University of Tripoli – Faculty of Information Technology

Software Engineering Department

# Software Architecture & Design

ITSE411

y Marwa Solla

# Software architecture and design

## for modern large-scale systems

Lecture 2 :

System Requirements & Architectural Drivers

# What We Learn In This Lecture

- Types of Designer Roles
- Software development Cycle
- System Requirements & Architectural Drivers

# Design vs. Architecture

Types of Designer Roles

- **User Interface (UI) designer**
- **Application domain designer**
- **Data designer**

# Design vs. Architecture

- **User Interface (UI) designer.**

  UI design is out of the scope. You can refer to Human– Computer–Interaction (HCI) Course for more extra details related this topic.

# Design vs. Architecture

- **Application domain designer.**
  - Domain designer is a major key player in the process of architecting and designing an application.
  - He is responsible of understanding the detailed business rules, and the main domain objects that are generated from this set of customer's business rule requirements.
  - He designs the application components that have enough algorithms to maintain these requirements processed, and implemented accurately.

# Design vs. Architecture

- **Data designer.**

  Those are a kind of database (DB) professionals who are concerned with designing the applications DB, and  defining its chosen schema; logically and  physically.

# Design vs. Architecture

**Order of architecture & design within the SW development lifecycle**

- Most of the required information by Architect and Designer are Requirements that come from Analysis, and scoping constraints that come from Planning.
- In waterfall process model, Architecture, and design comes after Analysis.
- In agile iterative based processes, some architectural and design questions may lead to getting back to analysis thus, changing the final requirements upon received answers.

# Modeling and Documenting

- Without documenting the decisions of architecture and design we will be risking lots of aspect including but not limited to:
  - The implementation.
  - The maintainability.

  - For OO, UML is the documentation standard.
  - UML is an ISO standard # ISO/IEC 19505

# Challenges of Software Architecture

We **cannot** prove Software Architecture to be either:

- Correct .
- Optimal.

What we **can** do to guarantee success is follow:

- Methodical design process
- Architectural patterns
- Best practices.

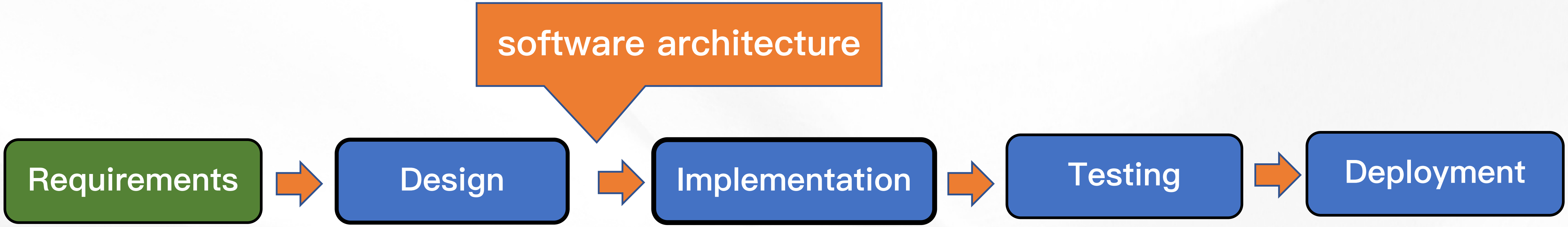
# Software Architecture Process

- ➢ On a larger scale, the process for creating software architectures can be executed using the following tasks:
  - ✓ Understand and evaluate requirements
  - ✓ Design the architecture
  - ✓ Evaluate the architecture
  - ✓ Document the architecture
  - ✓ Monitor and control implementation

# Software Architecture Process

- ➢ Software architects spend a great deal of time working with software requirements.
  - ✓ Even after requirements are specified, software architects find themselves going back and forth between requirements and design.
  - ✓ In some cases, architects are completely immersed in the requirements phase, playing a key role in the specification of requirements.

# Software Development Cycle

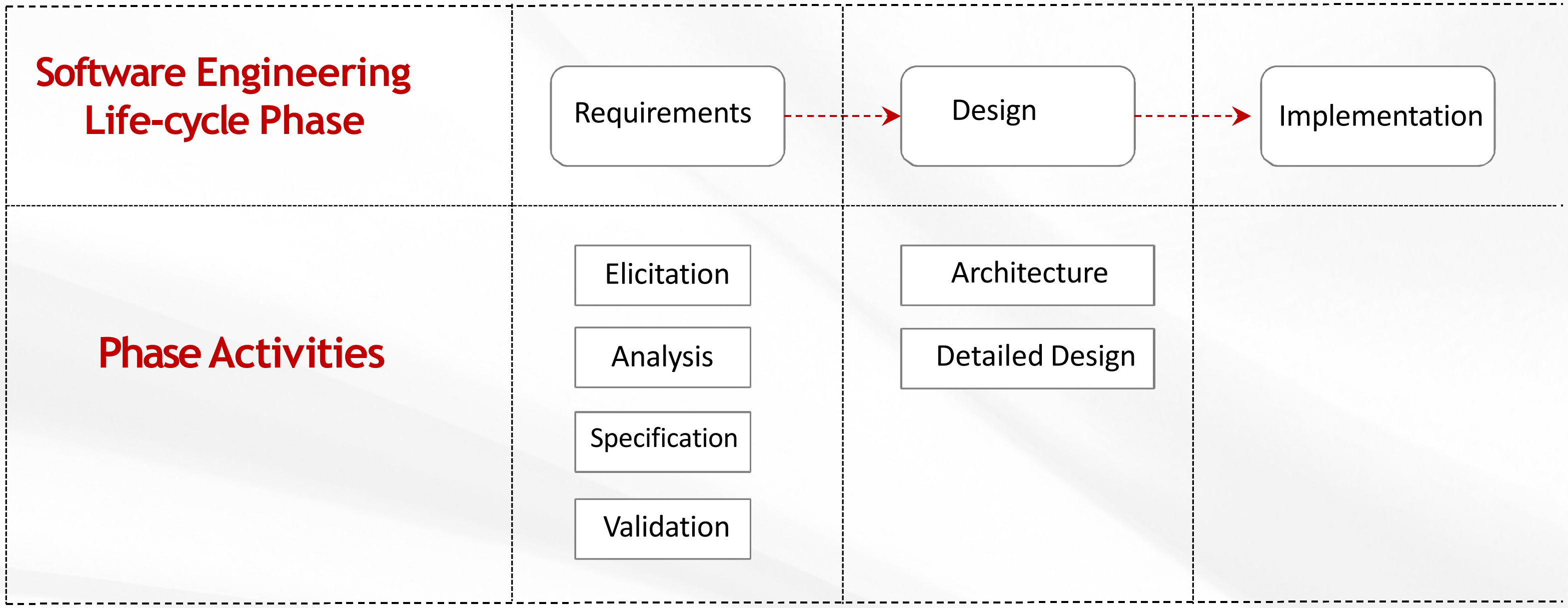


Notes:

- Understanding customer's requirements is a key factor for having any good design.
- software architecture **is** the output of the design phase and the input to our systems implementation.

# Software Development Cycle

✓Similar to the design phase, the requirements phase can be broken down into well defined activities



| Software Engineering Life-cycle Phase | Requirements | Design | Implementation |
|---|---|---|---|
| Phase Activities | Elicitation<br>Analysis<br>Specification<br>Validation | Architecture<br>Detailed Design | |

# System requirements and Architectural drivers

*Requirements – Motivation*

*Requirements Classification*

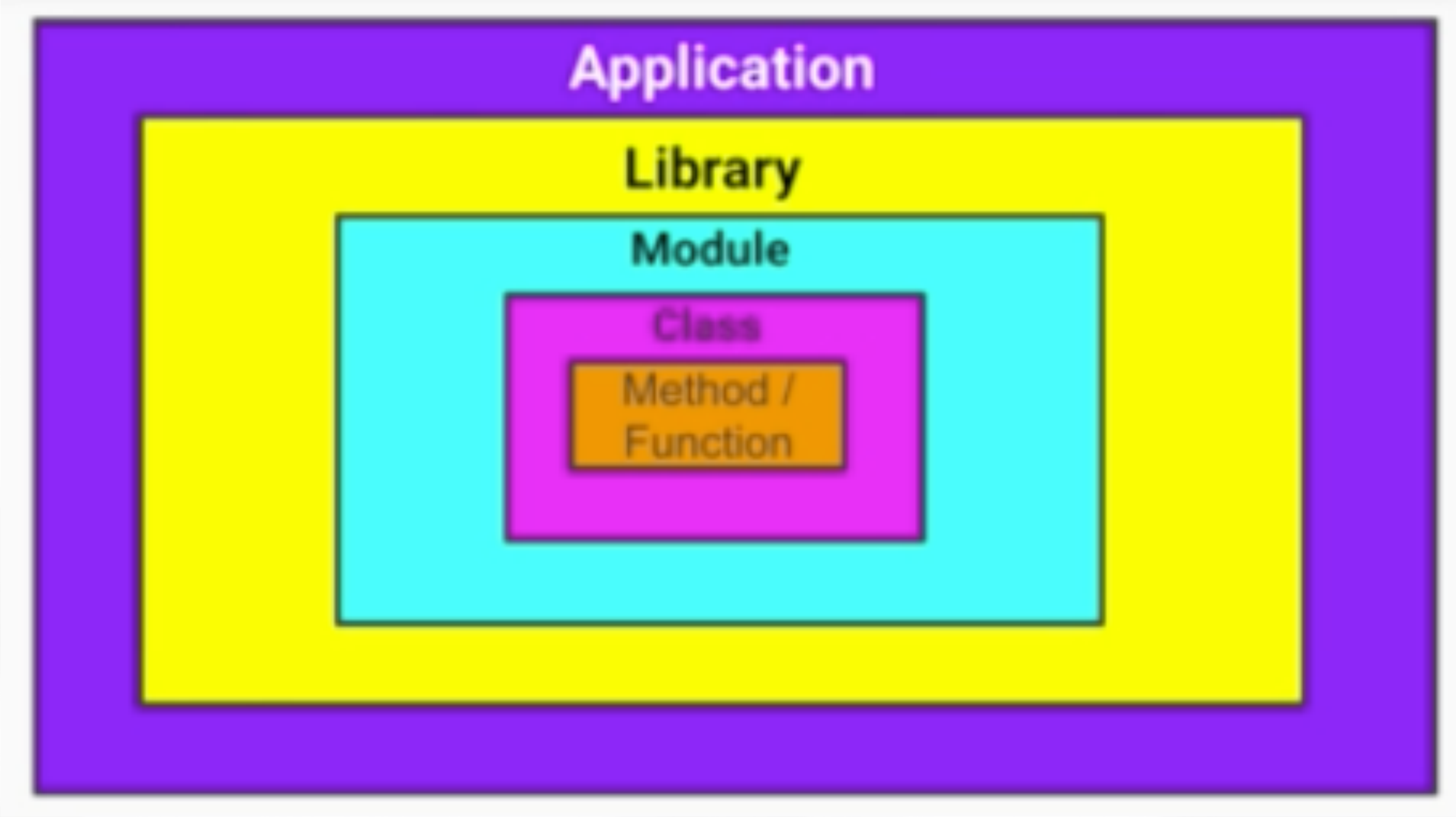
## What are system Requirements ?

***Requirements*** ：Formal description of what we need to build.

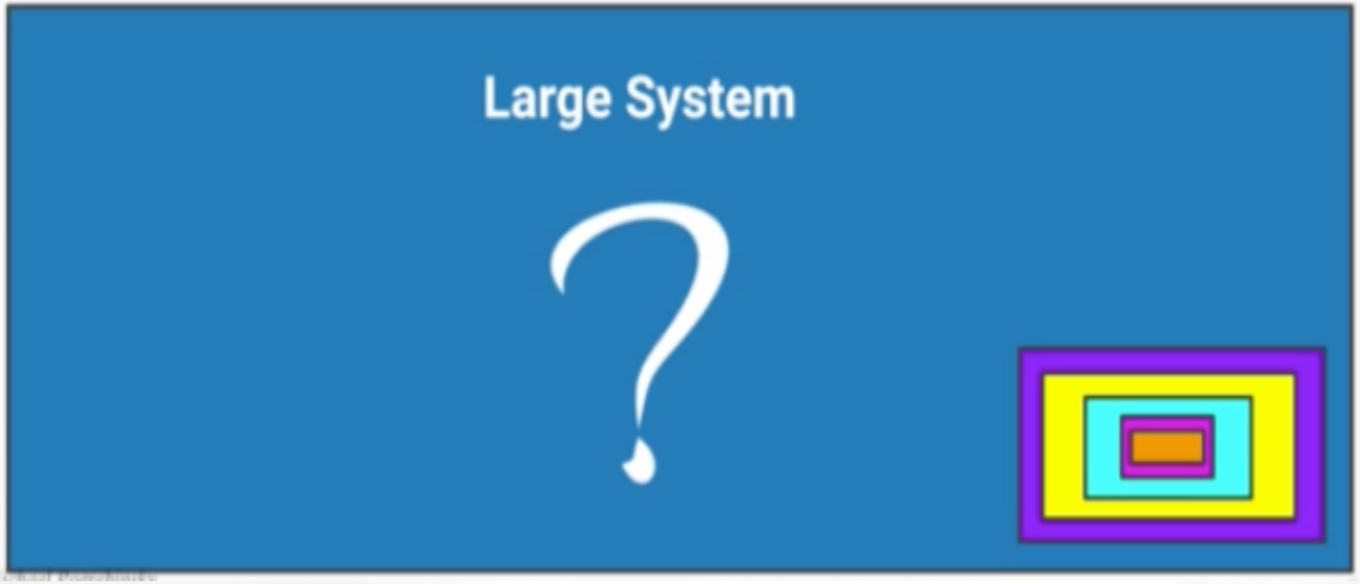
large scale system requirements are a few differences from the usual requirements we typically get for implementing:

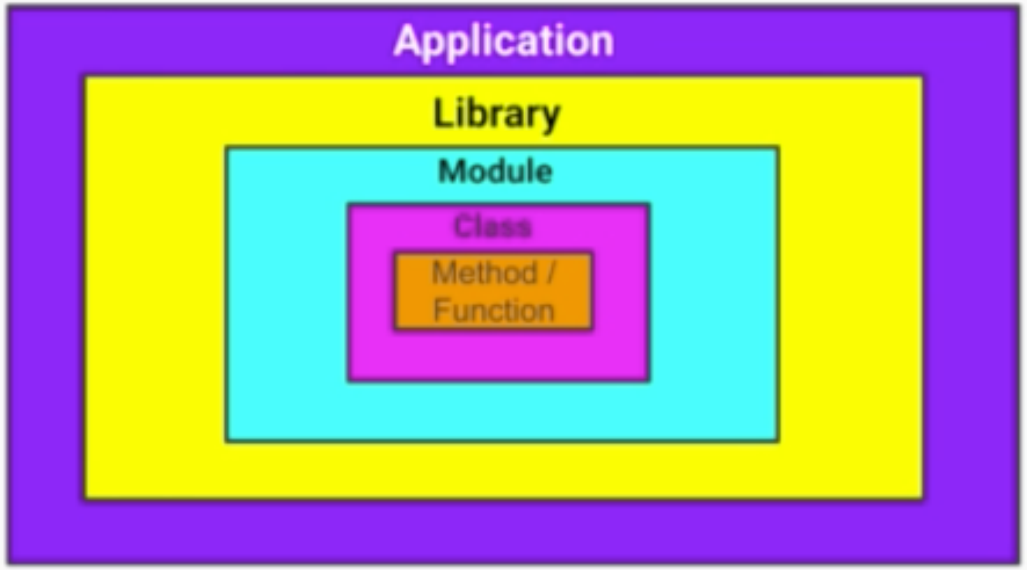
- Method
- Algorithm
- Class

# Big Scope and High level of Abstraction



# Big Scope and High level of Abstraction





# Examples of Scope and Abstraction

- File storage system,
- Video  streaming solution
- Ride sharing service,

# High level of Ambiguity

- System Design has high level of ambiguity
- Two reasons:
  - The person providing the requirements is often not an engineer and may even be not vey technical.
  - Getting the requirements is a part of the solution
    - ✓ The client doesn’t always know what they need
    - ✓ The client generally knows only what problem they need solved.

# Example: Hitchhiking Service

High Level Requirement – “ Allow people to join drivers on a route, who are willing to take passengers for a fee”

Clarifying questions:

- Real time vs advance reservation
- User Experience– Mobile ? Desktop? Both?
- Payment through us or direct payment?

# Importance Of Gathering Requirements

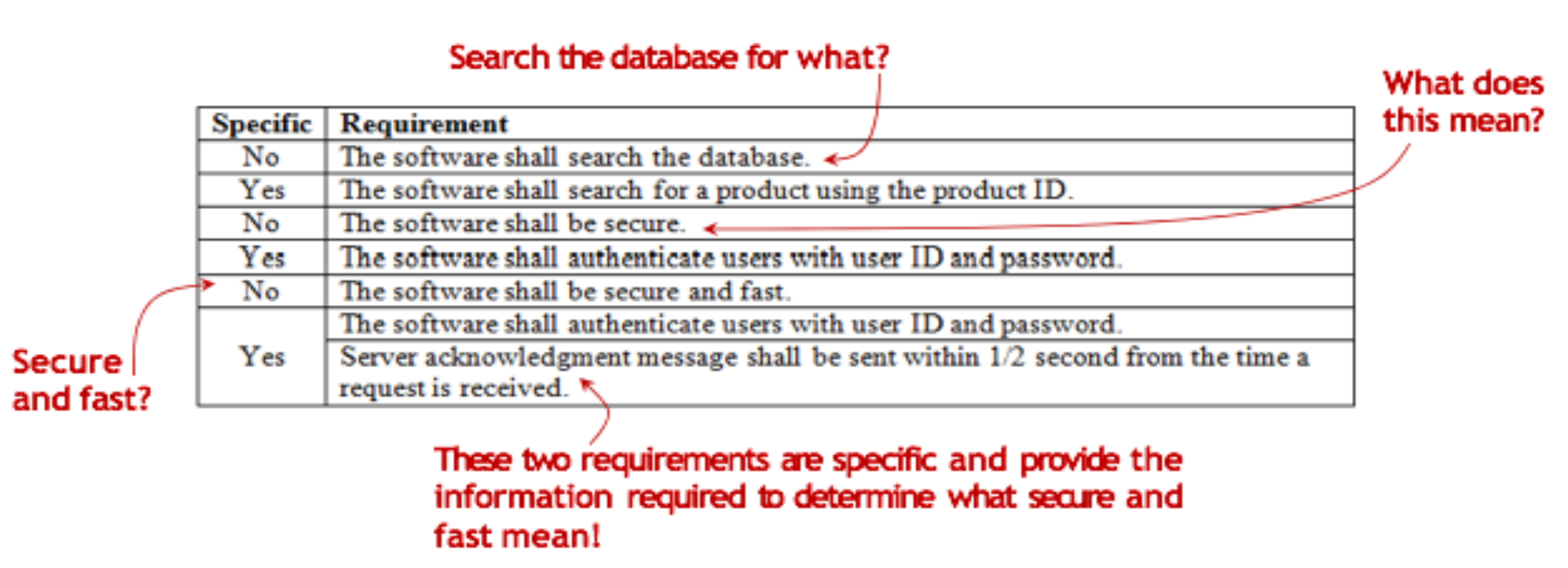
- What happens if we don't get the requirements right?
- We can simply build something and then fix it
- Seemingly there's no cost of materials in software so changes should be cheap??

# Importance Of Gathering Requirements

- Large scale systems are big projects that cannot be easily changed
  - ➢ Many engineers are involved
  - ➢ Many months of engineering work
  - ➢ Hardware and Software costs
- Contracts include financial obligations
- Reputation and brand

# Translates software requirements

- To create design elements from requirements, it is assumed that requirements are understood. Sometimes this is not the case.
  - *For examples:*



Search the database for what?

What does this mean?

| Specific | Requirement |
|---|---|
| No | The software shall search the database. |
| Yes | The software shall search for a product using the product ID. |
| No | The software shall be secure. |
| Yes | The software shall authenticate users with user ID and password. |
| No | The software shall be secure and fast. |
| Yes | The software shall authenticate users with user ID and password. |
|  | Server acknowledgment message shall be sent within 1/2 second from the time a request is received. |

Secure and fast?

These two requirements are specific and provide the information required to determine what secure and fast mean!

# Translates software requirements

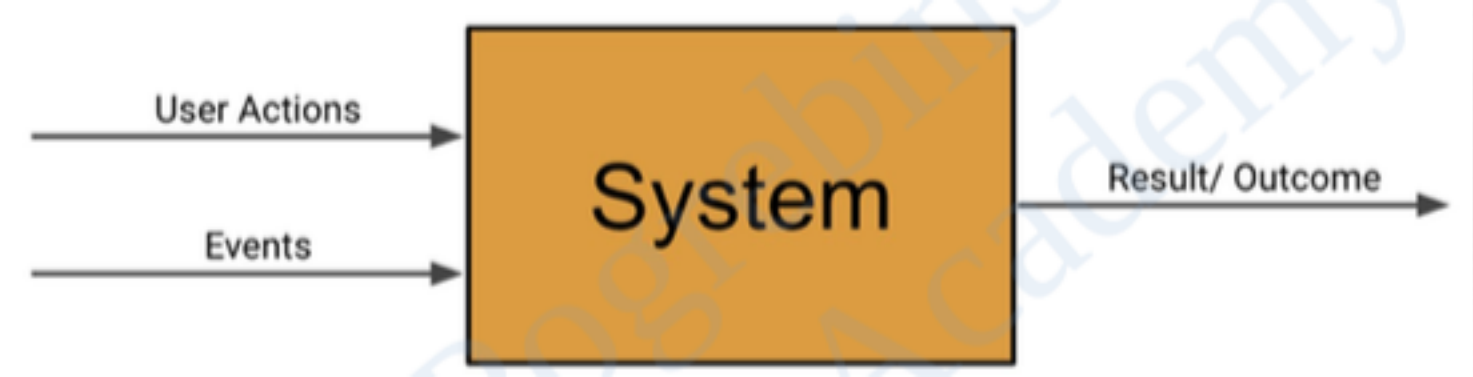
- So, some may think that “*The system shall perform fast.*” specifies a requirement that can be used to create design elements.
- Such statements create problems for designers. These problems need to be resolved before we can translate from requirement to design domain.

# Types of Requirements

- **Features of the System**
  - o Functional requirements
- **Quality Attributes**
  - o Non–Functional requirements
- **System Constraints**
  - o Limitations and boundaries

# Features / Functional Requirements

- Describe the system behavior – what "the system must do"
- Easily tied to the objective of our system



# / Functional Requirements

- Features / Functional Requirements
- • Functional requirements do not determine its system architecture
- Generally, any architecture can achieve any feature

# Functional Requirements/ Examples

"When a rider logs into the service mobile app, **the system must** display a map with nearby drivers within 5 miles radius”

"When a rider is completed, **the system will** charge the rider's credit card and credit the driver, minus service fees"

# Functional Requirements/ Examples

Hitchhiking Service – Example:

- Rider first time registration
- Driver registration
- Rider login
- Driver login
- Successful match and ride
- Unsuccessful ride

# Quality Attributes– Non Functional Requirements

Quality Attributes / Non-Functional Requirements System properties that "the system much have"

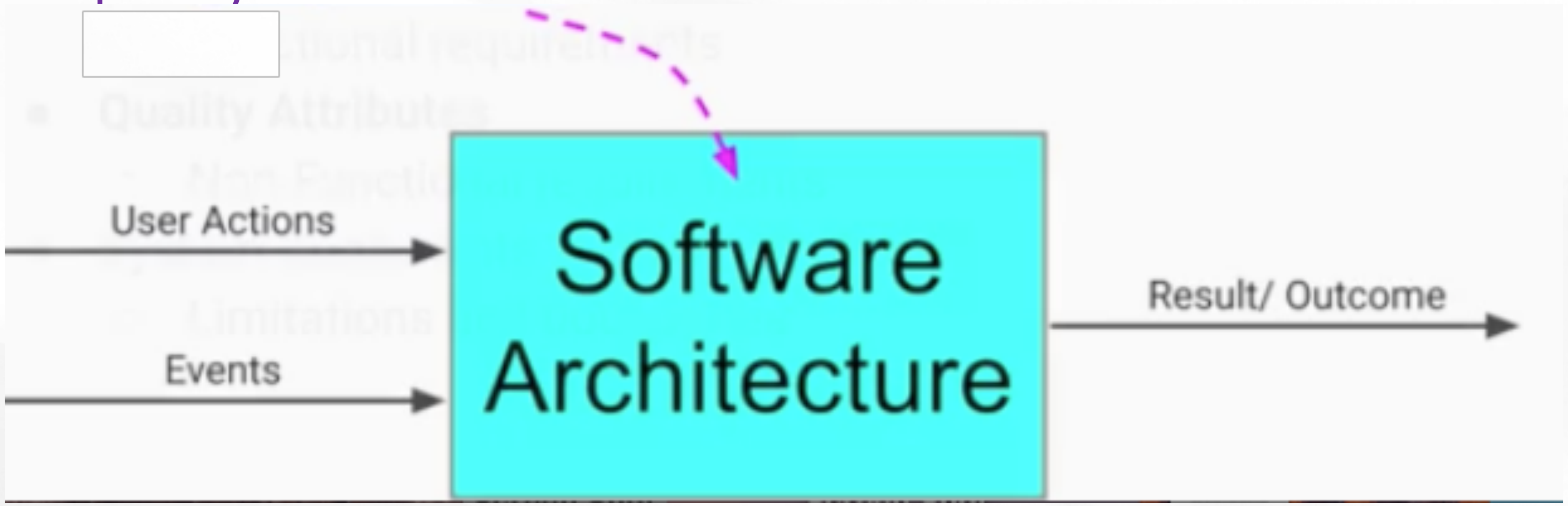
Examples:

- Scalability
- Availability
- Reliability
- Security
- Performance

# Non Functional Requirements

Quality attributes and Software Architecture

- The quality attributes dictate the software architecture of our system



# Quality Attributes – Motivation

Systems are frequently redesigned NOT because of functional requirements

- But because the system as it stands:

o Isn't fast enough

o Doesn't scale

o Slow to develop

o Hard to maintain

o Not secure enough

# System Quality Attributes – Definition

Quality attributes are non functional requirements, They describe

- The qualities of the functional requirements
- The overall properties of the system
- Provide a quality measure on how well our system performs on a particular dimension.
- They have direct correlation with the architecture of our system

# The Quality Attribute Example – Online Store 1

"when a user clicks on a search button after they typed in a particular search keywords, the user will be provided with a list of products that closely match the search keyword within at most a 100 milliseconds."

# The Quality Attribute Example – Online Store 1

*Functional Requirement*

"when a user clicks on a search button after they typed in a particular search keywords, the user will be provided with a list of products that closely match the search keyword within at most a 100 milliseconds."

*Performance Quality Attribute*

# The Quality Attribute Example – Online Store 2

The online store must be available to user a requests at least 99.9% of time

*Availability Quality Attribute*

# 1. Important Considerations – Testability and Measurability

Quality Attribute–Example

- Quality attributes need to be:

o **Measurable**

o **Testable**

If we can prove that our system satisfied the required the quality attribute we don't know if our system performs well or poorly

# Unmeasurable Quality Attribute – Example

"When I user clicks on the buy button, the purchase confirmation must be displayed *quickly* to the user"

# 2. Important Considerations – Tradeoffs

- No single software architecture can provide all the quality attributes.
- Certain quality attributes contradict one another
- Some combinations of quality attributes are very hard / impossible to achieve
- We (Software Architects) need to make the right tradeoff.

# Trade Off–Login Page Example

1. Performance – Login Time < 1 second

2. Security Username, Password, SSL

*Slower*

# 3. Important Considerations – Feasibility

- We need to make sure that the system is capable of delivering with the client asking for.
- The client may ask for something that is either

o Technically impossible

o Expensive to implement.

# Feasibility Examples – 100% Availability

- Our system can never fail
- We never have a chance to take our system down for
  - ✓ Maintenance
  - ✓ Upgrade
- Full protection against hackers
- High resolution video streaming in limited bandwidth areas
- Very high storage growth

# System Constraints

Once we define what our system must do, we have freedom on how to structure our system

- While defining the final architecture, we have to make a lot of decisions
- For quality attributes, we are expected to make trade–offs

- **System Constraints – Definition**
- "A system constraint is essentially a decision that was already either fully or partially made for us, restricting our degrees of freedom."

# System Constraints

- Instead of looking at a constraint as a choice that was taken away, we look at it as a decision that was already made
- System Constraints are referred as pillars for software architecture because:

o They provide us with a solid starting point

o The rest of the system need to be designed around them

# System Constraints/Examples

- Time Constraints – Strict deadlines
- Financial Constraints – Limited budget
- Staffing Constraints – Small number of available engineers

# Types of Constraints

There are three types of constraints:

❑ Technical constraints

❑ Business constraints

❑ Regulatory/legal constraints

# Technical Constraints

- Examples of technical constraints include:
  - ✓ Being locked to a particular **hardware/cloud vendor**
  - ✓ Having to use a particular **programming language**
  - ✓ Having to use a particular **database or technology**
  - ✓ Having to support certain **platforms, browsers, or OS**

# Technical Constraints

- Technical constraints may seem like they belong to implementation and not to software architecture
- In practice, they affect the decisions we make in the design phase and put restrictions on our architecture.
- Example 1

➢ If our company makes a decision to run on–premise data centers then:

✓ All the cloud architectures and paradigms will become unavailable to us

# Technical Constraints

- **Example 2:**

If we have to support some older browsers or low–end mobile devices then:

o We have to adapt our architecture to support those platforms and their APIs

o Keep providing a different, more high–end experience for newer browsers or higher–end devices

# Business Constraints

- As engineers, we make the right decisions and architectural choices from a technical perspective
- This forces us to make sacrifices in:
  - Architecture
  - Implementation

# Business Constraints – Examples

- Limited budget or a strict deadline will make us have very different choices than if we had an unlimited budget and unlimited time
- Different software architectural patterns are based on suitability between small startups or bigger organizations.
- Usage of third-party services with their own APIs and architectural paradigms as part of our architecture
  - ✓ Using third-party shipping/billing providers for an online store
  - ✓ Integration of different banks/brokers/security/fraud detection services for an investing platform

# Regulatory/Legal Constraints

Regulatory constraints may be:

- Global
- Specific to a region

Examples:

✓ In the US, HIPAA (Health Insurance Portability and Accountability Act) places constraints on accessing patients' data

✓ In the European Union, GDPR (General Data Protection Regulation) sets limitations on collecting, storing and sharing users' data

# Types of Requirements

- **Features of the System**
  - o Functional requirements
- **Quality Attributes**
  - o Non-Functional requirements
- **System Constraints**
  - o Limitations and boundaries

Architectural Drivers

# Summary

- We got the motivation for quality attributes
- Quality attribute definition: "Quality measure on how well our system performs on a particular dimension"
- 3 important considerations:
  - o Testability and Measurability
  - o Trade offs
  - o Feasibility
- The 3rd type of architectural driver, the System Constraints. "Decision that was already either fully or partially made for us, restricting our degrees of freedom".

# Summary

❑ Three types:

1) Technical Constraints
2) Business Constraints
3) The legal constraints

# Exercises

Question 1:We received the following requirement from the client:

***"We would like you to build a system that allows sharing of large files between users.***

***After a user uploads a file, they will get a unique link that they can share with other users. Any user with that link can download the file.***

***The link should become active no later than 1 second after the file is uploaded.***

***Download speeds should be at least 50 Mbit/sec.***

***You have to support at least PDF and JPG file formats, as well as the following web browsers: Google Chrome, Mozilla Firefox, and Microsoft Edge."***

# Exercises

**Which part is the non-functional / Quality Attributes requirement?**

❑ After a user uploads a file they will get a unique link that they can share with other users. Any user with that link can download the file.

❑ The link should become active no later than 1 second after the file was uploaded. Download speeds should be at least 50 Mbit/sec.

❑ You have to support at least PDF, and JPG file formats, as well as the following web browsers: Google Chrome, Mozilla Firefox, Microsoft Edge."

❑ All of It

# The End