manage complex tradeoffs to meet stakeholder objectives
  - Layout, socialize and influence your decisions.
  - Communicate with your team and convince your customer / boss of their merit.
  - Reason about the properties of the overall system in an abstract and convenient way.

Why Study Architecture? (cont'd)

- Certain "canonical" architectural designs provide valid blueprints for a lot of systems.
  - Architectural styles
- Learn to recognize applicability and characteristics of styles.
- Learn to use technologies that induce and support those styles.

Software Architects need to cover a lot of ground



What does a Software Architect do?

- Interact with stakeholders to make sure their needs are being met.
- Craft the right architecture to solve the problem at hand.
  - Make sure the right modeling is being done, to know that qualities like performance are going to be met.
  - Make sure that the architecture is not only the right one for operations, but also for deployment and sustainment.
- Document, influence, and communicate it.
- Give input as needed to issues like tool and environment selection.
- Manage risk identification and risk mitigation strategies associated with the architecture, and architecture decisions.
- Work closely with engineering teams, ensuring architecture guidance is relevant, is followed, and is adding value.

Expectations of a software architect

- Be able to create abstractions and models that capture key constraints and tradeoff decisions
- Continually analyze and update architecture as new things are learned
- Keep current with the latest trends
- Ensure compliance with decisions
- Have diverse exposure and experience
- Have business domain knowledge
- Possess interpersonal skills (required for influencing)
- Understand and navigate politics
- Effectively demonstrate credibility to diverse audiences

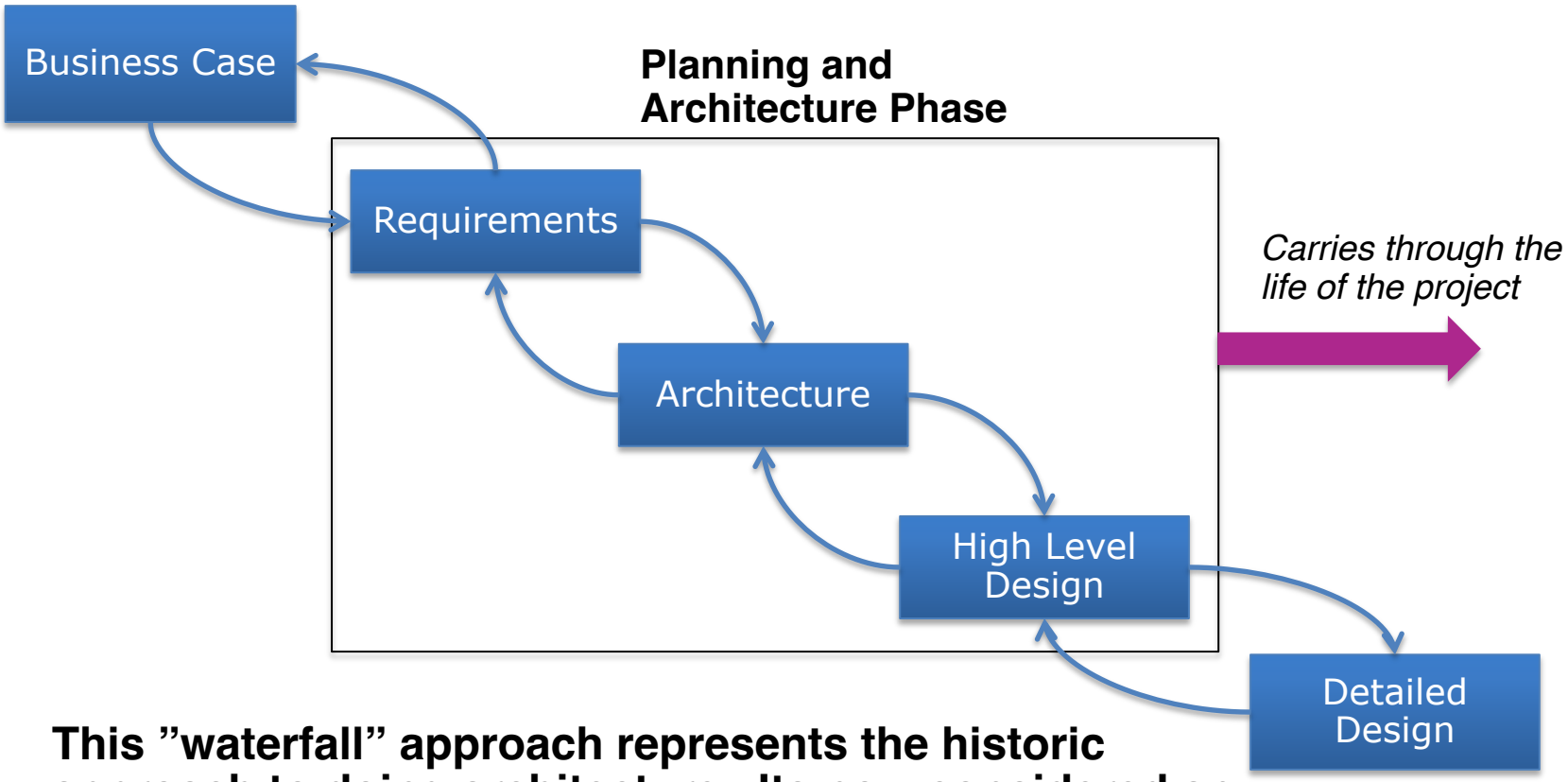
SE 577
Software Architecture

Basic SW Architectural Concepts

Acknowledgement

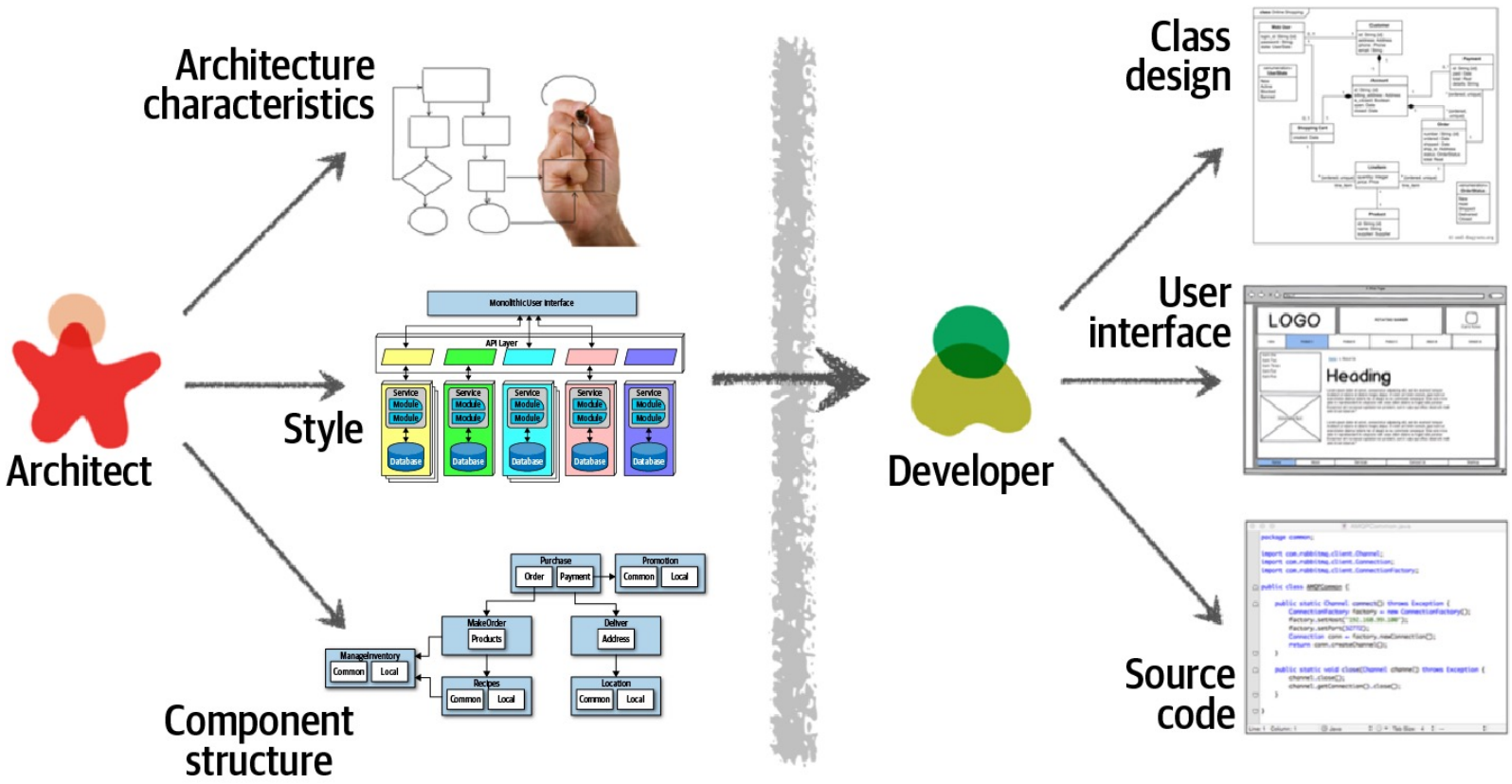
- Material from several sources including:
  - Ian Gorton. *Essential Software Architecture (2nd Edition)*, Springer-Verlag.
  - R.N. Taylor, N. Medvidovic, and E.M. Dashofy. *Software Architecture: Foundations, Theory and Practice*, Wiley.
  - Neil Ford & Mark Richards
    Software Architecture Fundamentals

Architecture in a Traditional Project Life Cycle



This ”waterfall” approach represents the historic approach to doing architecture. Its now considered an anti-pattern – “Big Architecture Up Front”

Major Areas of Concern for Architectural Design



Architecture in an Agile Project Life Cycle

Create Reference Architecture –
"how the proposed solution fits into the big picture"

Iterate

Create Solution Architecture for a Program Increment (approx. 5 Sprints)

Update Existing Architecture Decisions

Define new architecture decisions that need to be made

Create Technical Architecture Guidance to Guide Engineering, Delivery and Testing

Monitor Engineering, ensuring architecture guidance is followed or creating a backlog for adjusting architecture

Perform Architecture Retrospective

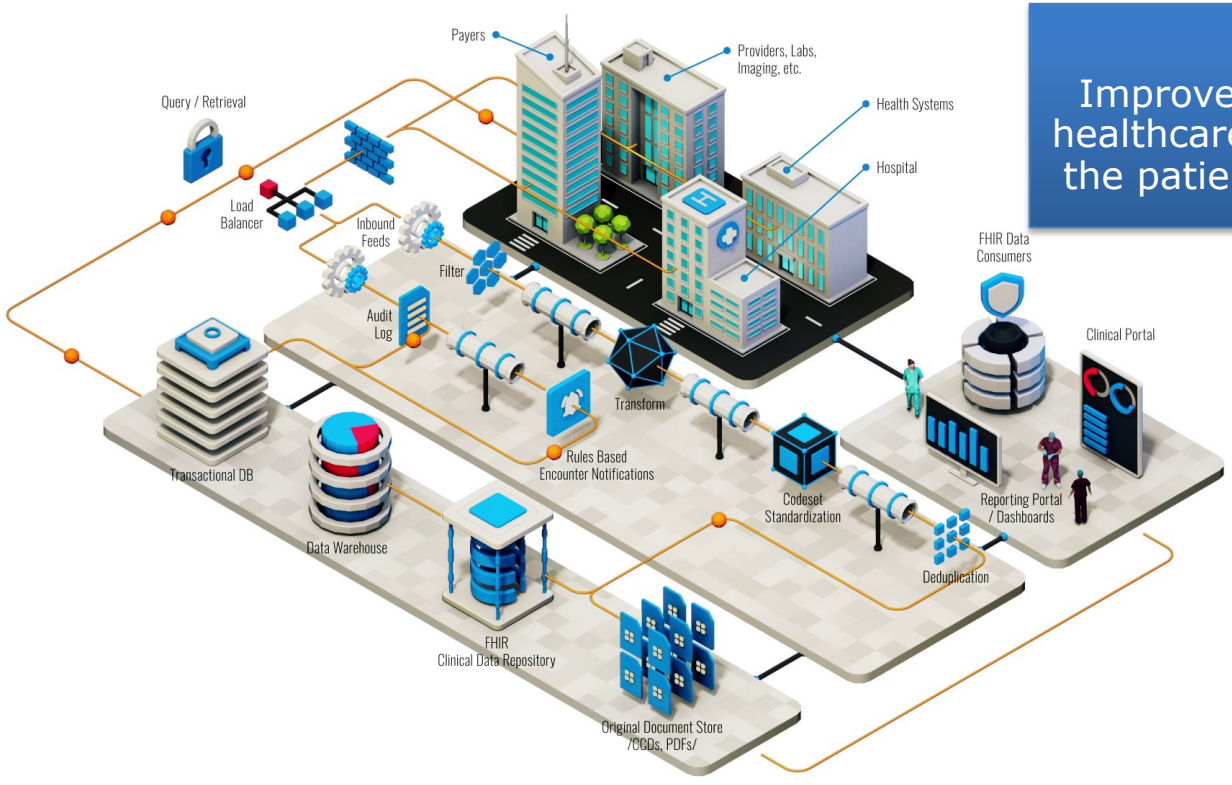
Monitor overall architecture against agreed objectives

Consistently support activities needed in support of quality attributes – e.g., security, compliance, etc

Negotiate and Influence on behalf of architecture and engineering ongoing progress and required changes

Example of Reference Architecture

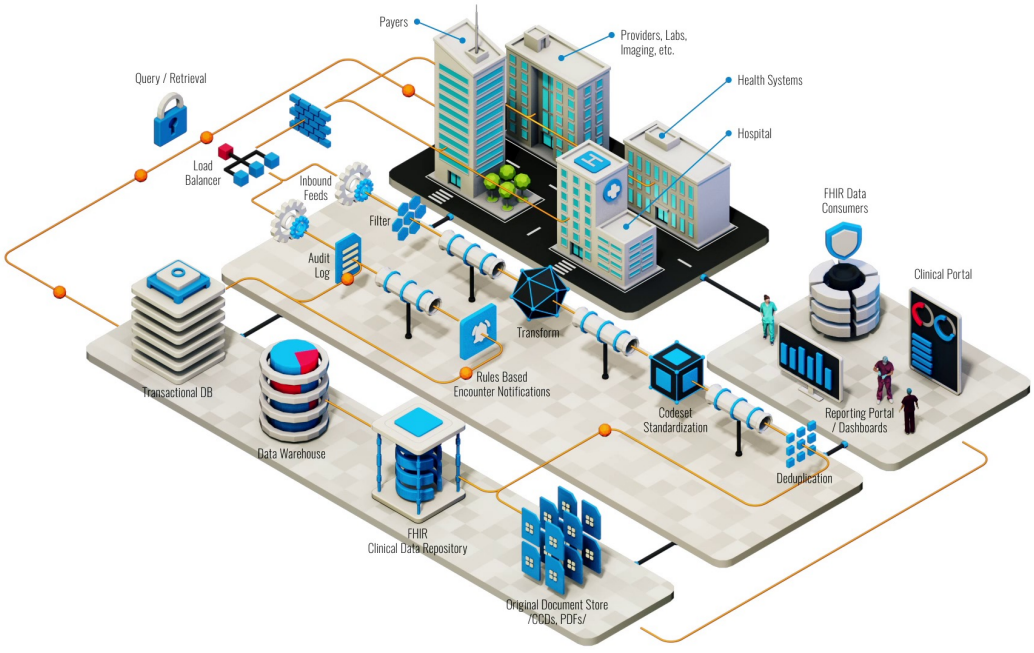
Create Reference Architecture –
"how the proposed solution fits into the big picture"



OBJECTIVE
Improve timeliness of insights from healthcare delivery system to improve the patient experience and outcomes

Example: Architecture Kickoff

Create Reference Architecture –
"how the proposed solution fits into the big picture"



OBJECTIVE

Improve timeliness of insights from healthcare delivery system to improve the patient experience and outcomes

KEY ARCHITECTURE NEEDS

Pick a healthcare interoperability partner – partner must support real-time FHIR events

Identify key systems that must be adjusted to react to real time patient data

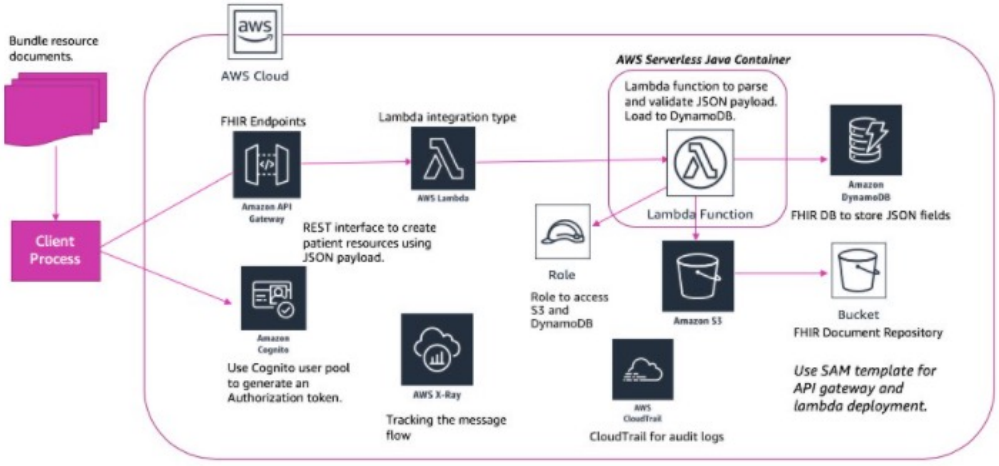
Build new data insights engine to process events real time and to build actionable insights

Modernize legacy systems to react in real time to key events of interest

Build analytics solution to measure effectiveness and patient outcomes

Example: Establish baseline solution architecture next

Create Solution Architecture –
"how the proposed solution will be structured"



OBJECTIVE

Employ cloud native architecture patterns to ensure resiliency, scale, and flexibility to ingest real-time clinical events

KEY ARCHITECTURE DECISIONS and GUIDENCE

Ensure that HIPAA privacy and security requirements are enforced in every component of the solution

Favor managed cloud services to support dynamic scalability and resiliency

Ensure audit and control policies are supported by the solution via CloudFront

Ensure data is flexibly stored to support FHIR resource standards

Major Areas of Concern for Architectural Design

- Requirements
  - Both *functional* and *non-functional*
- Operational excellence
  - Error recovery: How does the system handle *rainy day* scenarios?
  - Building resiliency and scale into the solution from the get go

Requirements

- Traditional SE suggests requirements analysis should remain unsullied by any consideration for a design.
- However, without reference to existing architectures it becomes difficult to assess practicality, schedules, or costs.
- Requirements analysis and consideration of design must be pursued at the same time.

Think about it, building a prototype app to get angel funding for a startup is a lot different than building a safety critical system

Non-Functional Requirements

- Non-Functional Requirements (NFRs) define “how” a system achieves its functional requirements.

- NFRs include:
  - Quality attributes
    - Application performance must provide sub-four second response times for 90% of requests.
  - Technical constraints
    - The system must be written in Java so that we can use existing development staff.
  - Business constraints
    - “We want to work closely with and get more development funding from *MegaHugeTech Corp*, so we need to use their technology in our application.”

Non-Functional Requirements (cont'd)

- NFRs rarely captured in functional requirements
  - They are also known as "architecture requirements"
  - Must be elicited by architect
- Specification of NFR might require an architectural framework to even enable their statement.
- An architectural framework will be required for assessment of whether the properties are achievable.

This is the key place where an architect plays –
Driving transparency around constraints and tradeoffs

System Quality Attributes

Ref: https://en.wikipedia.org/wiki/List_of_system_quality_attributes

Although there are many quality attributes, certain ones tend to appear again and again that impact the design and architecture of systems

- Modularity
- Maintainability
- Testability
- Availability
- Deployability
- Reliability
- Scalability
- Evolvability
- Affordability

++ FEASIBILITY ++

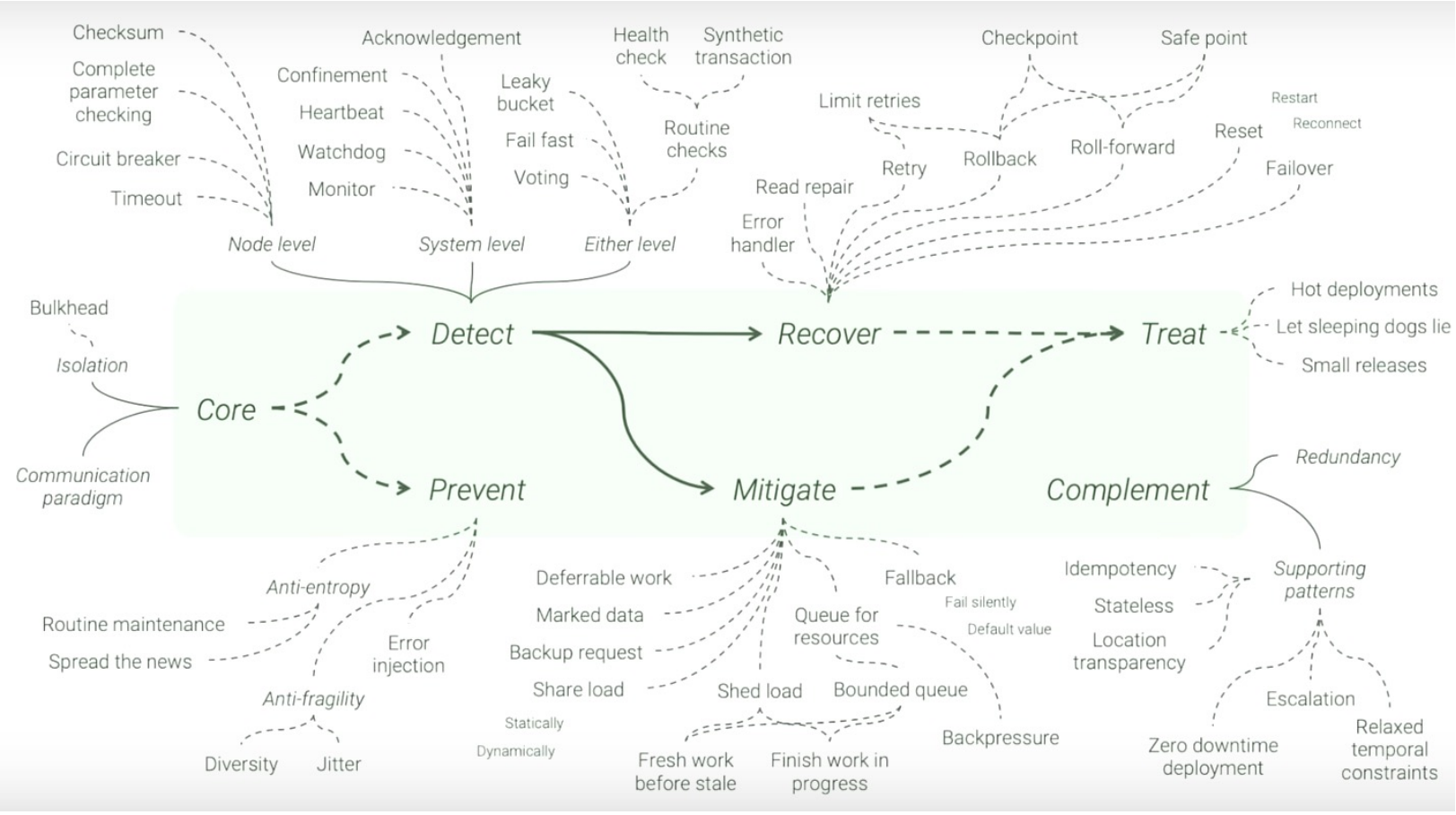
Error Recovery

- The system has to be designed from the standpoint of errors.
- The *sunny day* paths are easy; the error handling is what complicates the system.
- Must have a consistent approach for handling errors.
- Error recovery (like OA&M) is often overlooked in the system architecture.

Error Recovery: Kinds Of Errors

- Communication
  - Intermittent / interrupted communication
  - Unexpected / incorrect network changes
- Process
  - Abnormal termination
  - Asleep waiting (*patiently*) for an event (which may never occur, or that has been missed).
  - Resource leaks
- Coordination
  - Someone forgot to tell the up/downstream systems about the new data format.
  - Other systems in the network do not react kindly to a system going down due to a failure or for a scheduled upgrade

There is a whole body of work on architecture resiliency patterns



Ref: Uwe Fridrichsen – he has a lot of good YouTube talks on the subject

Software Architecture's Elements

- A software system's architecture should be a composition and interplay of different elements
  - Processing
  - Data, also referred as information or state
  - Interaction
- These are generally referred to as:
  - Components
  - Connectors
  - & Patterns and Constraints that govern how they can be connected to each other.

Components

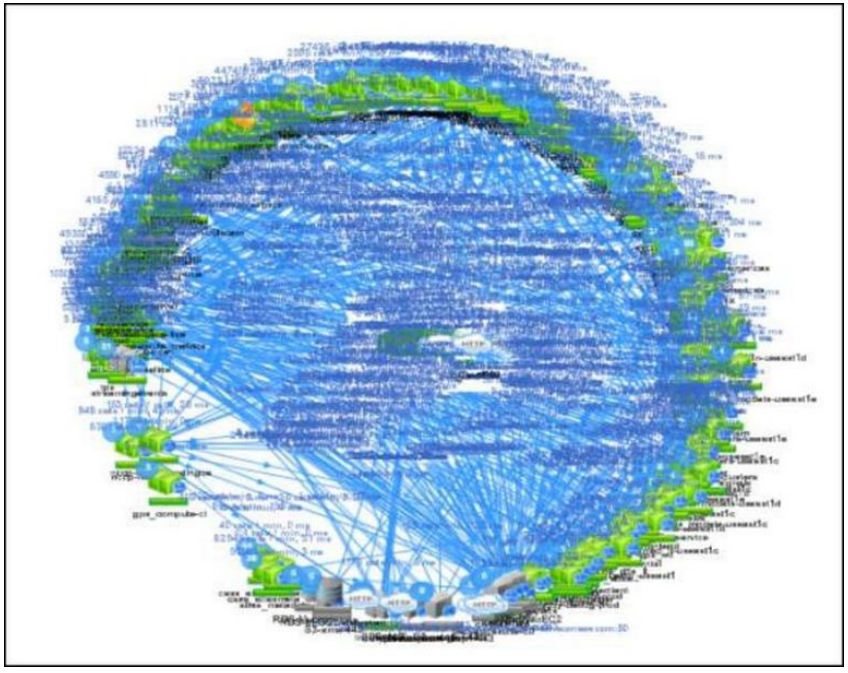
- Elements that encapsulate processing and data in a system's architecture are referred to as software components
- A software component is an architectural entity that:
  - Encapsulates a subset of the system's functionality and/or data.
  - Restricts access to that subset via an explicitly defined interface.
  - Has explicitly defined dependencies on its required execution context.
- Components typically provide application-specific services.

Connectors

- A software connector is an architectural building block tasked with effecting and regulating interactions among components.
- In complex systems interaction may become more important and challenging than the functionality of the individual components.
- In many software systems connectors are usually simple procedure calls or shared data accesses.
  - Much more sophisticated and complex connectors are possible!
- Connectors typically provide application-independent interaction facilities.

Examples of Connectors

- Procedure call connectors
- Shared memory connectors
- Message passing connectors
- Streaming connectors



The Netflix Microservice Architecture

Architecture Defines Structure

- Decomposition of system into components/modules/subsystems
- Architecture defines:
  - Component responsibilities
    - Precisely what a component will do when you ask it
  - Component interfaces
    - What a component can do
  - Component communications and dependencies
    - How components communicate

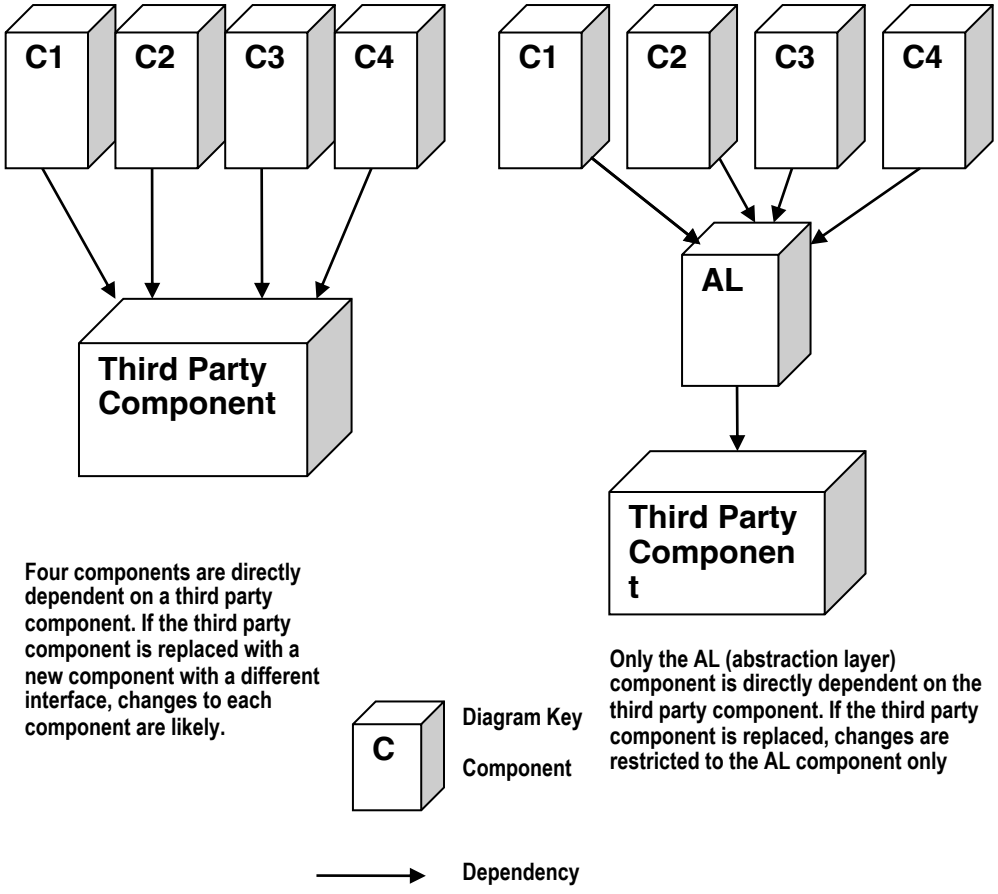
Architecture Specifies Component Communication

- Communication involves:
  - Data passing mechanisms
    - Remote method invocation
    - Asynchronous message
    - Streaming and eventing
  - Control flow
    - Sequential
    - Concurrent/parallel

An Important Concern: Structure and Dependencies

- Excessive component dependencies are bad!
- Key architecture issue
  - Identifying components that may change
  - Reduce direct dependencies on these components
- Creates more modifiable systems

Coupling & Cohesion



An Important Concern: Almost every modern systems is a distributed system

- Components may be owned by different entitles
- Interactions happen at massive scale across machine and network boundaries
- They are asynchronous by design
- They require automation just to manage them

Configurations

- Components and connectors are composed in a specific way in a given system's architecture to accomplish that system's objective
- An *architectural configuration*, or topology, is a set of specific associations between the components and connectors of a software system's architecture.

Architecture Styles/Patterns

- Patterns catalogue successfully used structures that facilitate certain kinds of component communication
  - Client-server
  - Message broker
  - Pipe-and-filter
- Patterns have well-known characteristics appropriate for particular types of requirements
- Patterns are very useful…
  - Reusable architectural blueprints
  - Help efficiently communicate a design
  - Large systems comprise a number of individual patterns
- "Patterns and Styles are the same thing – the patterns people won" [anonymous SEI member]

Three-Tiered Pattern

- Front Tier
  - Contains the user interface functionality to access the system's services
- Middle Tier
  - Contains the application's major functionality
- Back Tier
  - Contains the application's data access and storage functionality

Architectural Models, Views, and Visualizations

- Architecture Model
  - An artifact documenting some or all of the architectural design decisions about a system
- Architecture View
  - A subset of related architectural design decisions
- Architecture Visualization
  - A way of depicting some or all of the architectural design decisions about a system to a stakeholder

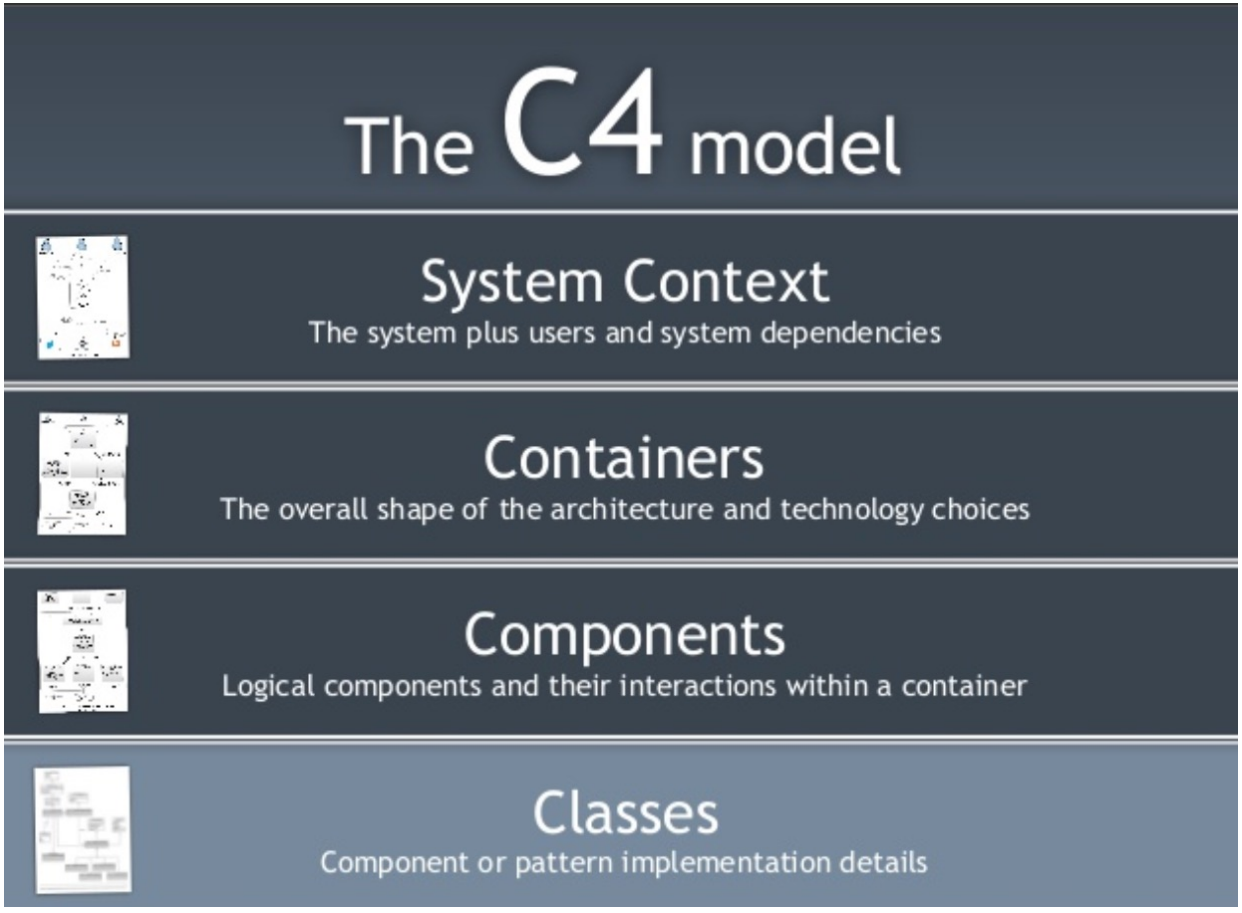
Architecture Views

- A software architecture represents a complex design artifact
- Many possible ‘views’ of the architecture
  - Cf. with buildings – floor plan, external, electrical, plumbing, air-conditioning

Architecture Views (cont'd)

- Logical view: describes architecturally significant elements of the architecture and the relationships between them.
- Process view: describes the concurrency and communications elements of an architecture.
- Physical view: depicts how the major processes and components are mapped on to the applications hardware.
- Development view: captures the internal organization of the software components as held in e.g. a configuration management tool.

Architecture Views (cont'd)



Prescriptive vs. Descriptive Architecture

- A system's *prescriptive architecture* captures the design decisions made prior to the system's construction
  - It is the *as-conceived* or *as-intended* architecture
- A system's *descriptive architecture* describes how the system has been built
  - It is the *as-implemented* or *as-realized* architecture

Temporal Aspect

- Architecture has a temporal aspect
  - Design decisions are made over a system's lifetime
  - A system's architecture will change over time
- At any given point in time the system has only one architecture

As-Designed vs. As-Implemented Architecture

- Which architecture is “correct”?
  - Intent vs. added experience of implementation
- Are the two architectures consistent with one another?
  - Architectural inconsistencies can be much more complex and/or insidious.
- What criteria are used to establish the consistency between the two architectures?
  - Sophisticated techniques might be needed
- On what information is the answer to the preceding questions based?
  - Diagrams may not contain adequate information

Architectural Evolution

- When a system evolves, ideally its prescriptive (*as-designed*) architecture is modified first.
- In practice, the system – and thus its descriptive (*as-implemented*) architecture – is often directly modified.
- This happens because of:
  - Developer sloppiness
  - Perception of short deadlines which prevent thinking through and documenting
  - Lack of documented prescriptive architecture
  - Need or desire for code optimizations
  - Inadequate techniques or tool support

Architectural Degradation

- Two related concepts
  - Architectural drift
  - Architectural erosion
- *Architectural drift* is introduction of principal design decisions into a system's descriptive architecture that
  - Are not included in, encompassed by, or implied by the prescriptive architecture
  - But which do not violate any of the prescriptive architecture' s design decisions
- *Architectural erosion* is the introduction of architectural design decisions into a system's descriptive architecture that violate its prescriptive architecture

Architectural Recovery

- If architectural degradation is allowed to occur, one will be forced to *recover* the system's architecture sooner or later
- *Architectural recovery* is the process of determining a software system's architecture from its implementation-level artifacts
- Implementation-level artifacts can be
  - Source code
  - Executable files
  - Java .class files