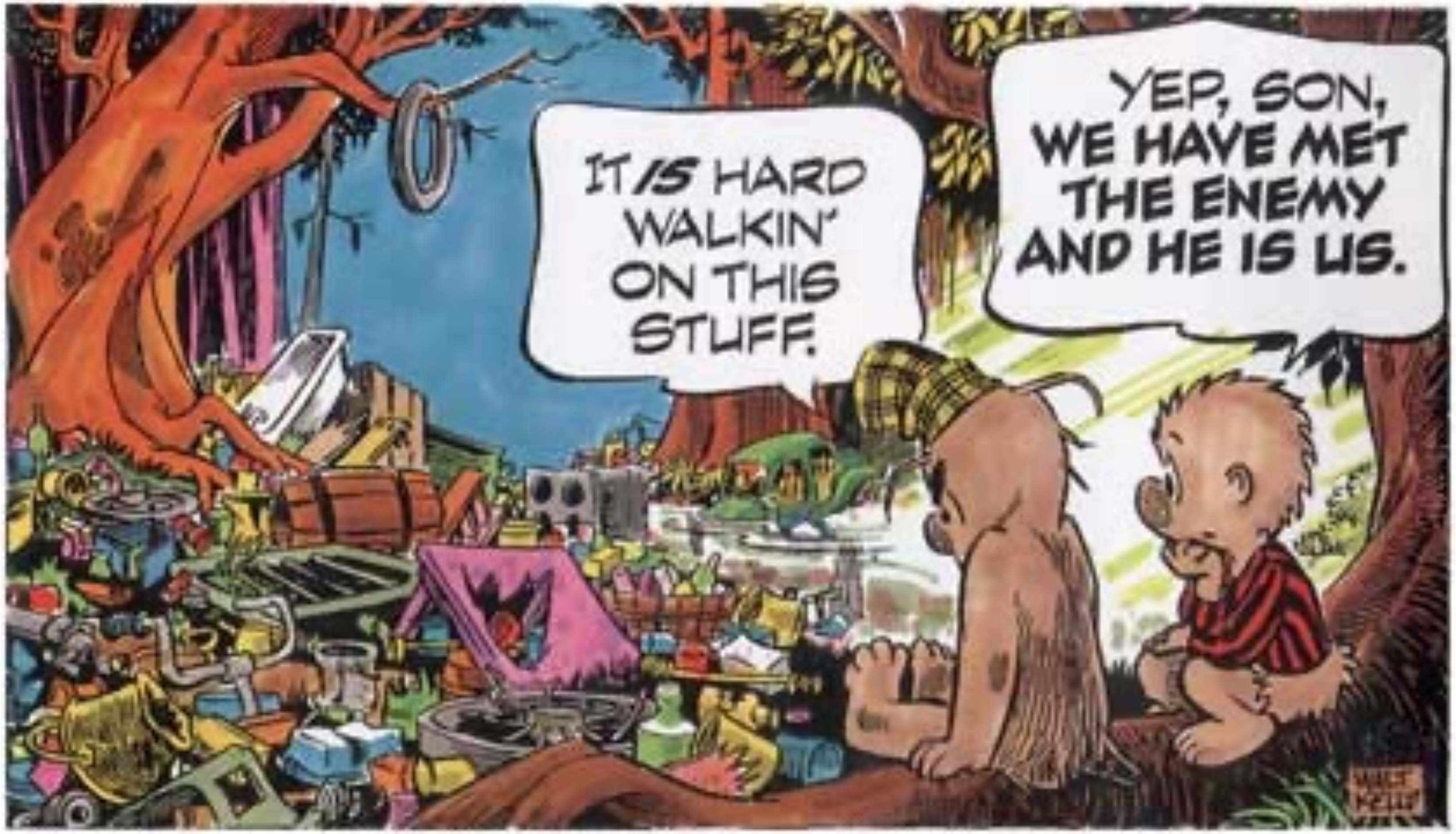
The origin of the quote at the end of Martin Fowler's article


IT *IS* HARD WALKIN' ON THIS STUFF.
YEP, SON, WE HAVE MET THE ENEMY AND HE IS US.
WALT KELLY

No Silver Bullet

Essential Complexity

- The Problem is complex.
- Cannot be refactored away
- Can increase over time, and sometimes faster than we can invent better ways to solve the problem.

Accidental Complexity

- The Solution is complex.
- Can be refactored away.
- Hopefully decreases as we invent better ways to solve the problem. Examples:
    - invention of high level programming languages
    - automatic memory management with garbage collection
    - actor model for concurrency

Brooks, Frederick P. “No Silver Bullet.” *Software State-of-the-Art*, 1975, 14–29.

https://en.wikipedia.org/wiki/No_Silver_Bullet

Software architecture

Architectural styles and more process

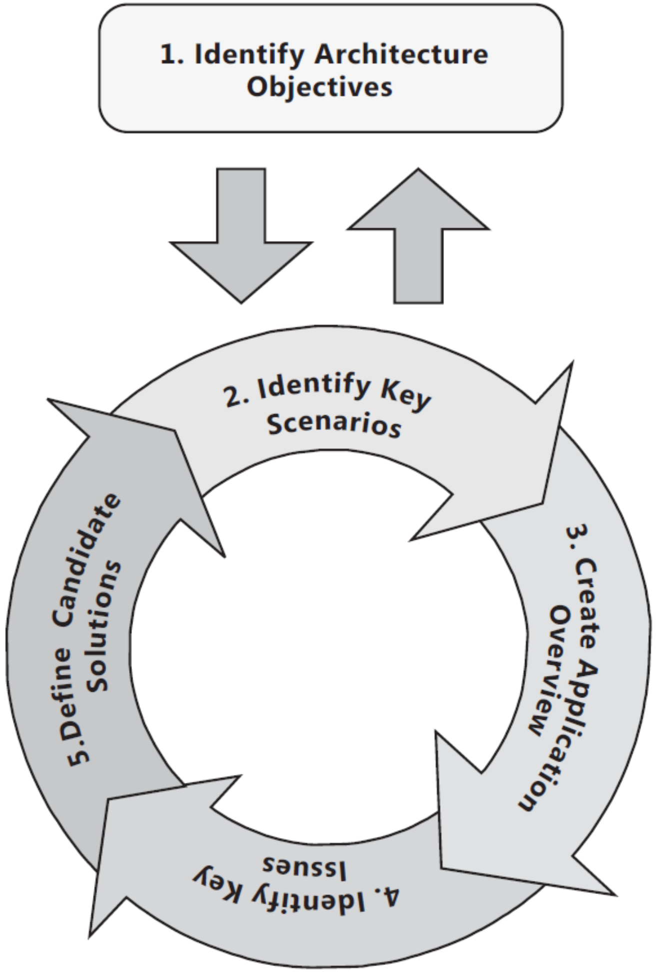
version: 1.0.3

Agenda

- Recap of the first steps in the architecture process
- Architectural styles
    - Layers – Concerns partitioned in layers
    - Client-server - the client makes requests to the server.
    - N-tier – Similar to layers, but each layer is in a separate computer
    - Pipes and filters – data flows and gets transformed in a pipeline.
    - Message bus – applications interact via a comunication channel.
- Quality attributes for videoflix prototype
- The next step in the architectural process
- Exercise: create an application overview

Recap

An architecture process (from Microsoft)


1. Identify Architecture Objectives
2. Identify Key Scenarios
3. Create Application Overview
4. Identify Key Issues
5.Define Candidate Solutions

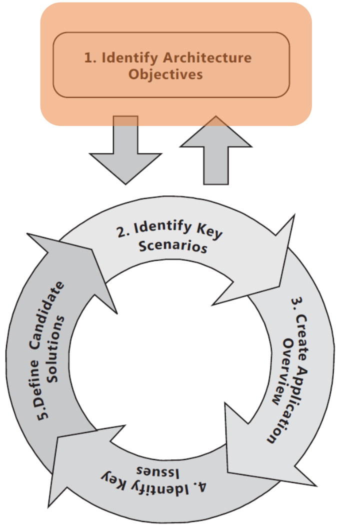
Iterative and incremental.

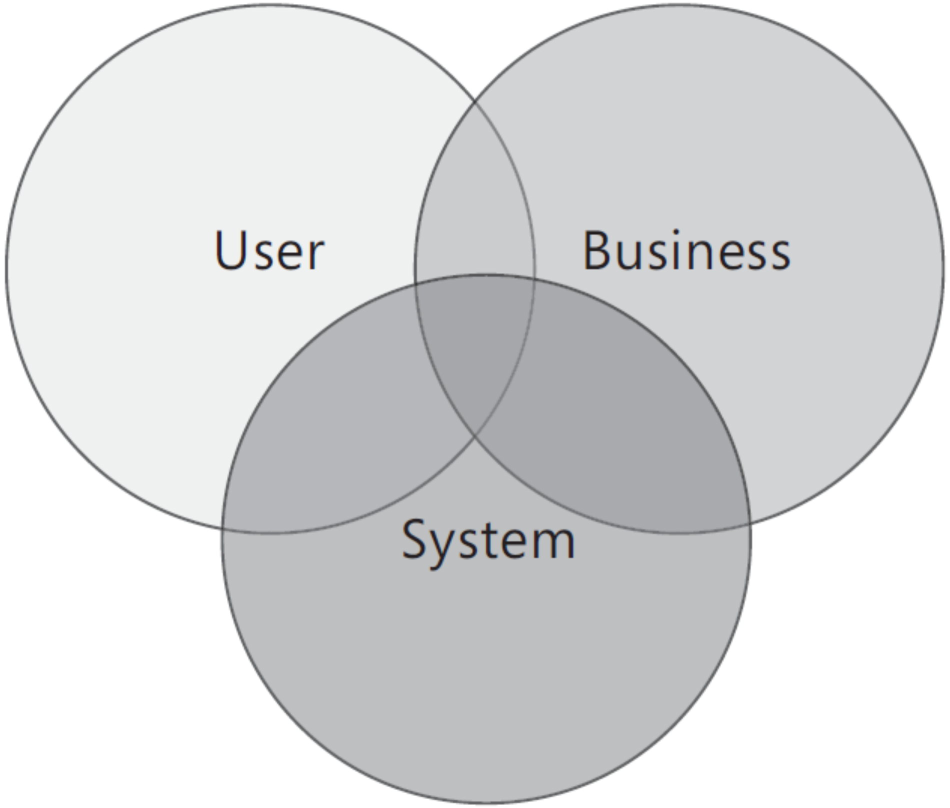
Test against:

- requirements
- known constraints
- quality attributes

Source: Microsoft Application Architecture Guide, 2nd Edition

1. Architecture objectives - input


1. Identify Architecture Objectives
2. Identify Key Scenarios
3. Create Application Overview
4. Identify Key Issues
5.Define Candidate Solutions



Goals and constraints shape your architecture and design process.

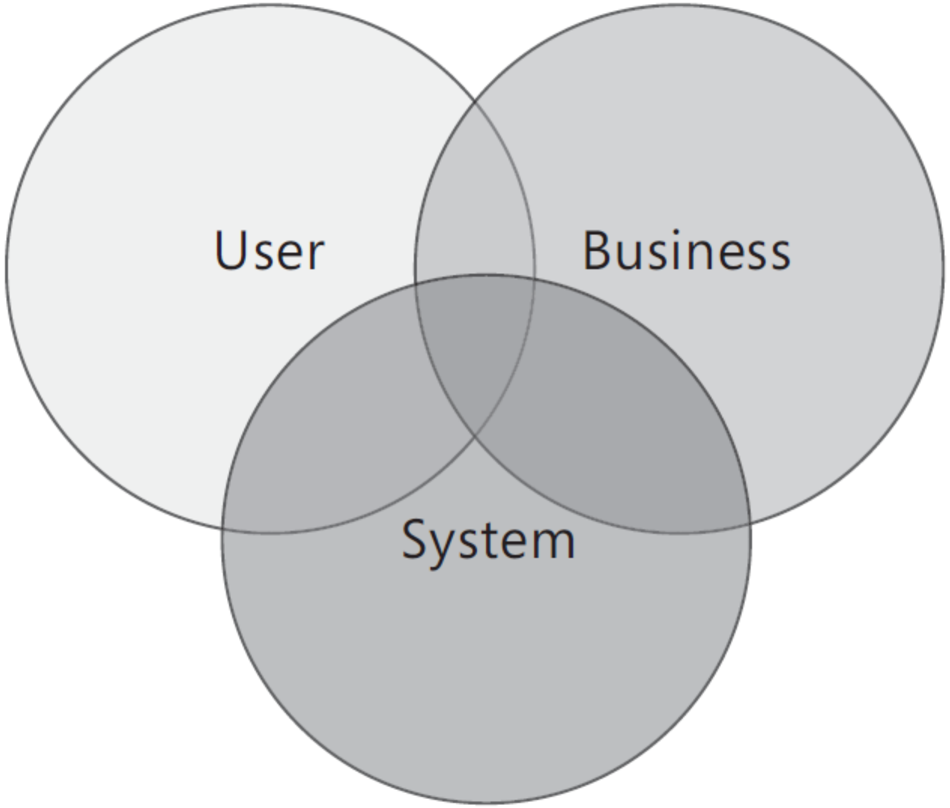
The architecture must satisfy:

- Functional requirements
- Non-functional requirements
  - External requirements (e.g. standards)
  - Organizational requirements
  - Product requirements (quality attributes)

  Cross-cutting concerns!

Source: Microsoft Application Architecture Guide, 2nd Edition

1. Architecture objectives – output

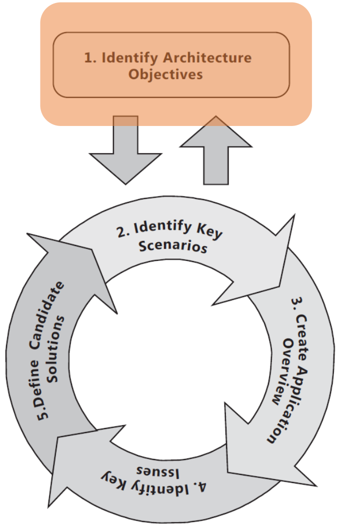


Who will use the output of this iteration?

- Management?
- Testers?
- Developers?
- Other architects?

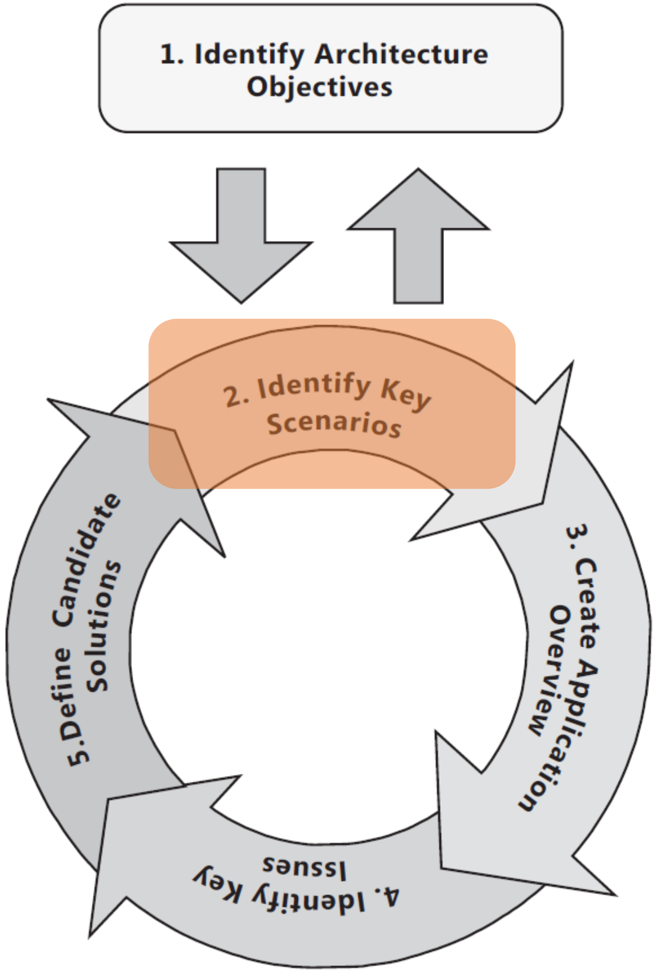
Are you:

- Creating a complete application design?
- Building a prototype?
- Examining technical risks?
- Testing potential options?
- Building shared models to gain an understanding of the system?


1. Identify Architecture Objectives
2. Identify Key Scenarios
3. Create Application Overview
4. Identify Key Issues
5.Define Candidate Solutions

Source: Microsoft Application Architecture Guide, 2nd Edition

2. Key scenarios


1. Identify Architecture Objectives
2. Identify Key Scenarios
3. Create Application Overview
4. Identify Key Issues
5.Define Candidate Solutions

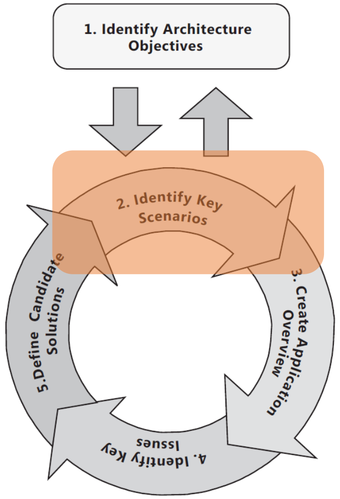
Source: Microsoft Application Architecture Guide, 2nd Edition

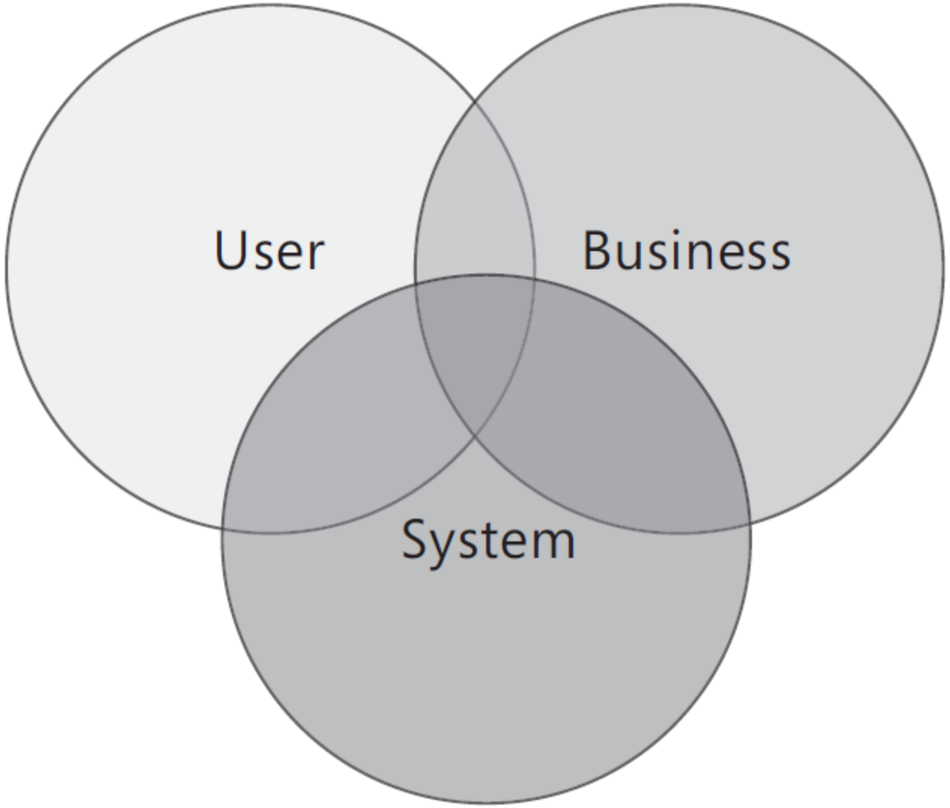
What are Key Scenarios?

This is where use cases and quality attributes meet!

To identify them takes practice, but here are some pointers:

2. Key scenarios


1. Identify Architecture Objectives
2. Identify Key Scenarios
3. Create Application Overview
4. Identify Key Issues
5.Define Candidate Solutions



Critical functionality.

Critical non-functional reqs

Exploration (unknown areas)

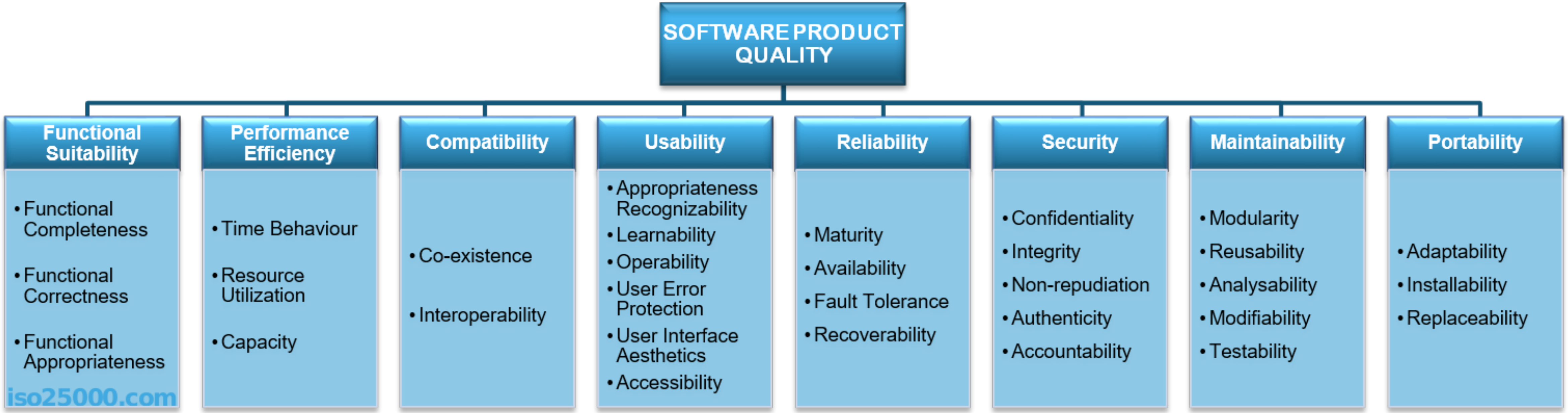
Risk mitigation

Look for intersections between the user, business and system views.

Prefer exercising multiple layers in the architecture when you select scenarios for the current iteration

Source: Microsoft Application Architecture Guide, 2nd Edition

Quality attributes



SOFTWARE PRODUCT QUALITY

Functional Suitability
- Functional Completeness
- Functional Correctness
- Functional Appropriateness

Performance Efficiency
- Time Behaviour
- Resource Utilization
- Capacity

Compatibility
- Co-existence
- Interoperability

Usability
- Appropriateness Recognizability
- Learnability
- Operability
- User Error Protection
- User Interface Aesthetics
- Accessibility

Reliability
- Maturity
- Availability
- Fault Tolerance
- Recoverability

Security
- Confidentiality
- Integrity
- Non-repudiation
- Authenticity
- Accountability

Maintainability
- Modularity
- Reusability
- Analysability
- Modifiability
- Testability

Portability
- Adaptability
- Installability
- Replaceability

http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Example Quality Attributes selected by Microsoft

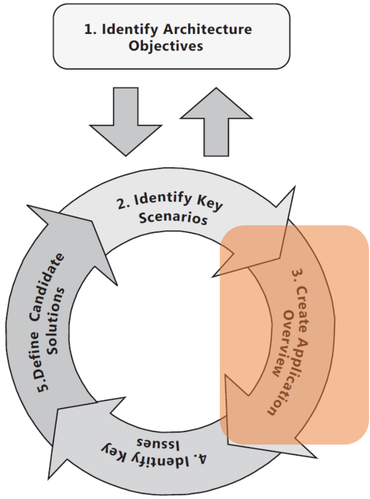
- Availability
- Conceptual Integrity
- Interoperability
- Maintainability
- Manageability
- Performance
- Reliability
- Reusability
- Scalability
- Security
- Supportability
- Testability
- User Experience / Usability

See more in chapter 16 of "Microsoft Application Architecture Guide, 2nd Ed." as found on Brightspace

Selected QA for Course

Quality Attribute	Explanation
Development cost	Reusability. Low complexity – in spite of size
Development time	Modular. Time to market – work in parallel
Testability	Test principles – from unit tests to acceptance test
Maintainability	How easy is it to correct, change and expand
Availability	Reliability. 99,99… %
Manageability	How easy is it to operate it on a day to day basis
Performance	Latency (delay until the first answer) Throughput (how many users can it support)
Scalability	Can performance grow? Can the system be downgraded? Is it dynamically scalable?
Security	Protect data (protect against unauthorized reading and writing) Protect the system
User Experience	User friendliness/experience

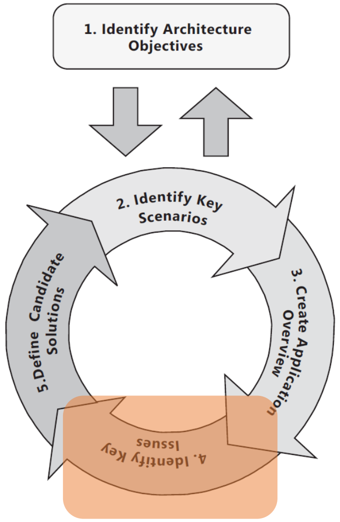
3. Create an application overview


1. Identify Architecture Objectives
2. Identify Key Scenarios
3. Create Application Overview
4. Identify Key Issues
5.Define Candidate Solutions

- What is an application overview?
  - It is one or more proposals for an architecture
  - Determine your application type.
  - Identify your deployment constraints.
  - Determine relevant technologies.
  - Identify important architectural design styles.

Topic for today's lecture

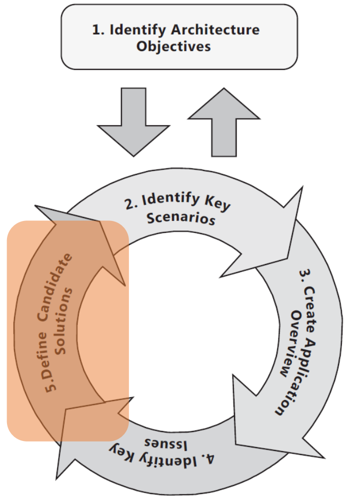
4. Identify Key Issues


1. Identify Architecture Objectives
2. Identify Key Scenarios
3. Create Application Overview
4. Identify Key Issues
5.Define Candidate Solutions

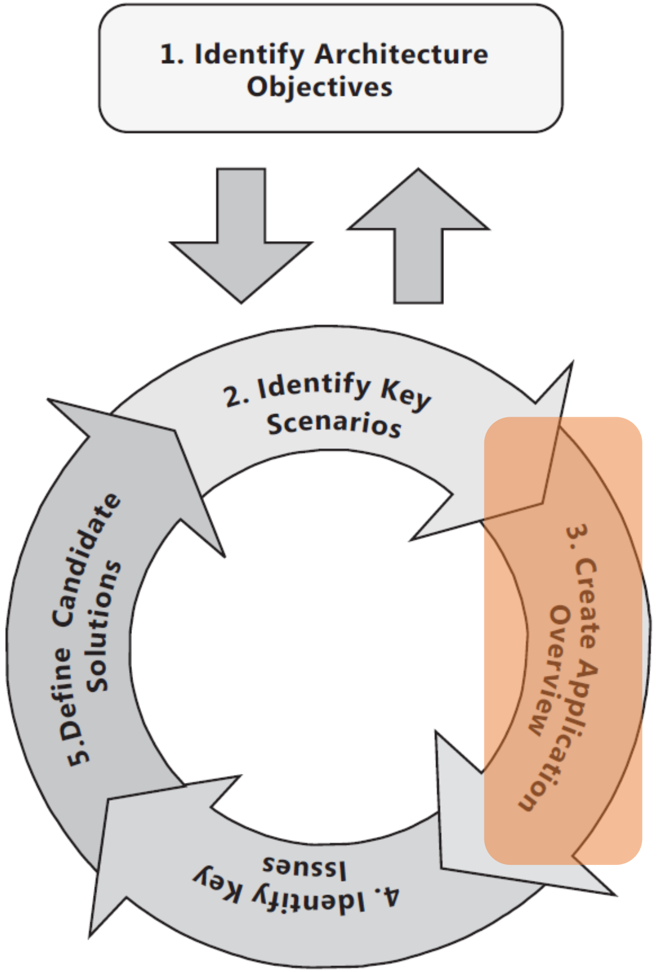
- What are Key Issues?

- Problems that must be solved with this (version of the) architecture
- E.g. Key Scenarios, are they solved?
- Pose relevant hypothetical future changes:
  - “Can I swap from one third party service to another?,”
  - “Can I add support for a new client type?,”
  - “Can I quickly change my business rules relating to billing?,”
  - “Can I migrate to a new technology for X?”
- Try it out, analyze and document outcomes

5. Define Candidate Solutions


1. Identify Architecture Objectives
2. Identify Key Scenarios
3. Create Application Overview
4. Identify Key Issues
5.Define Candidate Solutions

- Propose candidate solutions to key issues.
- Evaluate against "baseline" architecture:
  - Does this architecture succeed without introducing any new risks?
  - Does this architecture mitigate more known risks than the previous iteration?
  - Does this architecture meet additional requirements?
  - Does this architecture enable architecturally significant use cases?
  - Does this architecture address quality attribute concerns?
  - Does this architecture address additional crosscutting concerns?
- May involve coding to validate assumptions (architectural spike)


1. Identify Architecture Objectives
2. Identify Key Scenarios
3. Create Application Overview
4. Identify Key Issues
5.Define Candidate Solutions

Application overview

Architectural styles

The toolbox for the Application Overview

What Is an Architectural Style?

Garlan and Shaw define an architectural style as:

"…a family of systems in terms of **a pattern of structural organization**. More specifically, an architectural style determines the vocabulary of components and connectors that can be used in instances of that style, together with a set of constraints on how they can be combined. These can include topological constraints on architectural descriptions (e.g., no cycles). Other constraints—say, having to do with execution semantics—might also be part of the style definition."

[David Garlan and Mary Shaw, January 1994, CMU-CS-94-166, see "*An Introduction to Software Architecture*" at http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf]

What Is an Architectural Style?

Garlan and Shaw define an architectural style as:

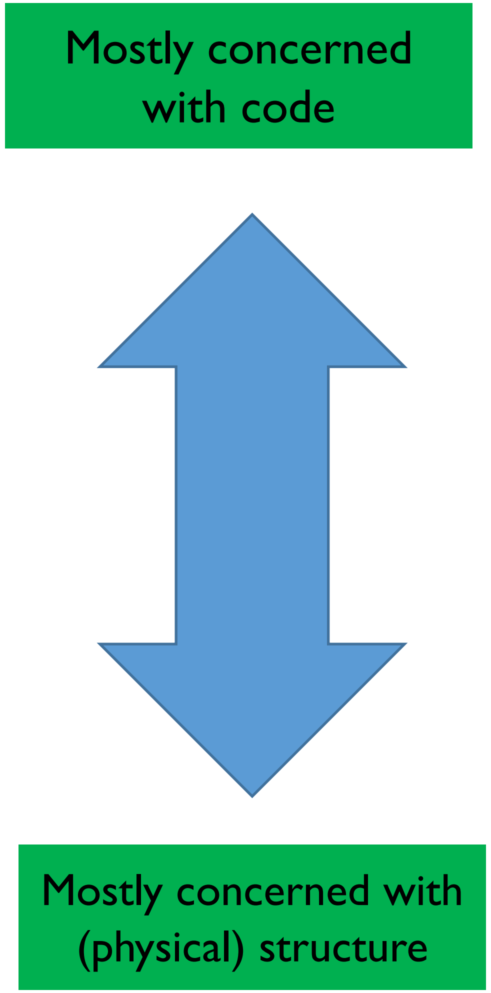
“ **a pattern of structural organization**. the vocabulary of components and connectors together with a set of constraints on how they can be combined.

”

[David Garlan and Mary Shaw, January 1994, CMU-CS-94-166, see “*An Introduction to Software Architecture*” at http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf]

Categories based on focus area

- Structure
  - Layered, Component-based, Object Oriented
- Domain
  - Domain Driven Design
- Communication
  - Message Bus, SOA, *Event driven, CQRS*
- Deployment
  - Client/Server, N-Tier / 3-Tier, *Microservice, Serverless*



Mostly concerned with code

Mostly concerned with (physical) structure

[MS AAG] (+ additions)

Combining Architectural Styles

The architecture of a software system is almost never limited to a single architectural style but is often a combination of architectural styles that make up the complete system.

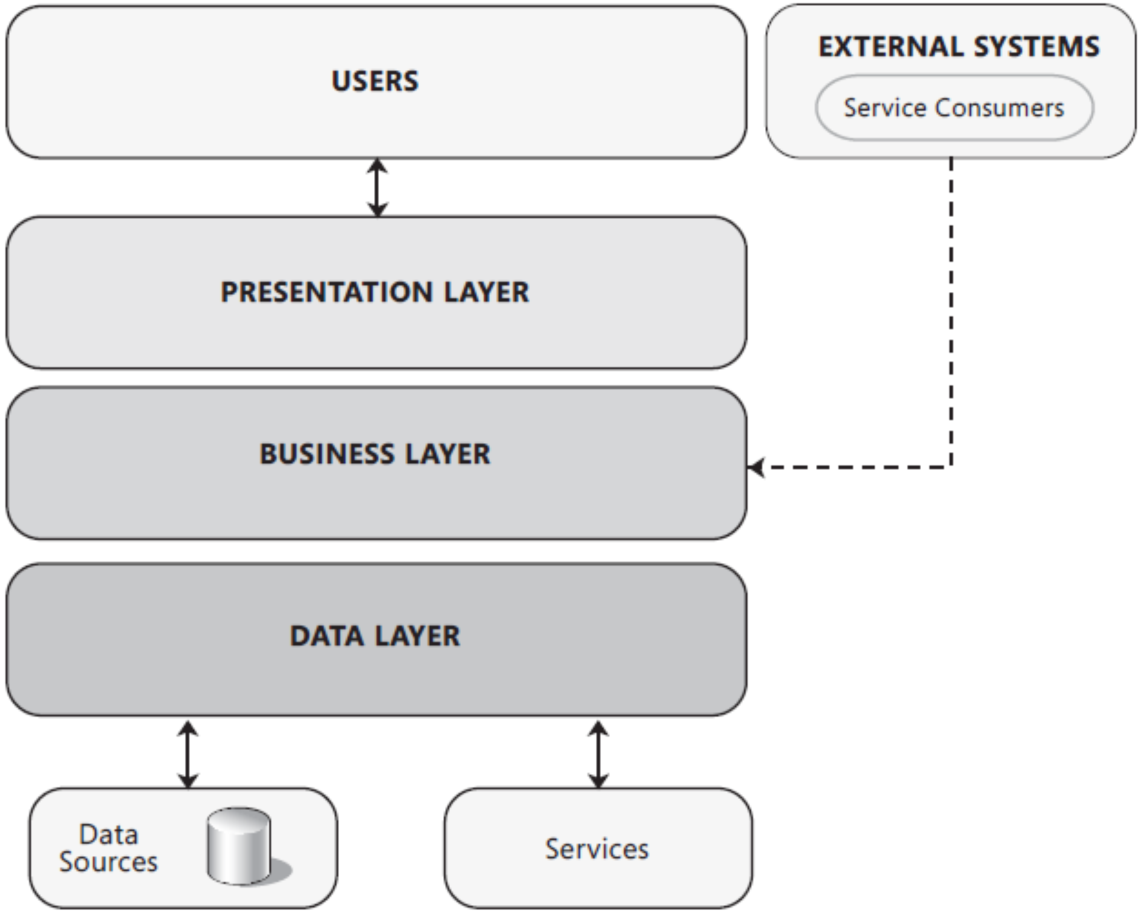
For example, you might have a message bus design composed of services developed using a layered architecture approach.

Styles we will look at

- Layers
- Client-server
- N-tier
- Pipes and filters
- Message bus

Layers

Architectural style: Layers



Warning – DIP talks about High Level Abstraction Modules vs. Low Level Abstraction Modules

Partitions the concerns of the application into stacked groups (layers) of:

- classes/packages/modules/subsystems

Dependencies are only allowed from higher layer to lower layer.

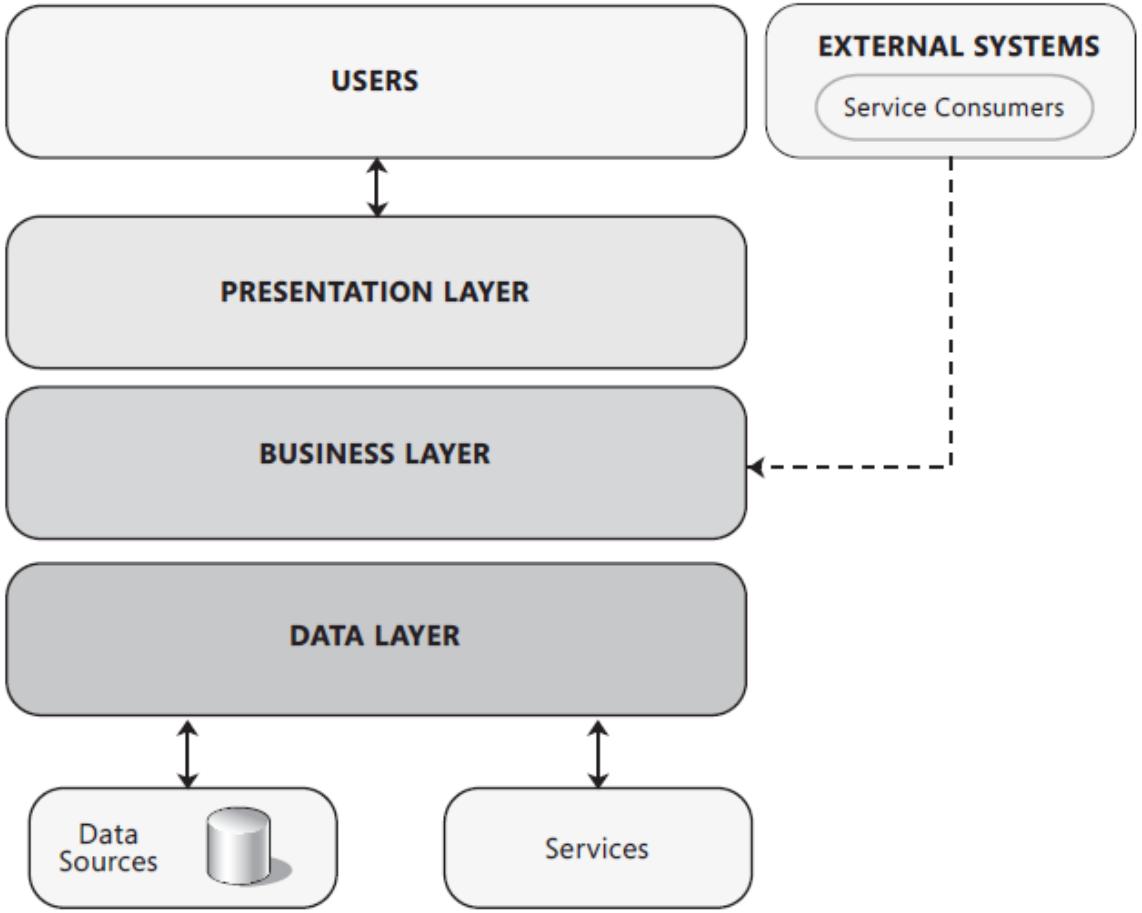
- Just like you know from DIP Dependency Inversion Principle.

Don't get confused – this doesn’t mean that data cannot flow up!

Decoupling is important – e.g. using events. The lower layer should work without the higher layer.

Source: Microsoft Application Architecture Guide, 2nd Edition

Architectural style: Layers



Layer vs. Tier

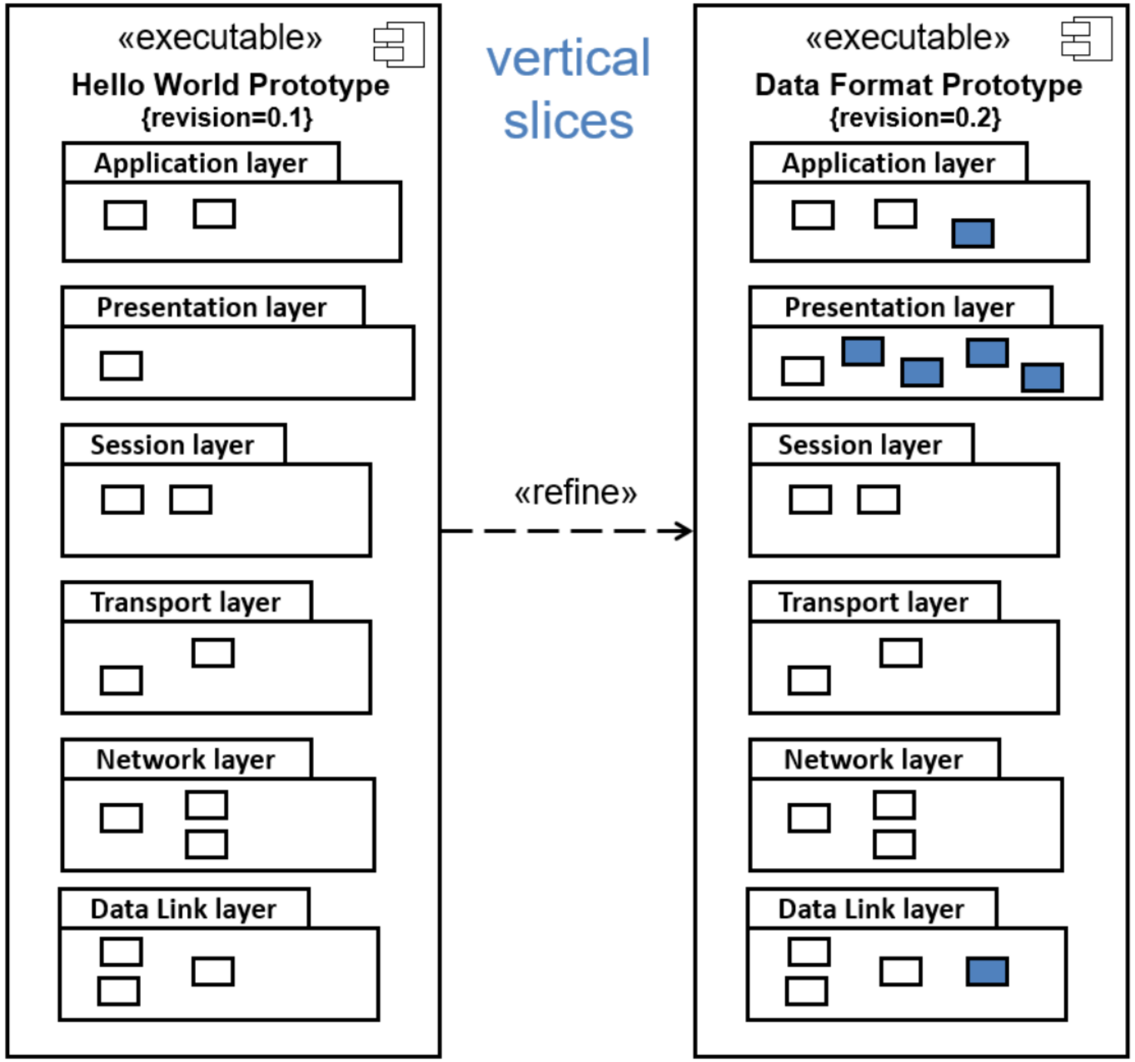
- Layers is a logical separation.
- Tier represent a physical separation.

Other examples

- MVC – Model-View-Control
- MVVM – Model-View-ViewModel
- Boundary-Control-Domain/Entity
- Network Layers (ISO OSI model)

Source: Microsoft Application Architecture Guide, 2nd Edition

Iterative development of layers



Benefits of Using Layers

Separation of concerns (SRP)

- separation of application-specific from general services.
- separation of high-level from low-level services

Reduces coupling and dependencies

Improves cohesion

Increases potential reuse

- Lower layers can easily be reused in other applications

Increases clarity

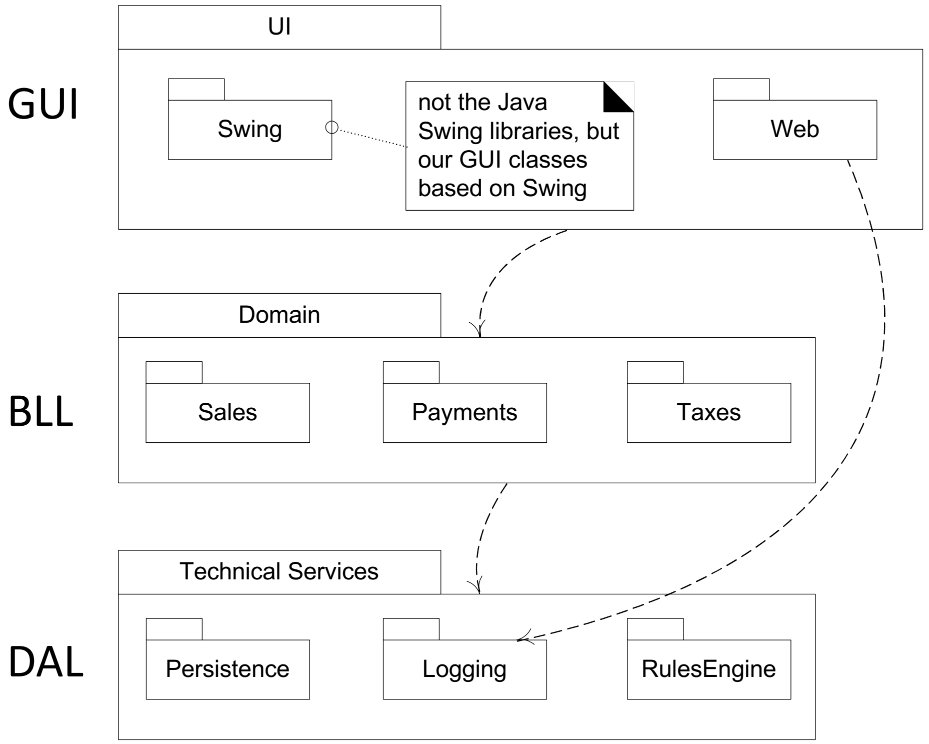
Concurrent development by teams is aided by the logical segmentation

A layer can be replaced

- if interfaced based programming and dependency injection is used

Testing is easier

Typical Layers for a small Application



Larman fig. 13.2

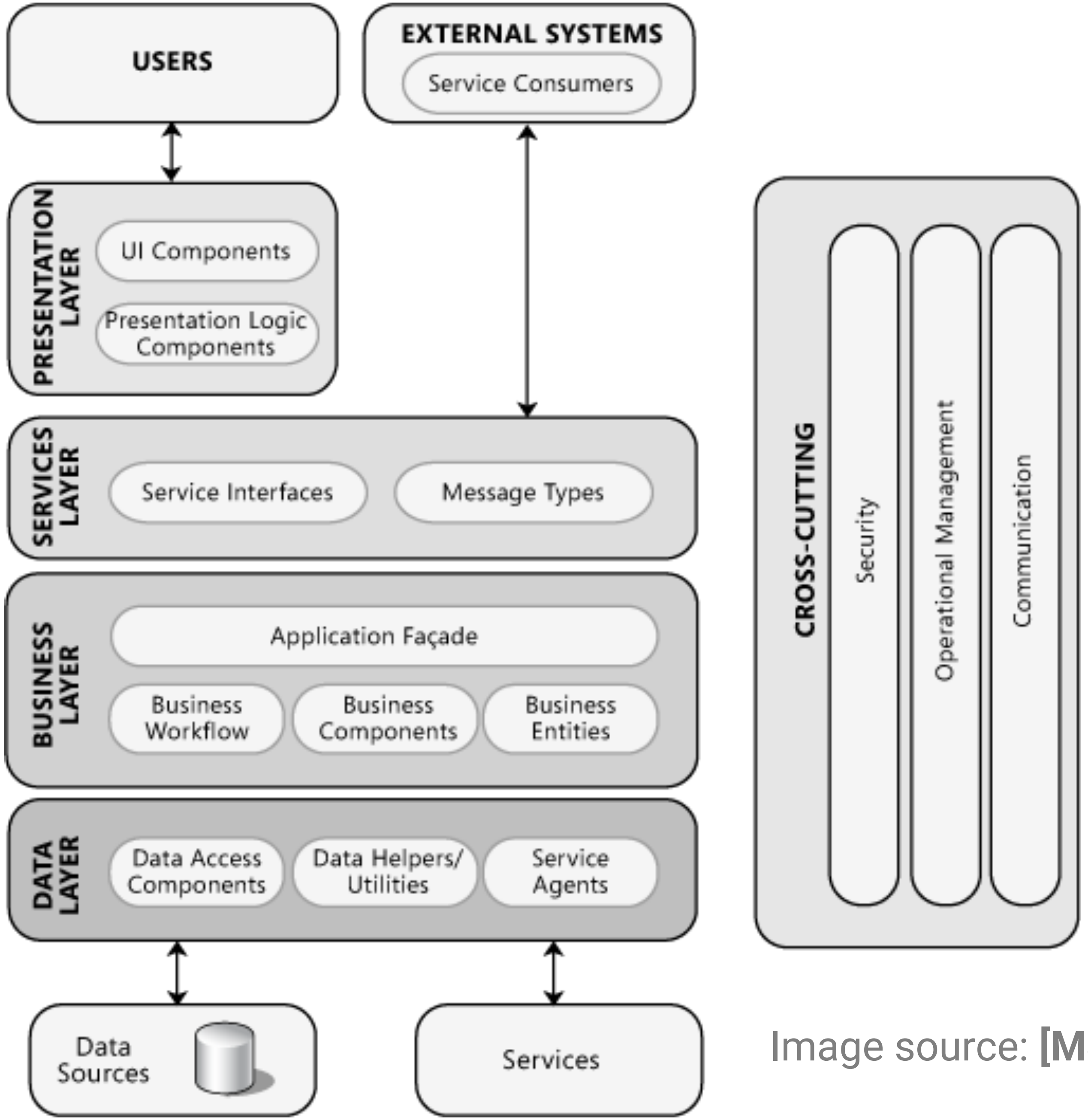


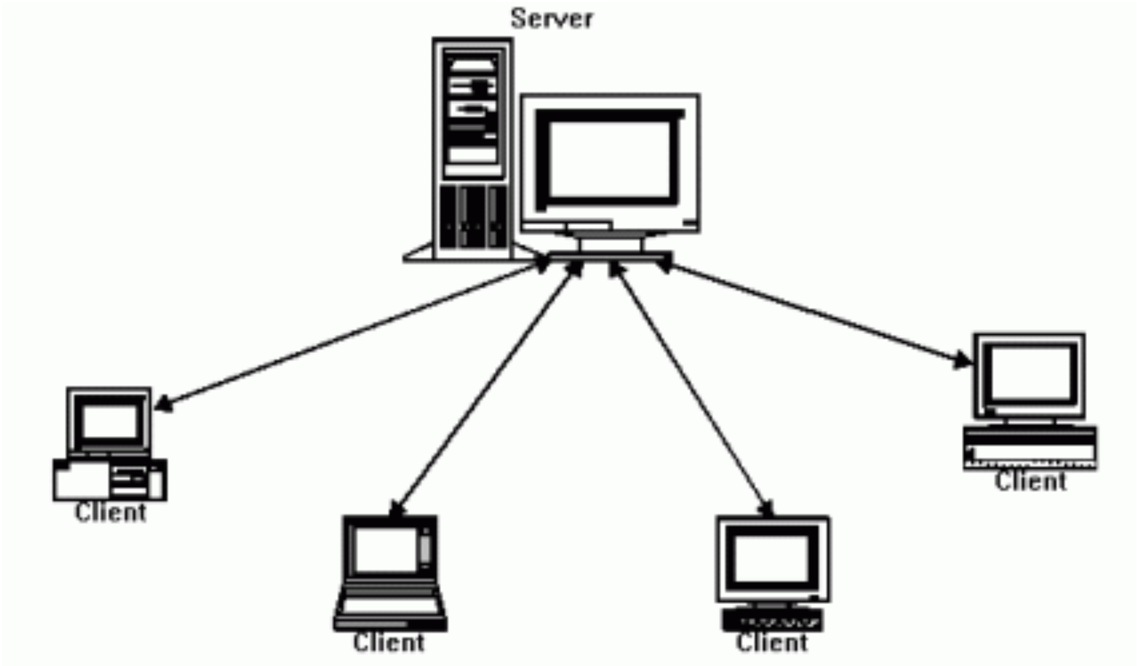
Image source: **[MS AAG]**

Selected QA for Course

Quality Attribute	Explanation
Development cost	Reusability. Low complexity – in spite of size
Development time	Modular. Time to market – work in parallel
Testability	Test principles – from unit tests to acceptance test
Maintainability	How easy is it to correct, change and expand
Availability	Reliability. 99,99… %
Manageability	How easy is it to operate it on a day to day basis
Performance	Latency (delay until the first answer) Throughput (how many users can it support)
Scalability	Can performance grow? Can the system be downgraded? Is it dynamically scalable?
Security	Protect data (protect against unauthorized reading and writing) Protect the system
User Experience	User friendliness/experience

Client/Server

Client/Server Architectural Style



A client initiates a requests, waits for replies, and processes the replies on receipt

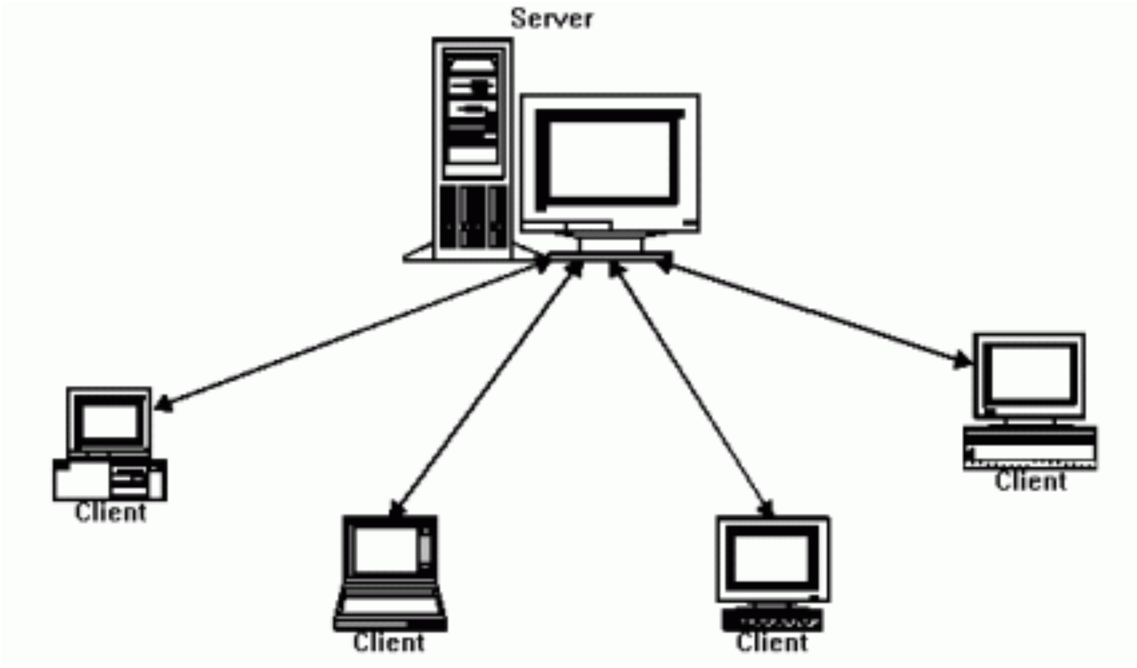
The clients and servers are connected by some kind of network.

Multiple clients connect to the same server: Many to One

The Client and the Server have different roles

The clients can vary from thin to thick clients based on how much they must process themselves

Client/Server Architectural Style



The server can

- Handle the request
- Save and serve data
- Calculate or process data

The client can

- Handle user input
- Assemble the request
- Receive the data
- Process and/or display data

This is aka 2-Tier Architectural Style

Selected QA for Course

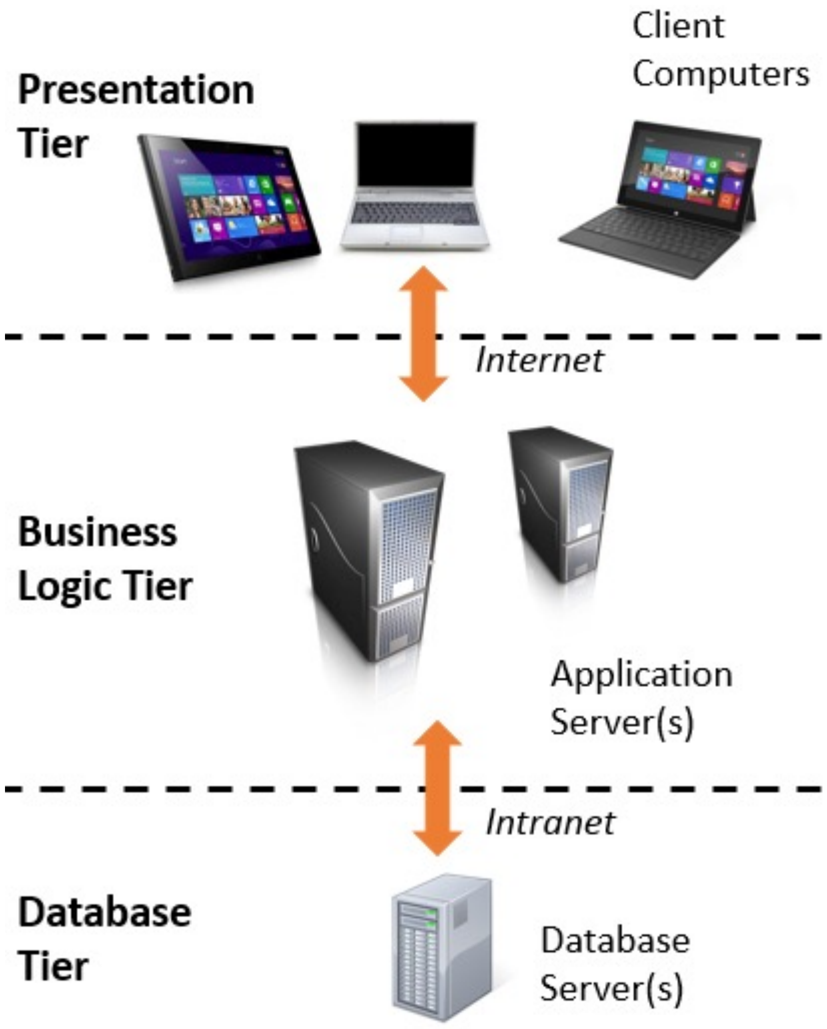
Quality Attribute	Explanation
Development cost	Reusability. Low complexity – in spite of size
Development time	Modular. Time to market – work in parallel
Testability	Test principles – from unit tests to acceptance test
Maintainability	How easy is it to correct, change and expand ⬅
Availability	Reliability. 99,99… %
Manageability	How easy is it to operate it on a day to day basis ⬅
Performance	Latency (delay until the first answer) Throughput (how many users can it support)
Scalability	Can performance grow? Can the system be downgraded? Is it dynamically scalable?
Security	Protect data (protect against unauthorized reading and writing) ⬅ Protect the system
User Experience	User friendliness/experience ⬅

N-Tier

(EN-EN): tier, n., row, rank, esp. one of several placed one above another as in a theatre

(EN-DK): tier [tiə] sb (trinvis opstigende) række; etage, lag

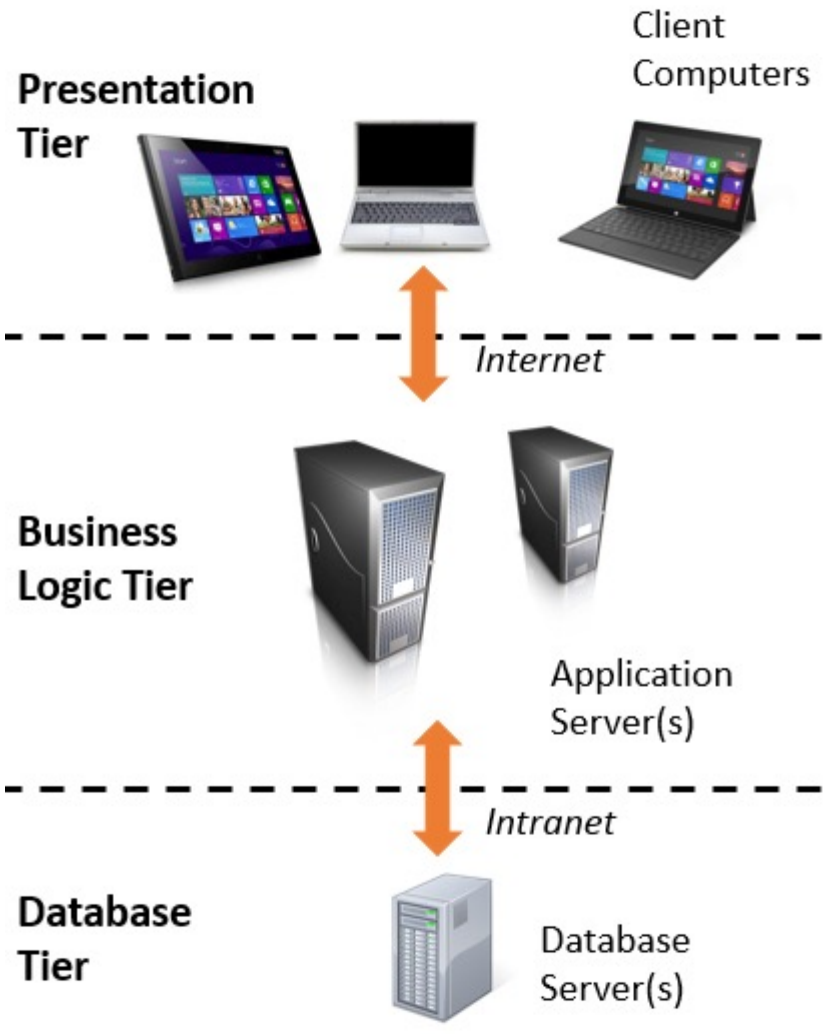
N-Tier Architectural Style



Is a client–server architecture in which presentation, application processing, and data management functions are physically separated

- a *tier* is a physical structuring mechanism for the system infrastructure.
- a tier executes on a *node* (a processing unit)

N-Tier Architectural Style

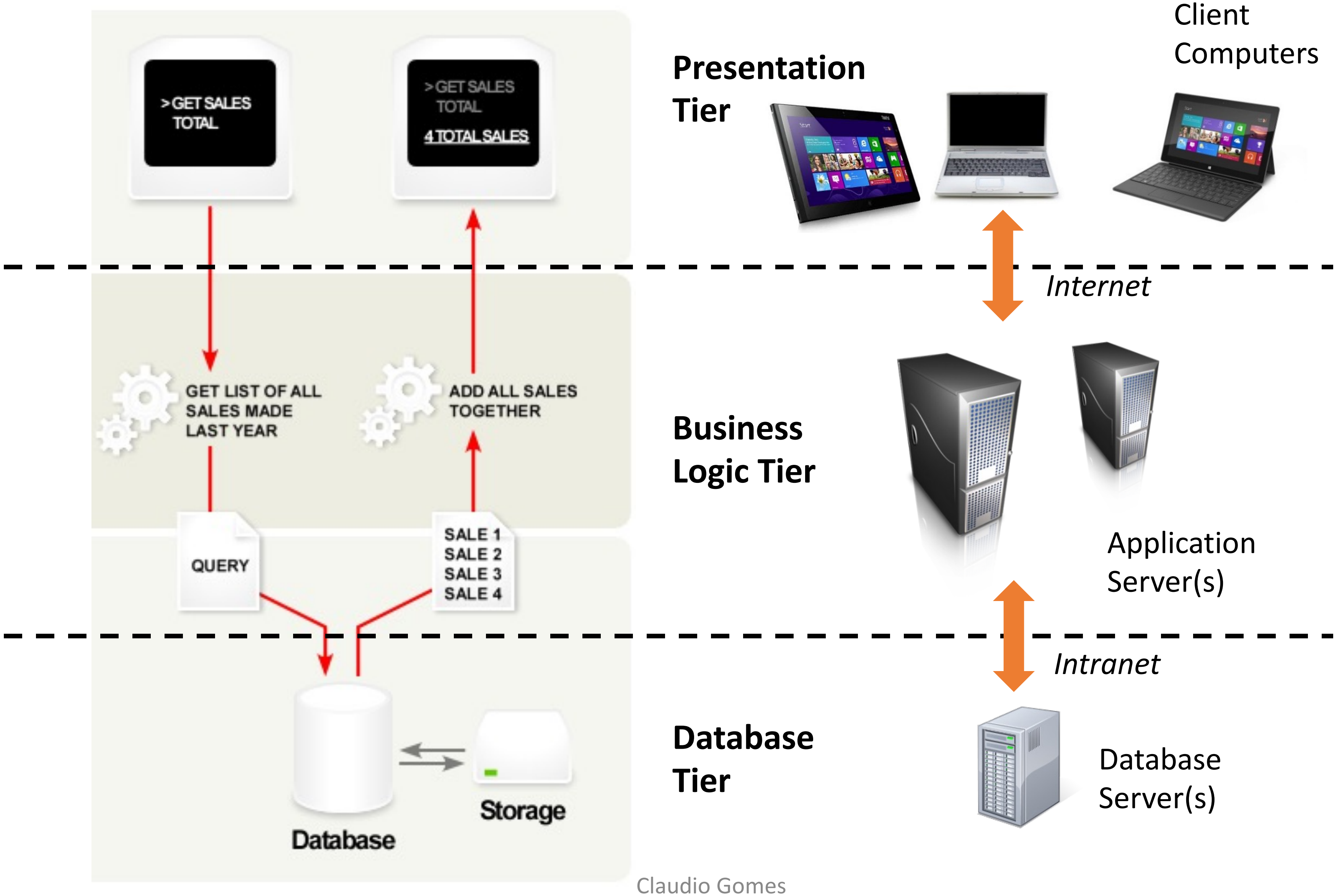


Is a client–server architecture in which presentation, application processing, and data management functions are physically separated

- a *tier* is a physical structuring mechanism for the system infrastructure.
- a tier executes on a *node* (a processing unit)

Important! The software running on a tier may itself consist of multiple layers.

3-Tier Architecture



Presentation Tier

Client Computers

Internet

Business Logic Tier

Application Server(s)

Intranet

Database Tier

Database Server(s)

N-Tier Advantages

- **Scalability**
  - The Application Servers can be deployed on many machines
  - The Database no longer requires a connection from every client.
- **Reusability**
  - E.g. different types of clients (web/mobile/other systems) can connect to the same backend.
- **Data Integrity**
  - The business logic tier can ensure that only valid data is allowed to be updated in the database.
- **Improved Security**
  - Since the client doesn’t have direct access to the database, Data layer is more secure.
  - Business Logic is generally more secure since it is placed on a secured central server.
- **Reduced Distribution**
  - Changes to business logic only need to be updated on application servers and need not to be distributed on clients
- **Improved Availability**
  - Mission Critical Applications can make use of redundant application servers and redundant database servers, so it can recover from network or server failures.

N-Tier Disadvantages

Increased Complexity / Effort

In General N-tier Architecture is more complex to build compared to 2-tier Architecture.

Communication between tiers has to be defined, developed, and evolved.

Servers may crash and communication may fail or be slow.

Security has to be considered at all boundaries and also in the communication.

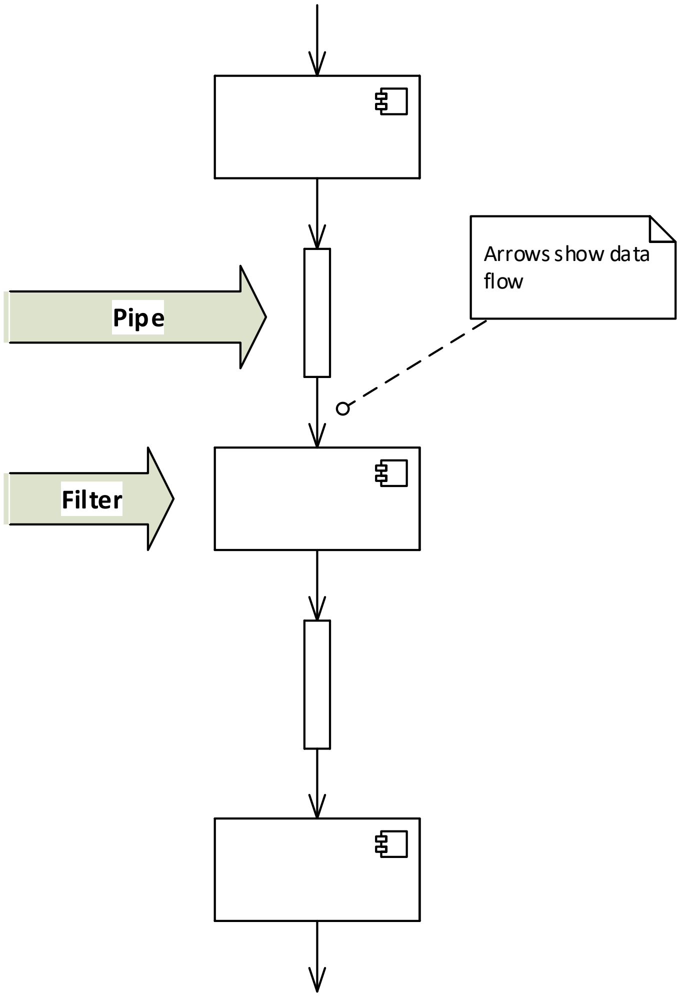
Selected QA for Course

Quality Attribute	Explanation	
Development cost	Reusability. Low complexity – in spite of size	
Development time	Modular. Time to market – work in parallel	
Testability	Test principles – from unit tests to acceptance test	
Maintainability	How easy is it to correct, change and expand	⬅
Availability	Reliability. 99,99… %	⬅
Manageability	How easy is it to operate it on a day to day basis	⬅
Performance	Latency (delay until the first answer) Throughput (how many users can it support)	
Scalability	Can performance grow? Can the system be downgraded? Is it dynamically scalable?	⬅
Security	Protect data (protect against unauthorized reading and writing) Protect the system	⬅
User Experience	User friendliness/experience	

Pipes and Filters

Anecdote

Architectural style: Pipes and Filters



A structure for systems that process a stream of data.

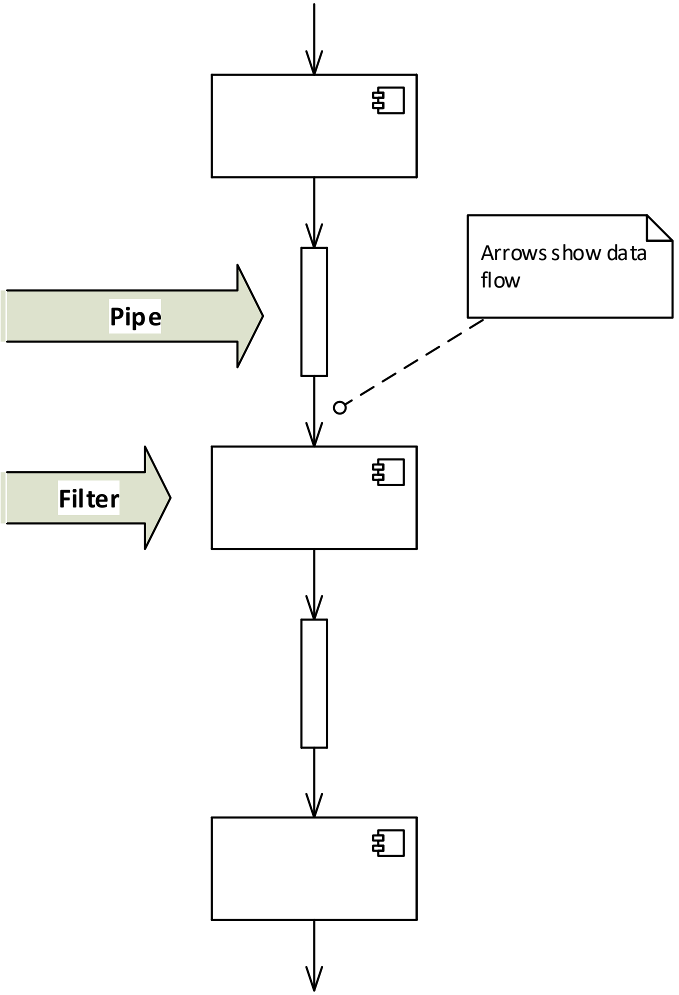
Each processing step is encapsulated in a filter component.

Data is passed through pipes between adjacent filters.

Recombining filters allows you to build families of related systems.

Source: Pattern-Oriented Software Architecture, Vol 1

Architectural style: Pipes and Filters



Implementation:

1. Divide the system's task into a sequence of processing stages.
2. Define the data format to be passed along each pipe.
3. Decide how to implement each pipe connection.
4. Design and implement the filters.
5. Design the error handling.
6. Set up the processing pipeline.

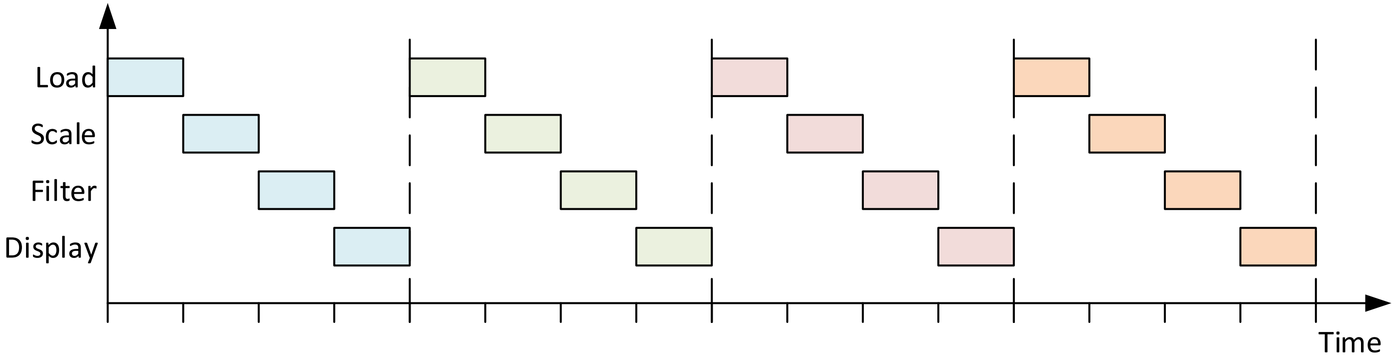
Source: Pattern-Oriented Software Architecture, Vol 1

Architectural style: Pipes and Filters - example

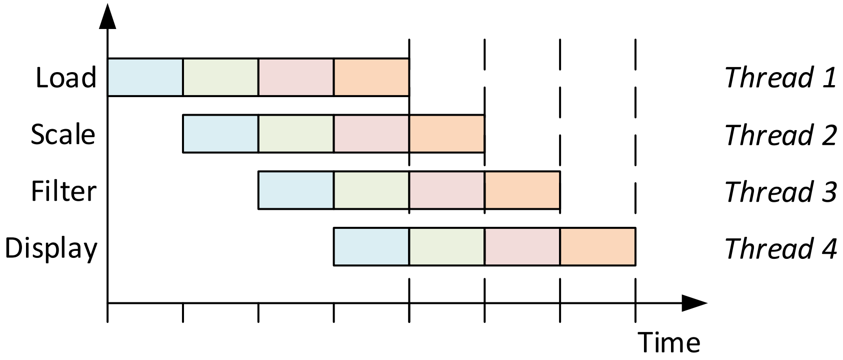
Producing thumbnails of images in stages:

- Load image > scale image > filter image > display image

Sequential
(4 images)

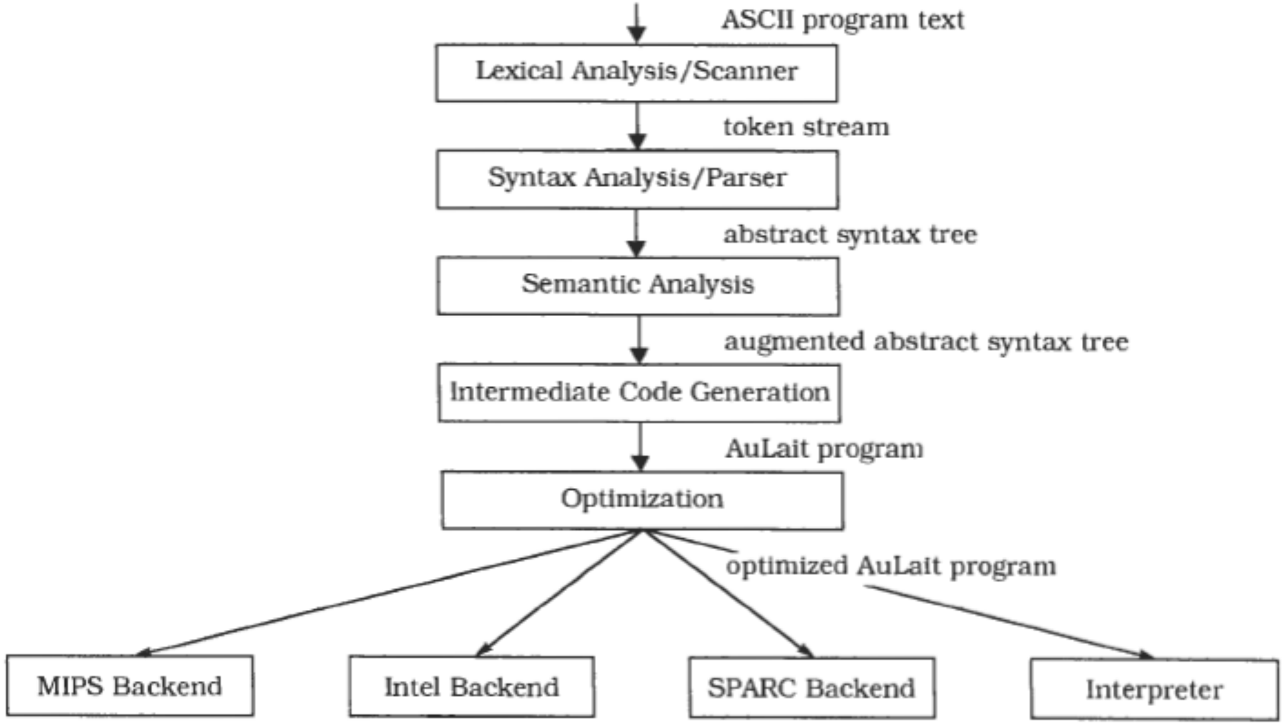


Pipelined –
throughput x 4

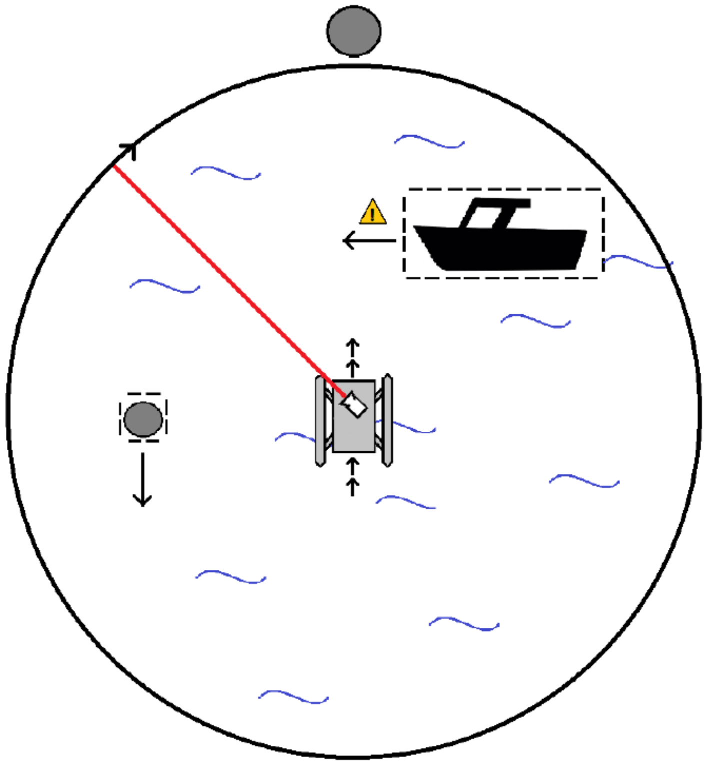


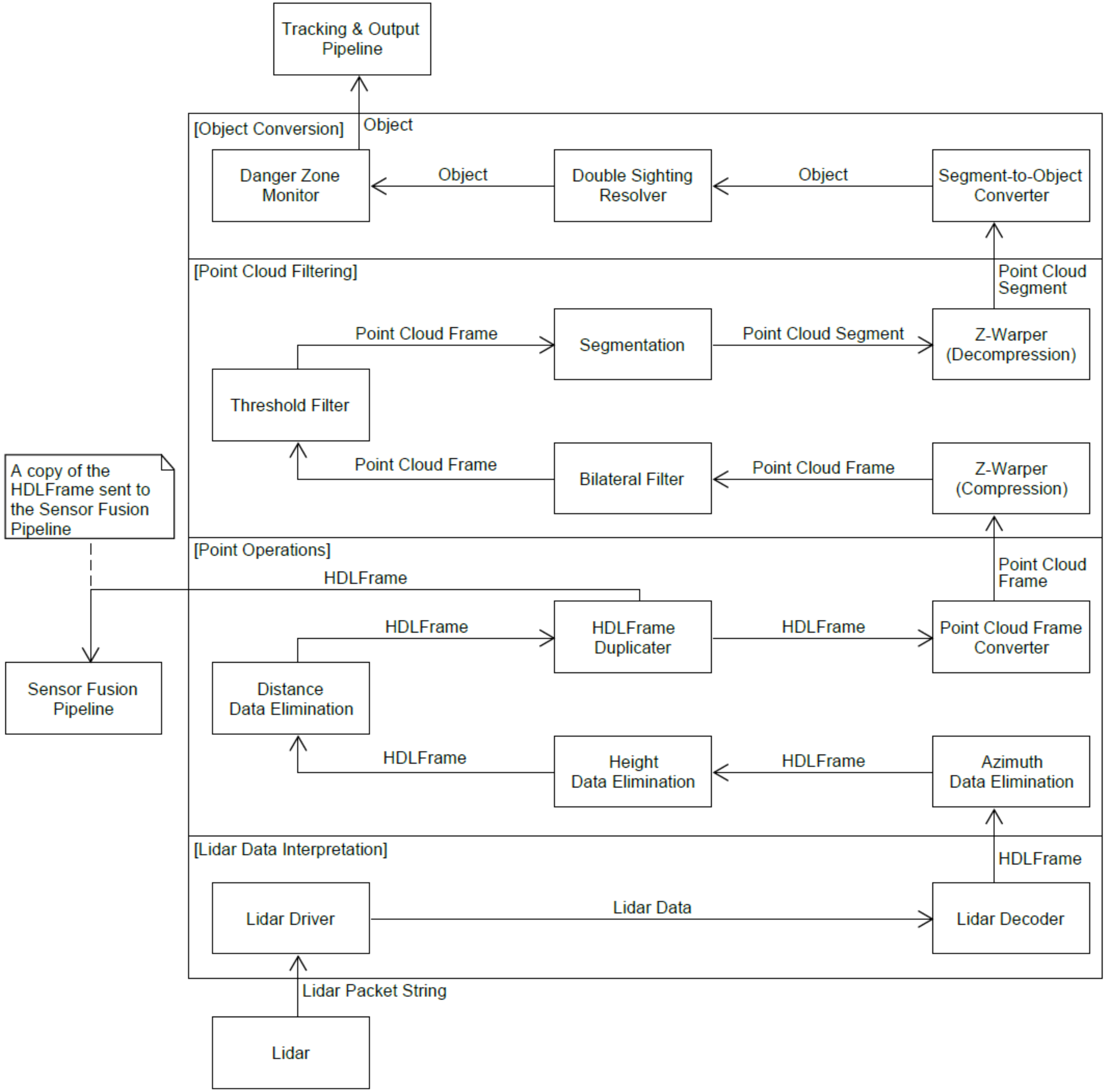
Architectural style: Pipes and Filters - example

Another example could be a compiler.



Source: Pattern-Oriented Software Architecture, Vol 1





Tracking & Output Pipeline

Object

[Object Conversion]

Danger Zone Monitor

Object

Double Sighting Resolver

Object

Segment-to-Object Converter

Point Cloud Segment

[Point Cloud Filtering]

Point Cloud Frame

Segmentation

Point Cloud Segment

Z-Warper (Decompression)

Threshold Filter

Point Cloud Frame

Bilateral Filter

Point Cloud Frame

Z-Warper (Compression)

A copy of the HDLFrame sent to the Sensor Fusion Pipeline

Point Cloud Frame

[Point Operations]

HDLFrame

HDLFrame

HDLFrame Duplicater

HDLFrame

Point Cloud Frame Converter

Sensor Fusion Pipeline

Distance Data Elimination

HDLFrame

Height Data Elimination

HDLFrame

Azimuth Data Elimination

HDLFrame

[Lidar Data Interpretation]

Lidar Driver

Lidar Data

Lidar Decoder

Lidar Packet String

Lidar

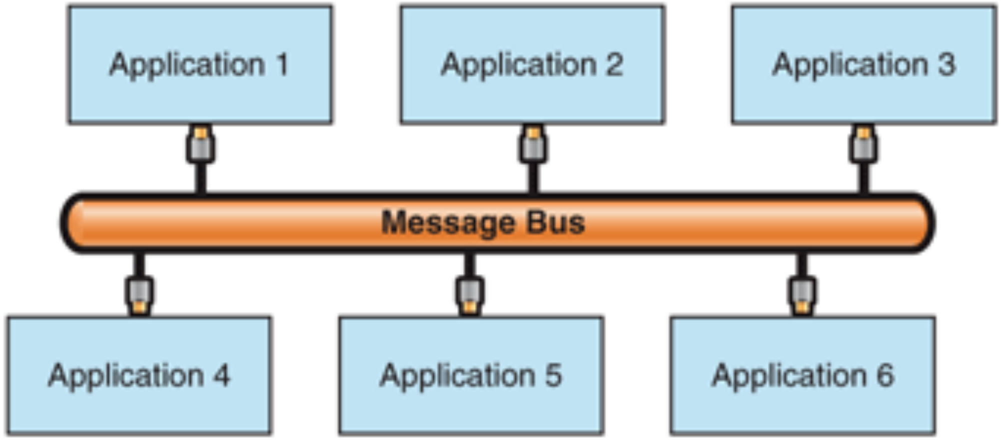
Fall 2018 Bachelor project: ”Object Detection and Tracking on Water Using Lidar and Camera” - Mads Krabsen, Tobias Dalgaard Nielsen, Jonas Kjær Rask

Selected QA for Course

Quality Attribute	Explanation
Development cost	Reusability. Low complexity – in spite of size
Development time	Modular. Time to market – work in parallel
Testability	Test principles – from unit tests to acceptance test
Maintainability	How easy is it to correct, change and expand
Availability	Reliability. 99,99… %
Manageability	How easy is it to operate it on a day to day basis
Performance	Latency (delay until the first answer) Throughput (how many users can it support)
Scalability	Can performance grow? Can the system be downgraded? Is it dynamically scalable?
Security	Protect data (protect against unauthorized reading and writing) Protect the system
User Experience	User friendliness/experience

Message Bus

Architectural style: Message Bus

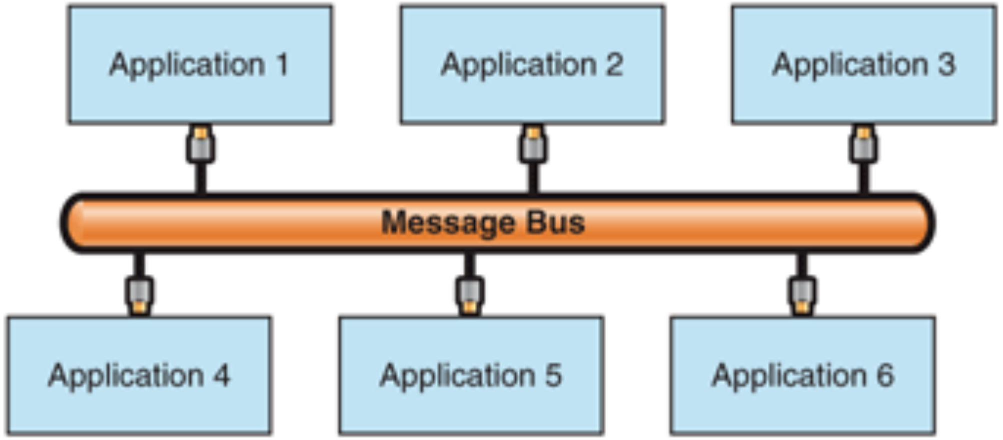


A software system that can receive and send messages using one or more communication channels

So that applications can interact without needing to know specific details about each other.

Source: https://msdn.microsoft.com/en-us/library/ff647328.aspx

Architectural style: Message Bus



Connect all applications through a logical component known as a message bus.

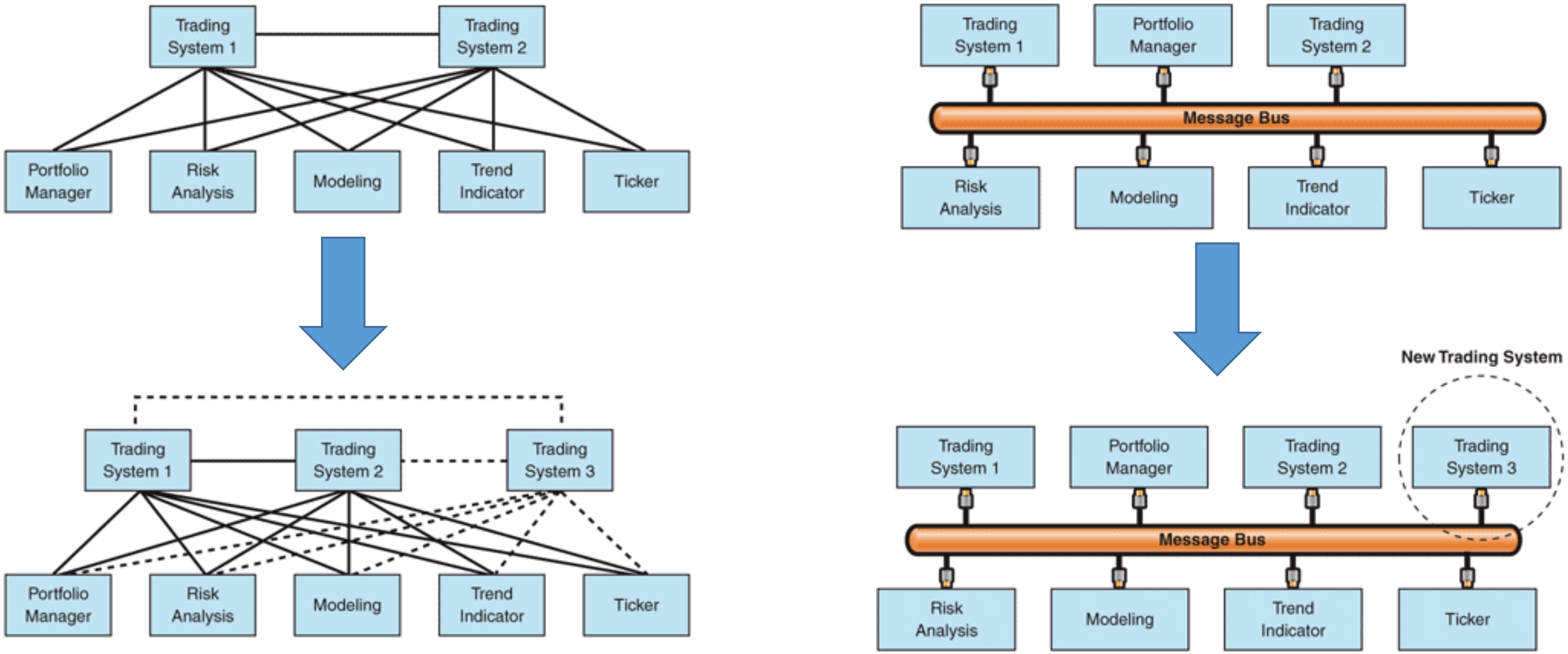
A message bus specializes in transporting messages between applications.

A message bus contains three key elements:

- A set of agreed-upon message schemas
- A set of common command messages
- A shared infrastructure for sending bus messages to recipients

Source: https://msdn.microsoft.com/en-us/library/ff647328.aspx

Architectural style: Message Bus - example



Source: https://msdn.microsoft.com/en-us/library/ff647328.aspx

Selected QA for Course

Quality Attribute	Explanation
Development cost	Reusability. Low complexity – in spite of size
Development time	Modular. Time to market – work in parallel
Testability	Test principles – from unit tests to acceptance test
Maintainability	How easy is it to correct, change and expand
Availability	Reliability. 99,99… %
Manageability	How easy is it to operate it on a day to day basis
Performance	Latency (delay until the first answer) Throughput (how many users can it support)
Scalability	Can performance grow? Can the system be downgraded? Is it dynamically scalable?
Security	Protect data (protect against unauthorized reading and writing) Protect the system
User Experience	User friendliness/experience

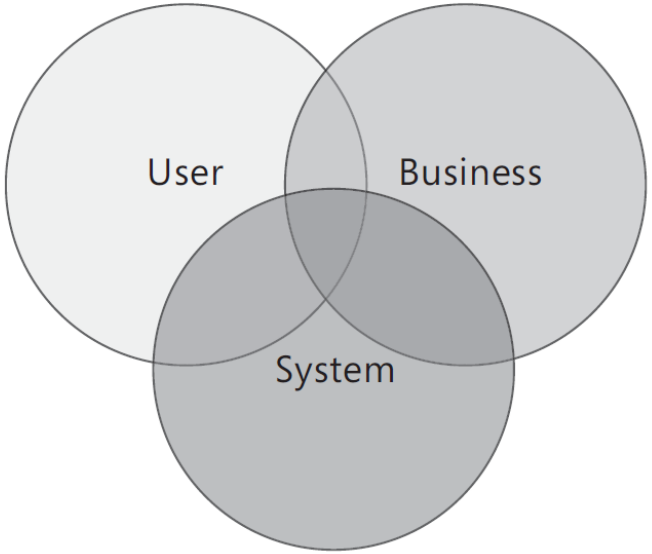
Break

Videoflix key scenarios from Tuesday

As a user

I want to stream videos

In order to be entertained



As business

I want to stream videos to a lot of users

In order to have many customers

As system

I want to share user load on multiple servers

In order to scale to many customers

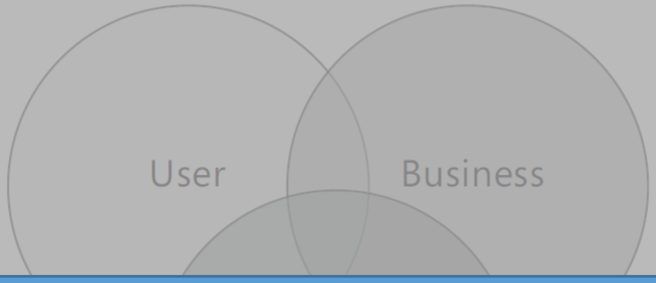
VideoFlix

Videoflix key scenarios from tuesday

As a user

I want to stream videos

In



As business

I want to stream videos to a lot of

ers

Here, the key scenario for the prototype is to stream video
to a lot of users,
in a good-enough quality and to do so,
share the load on multiple servers.

I want to share user load on multiple servers

In order to scale to many customers

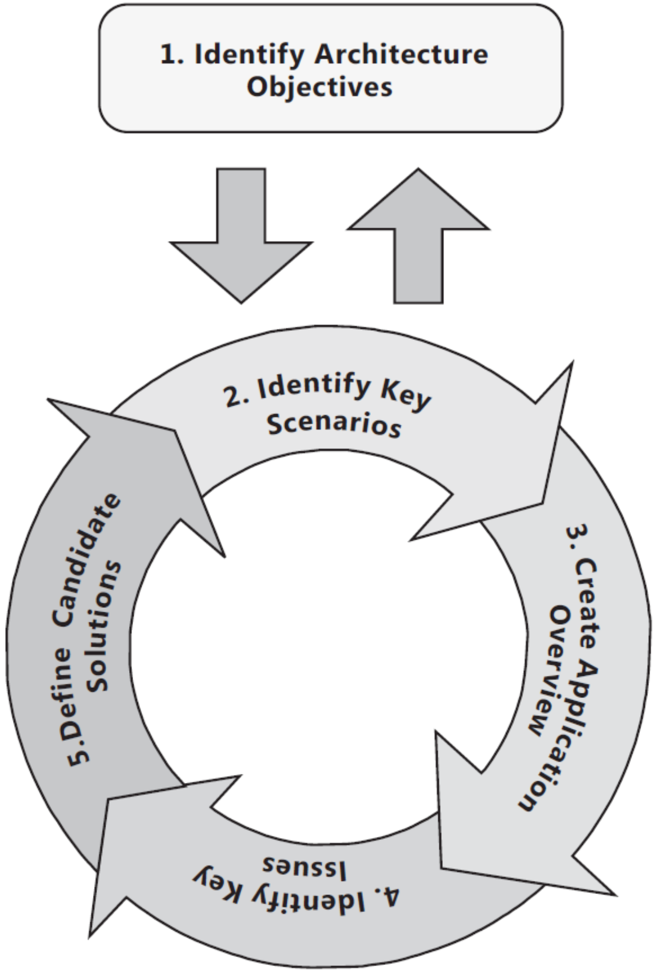
VideoFlix

Videoflix prototype quality attributes

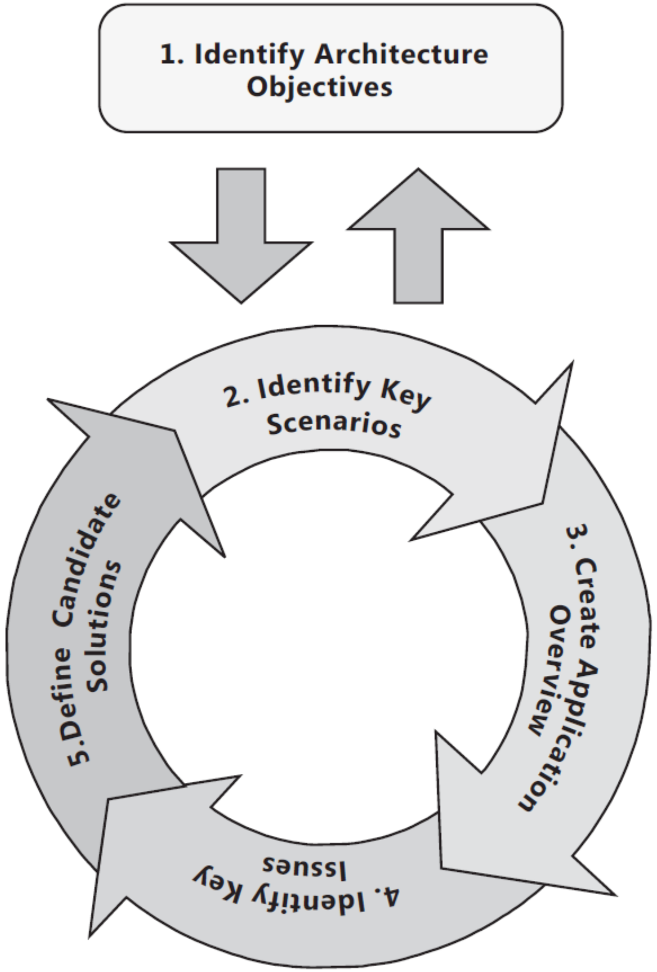
Quality attribute (category)	Non-functional requirement	How to test?
Usability / Performance	Start streaming within 5 seconds of click	
Scalability	100.000 users should be able to start streaming at the same time	
(Compatibility)		
Performance	Start a new server when there is no response to requests in more than 3 seconds.	
(Security)	Movies should be protected	
(Performance)		
(Usability)	Must be learnable within 5 minutes of use of the average user.	

Attributes in parentheses are important but not considered for the prototype.

Next step in the architecture process


1. Identify Architecture Objectives
2. Identify Key Scenarios
3. Create Application Overview
4. Identify Key Issues
5.Define Candidate Solutions

Application overview


1. Identify Architecture Objectives
2. Identify Key Scenarios
3. Create Application Overview
4. Identify Key Issues
5.Define Candidate Solutions

Application type

- mobile, web, service, embedded, ... ?

Deployment constraints

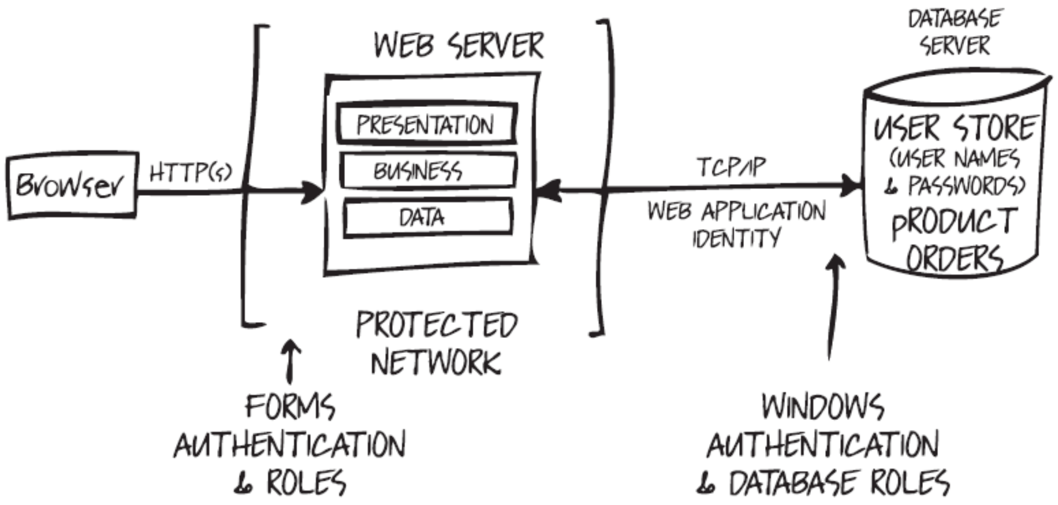
- infrastructure
- quality attributes, consider:
  - security
  - reliability
  - scaleability
  - performance

Architectural style(s)

Technologies

Source: Microsoft Application Architecture Guide, 2nd Edition

Application overview



Source: Microsoft Application Architecture Guide, 2nd Edition

Application type

- mobile, web, service, embedded, ... ?

Deployment constraints

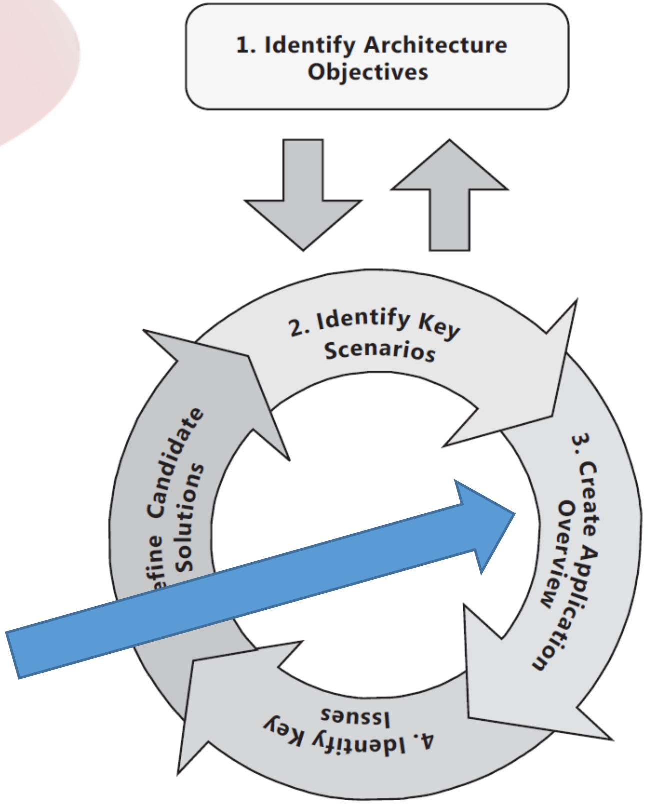
- infrastructure
- quality attributes, consider:
  - security
  - reliability
  - scaleability
  - performance

Architectural style(s)

Technologies

Exercise

VideoFlix


1. Identify Architecture Objectives
2. Identify Key Scenarios
3. Create Application Overview
4. Identify Key Issues
efine Candidate Solutions

Exercise 5:

Create an application overview for the prototype.

Send your suggestion to henrik@ece.au.dk

AARHUS UNIVERSITY

References and image sources

Video(s):

Martin Fowler - Making architecture matter: https://www.youtube.com/watch?v=DngAZyWMGR0

Simon Brown - The Frustrated Architect:

https://www.infoq.com/presentations/The-Frustrated-Architect