Software Engineering

A Faculty of Engineering Course: CSEN 303

Software Architectures

9

Dr. Iman Awaad

iman.awaad@giu-uni.de

GERMAN INTERNATIONAL UNIVERSITY
الجامعة الألمانية الدولية

Acknowledgments

They are **heavily** based on the slides and textbook by **Ian Somerville**.

The slides are also based on the **slides** by **Prof. Dr. John Zaki**.

Their contribution is gratefully acknowledged.

Any additional sources are referenced.

Software Architecture

- Architecture
- Characteristics of good architectures
- Patterns
  - Client-server
  - Model-View-Controller
  - Layered
  -

How do we organise our system into components and how do they fit together?

What are characteristics of a well-designed system?

Your design decisions matter!

Your goal: Create a **reliable**, **secure** and **efficient** product

Pay attention to **architectural design**:
its overall **organization**
how the software is **decomposed into components**
the **server organization**
the **technologies** that you use to build the software.

The **architecture** of a software product affects
its **performance**, **usability**, **security**, **reliability** and **maintainability**!

Abstraction levels

When working on a complex system *e.g.* a robot, must understand the components of that system and the manner in which they interact with each other

Otherwise, difficult to keep track of the system's design and evolution

An architecture: a model that describes how a system works and what it needs

It abstracts away various aspects of the system's operation

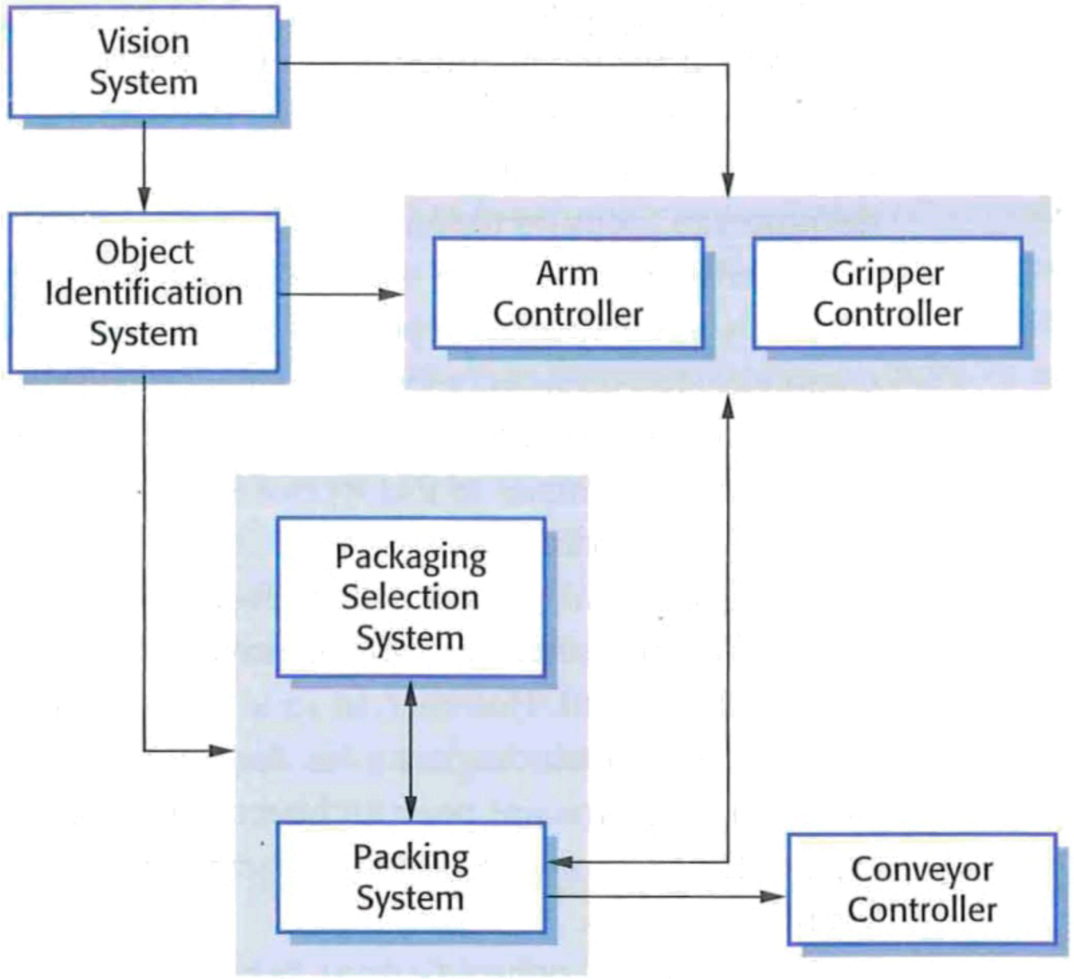
1. **Operational architecture:** Specifies what a system should do without describing how

2. **System architecture:** Defines the components of a system and their connections

3. **Technical architecture:** Describes how the system actually works at an implementation level (down to the algorithmic level)

Example: Robot architectures

Various operational architectures used in robotics, such as sense-plan-act, behaviour-based, three-tier, hybrid, and cognitive architectures…



What is a **Software Architecture**?

… is the fundamental **organization** of a software system embodied in its **components**, their **relationships to each other** and **to the environment**, and the **principles** guiding its design and evolution.

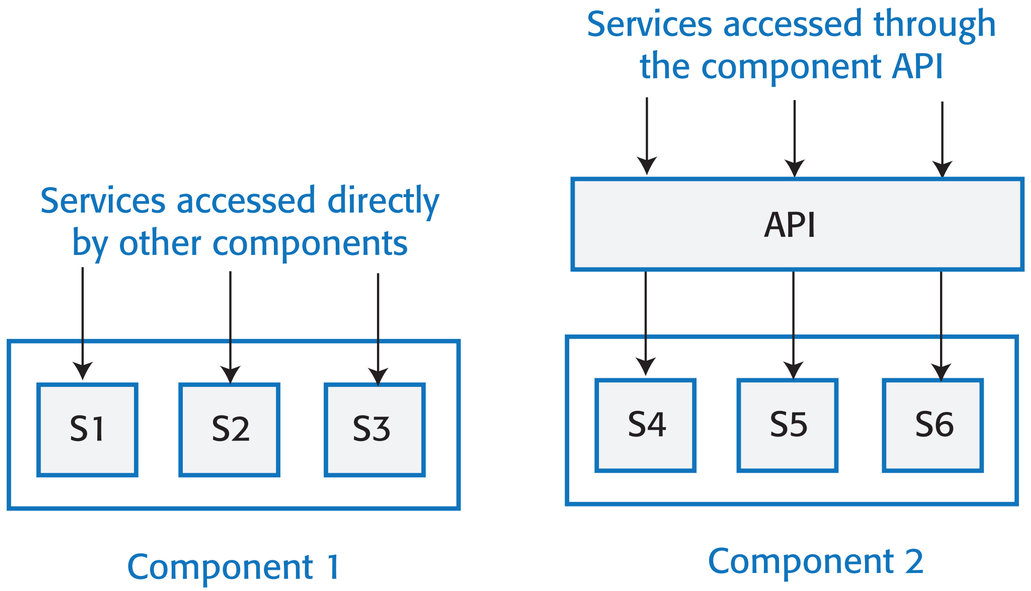
What is a **Component**?

… is an element that implements a coherent set of functionality or features; …a collection of one or more services that may be used by other components

When designing an architecture: you design the **component interface** and **leave the implementation** of that interface **to a later stage** of the development process.

What is a **Component**?

Figure 4.1 Access to services provided by software components



When designing an architecture: you design the **component interface** and **leave the implementation** of that interface **to a later stage** of the development process.

Why is this topic important?

The architecture has a fundamental influence on the **non-functional system properties**!

(Remember the **SCARIF VAULT**… security is a **non-functional requirement** (**NFR**))

Designing your architecture: understand issues that affect the architecture and create an architectural description that shows the **critical components** and their **relationships**.

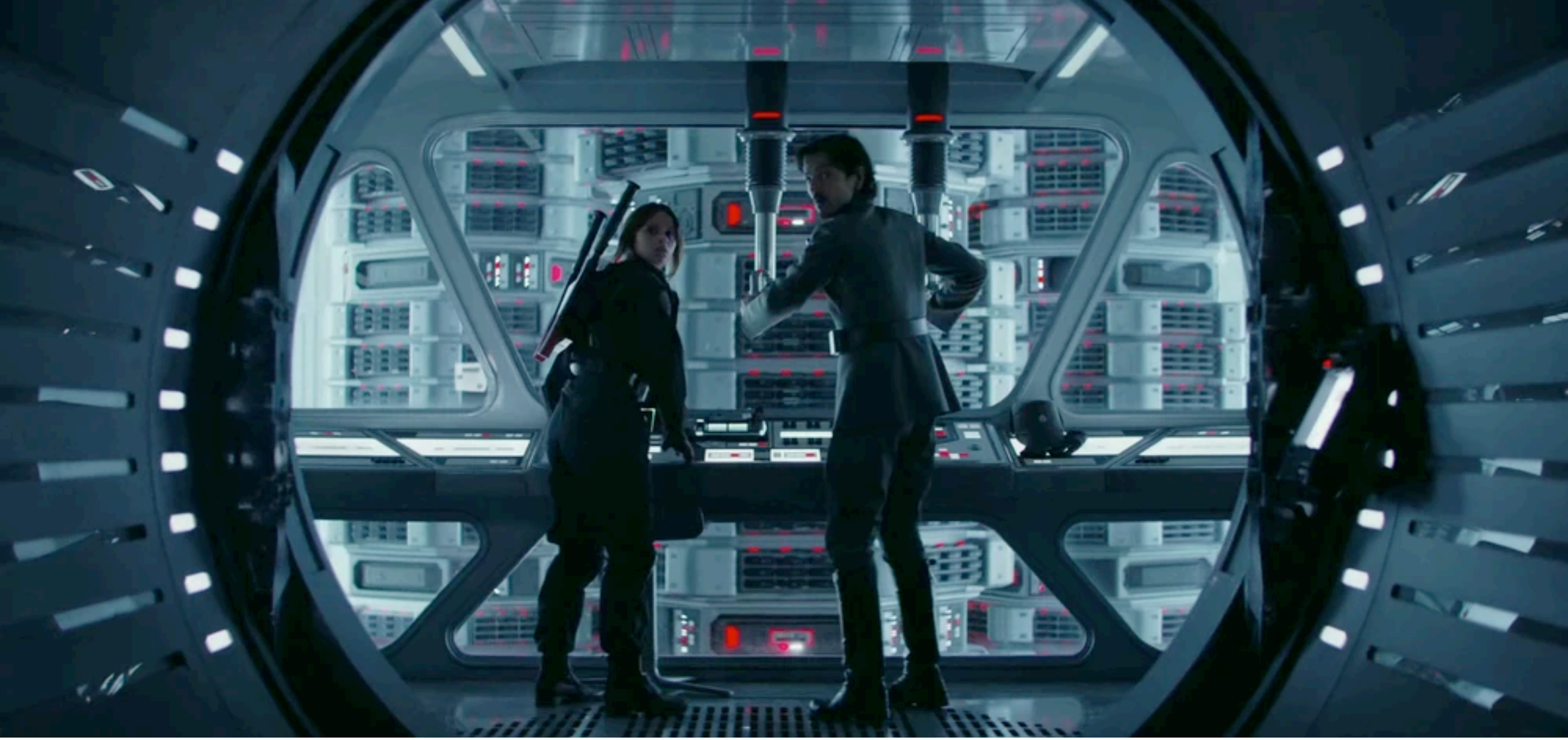
Minimize complexity!

The more complex it is, the more difficult & expensive it is to understand and change

Mistakes & bugs are more likely to be introduced when developing complex systems!
Also, security vulnerabilities when they are being modified or extended…

Follow the **kiss** principle: "**Keep it simple stupid**"

Be aware of common architectural **patterns**!



THE SCARIF VAULT: A CENTRALISED SECURITY ARCHITECTURE

From Star Wars: Rogue One
Example courtesy Ian Somerville

https://starwars.fandom.com/wiki/Data_vault

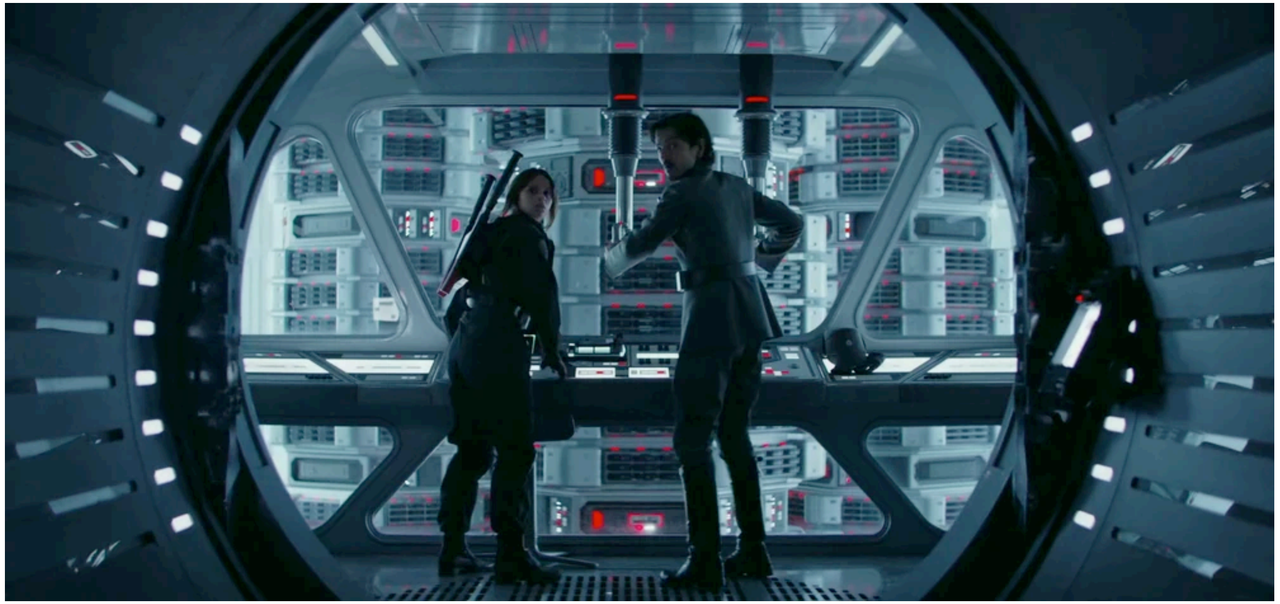
In the Star Wars prequel **Rogue One** (https://en.wikipedia.org/wiki/Rogue_One), the evil Empire have stored the plans for all of their equipment in a single, highly secure, well-guarded, remote location. This is called a centralised security architecture. It is based on the principle that if you maintain all of your information in one place, then you can apply lots of resources to protect that information and ensure that intruders can't get hold of it.

Unfortunately (for the Empire), the rebels managed to breach their security. They stole the plans for the Death Star, an event which underpins the whole Star Wars saga. In trying to stop them, the Empire destroyed their entire archive of system documentation with who knows what resultant costs. Had the Empire chosen a distributed security architecture, with different parts of the Death Star plans stored in different locations, then stealing the plans would have been more difficult. The rebels would have had to breach security in all locations to steal the complete Death Star blueprints.

THE SCARIF VAULT: A CENTRALISED SECURITY ARCHITECTURE

From Star Wars: Rogue One
Example courtesy Ian Somerville

Centralised security architectures



THE SCARIF VAULT

https://starwars.fandom.com/wiki/Data_vault

From Star Wars: Rogue One

Easier to design and build protection; protected information can be **accessed more efficiently**…

But, if your security is breached, you lose everything

Distributing information: takes longer to access all of the information and costs more to protect it…

But, if security is breached in one location, you only lose the information that you stored there

Other non-functional system attributes…

Responsiveness
Does the system return results to users in a reasonable time?

Reliability
Do the system features behave as expected by both developers and users?

Availability
Can the system deliver its services when requested by users?

Security
Does the system protect itself and users' data from unauthorized attacks?

Usability
Can system users access the features that they need and use them quickly and without errors?

Maintainability
Can the system be readily updated and new features added without undue costs?

Resilience
Can the system continue to deliver user services in the event of partial failure or external attack?

Architectural design questions

1. **How should the system be organized** as a set of architectural components, where each of these components provides a subset of the overall system functionality?
2. **How should these architectural components be distributed and communicate with each other?**
3. **What technologies should you use in building the system and what components should be reused?**

Component organisation

Abstraction: focus on essential elements of a system or software component without worrying about its details

Architectural level: large-scale architectural **components**… analyse them and decompose them into finer-grain components…

Layered models: used to illustrate how a system is composed of components

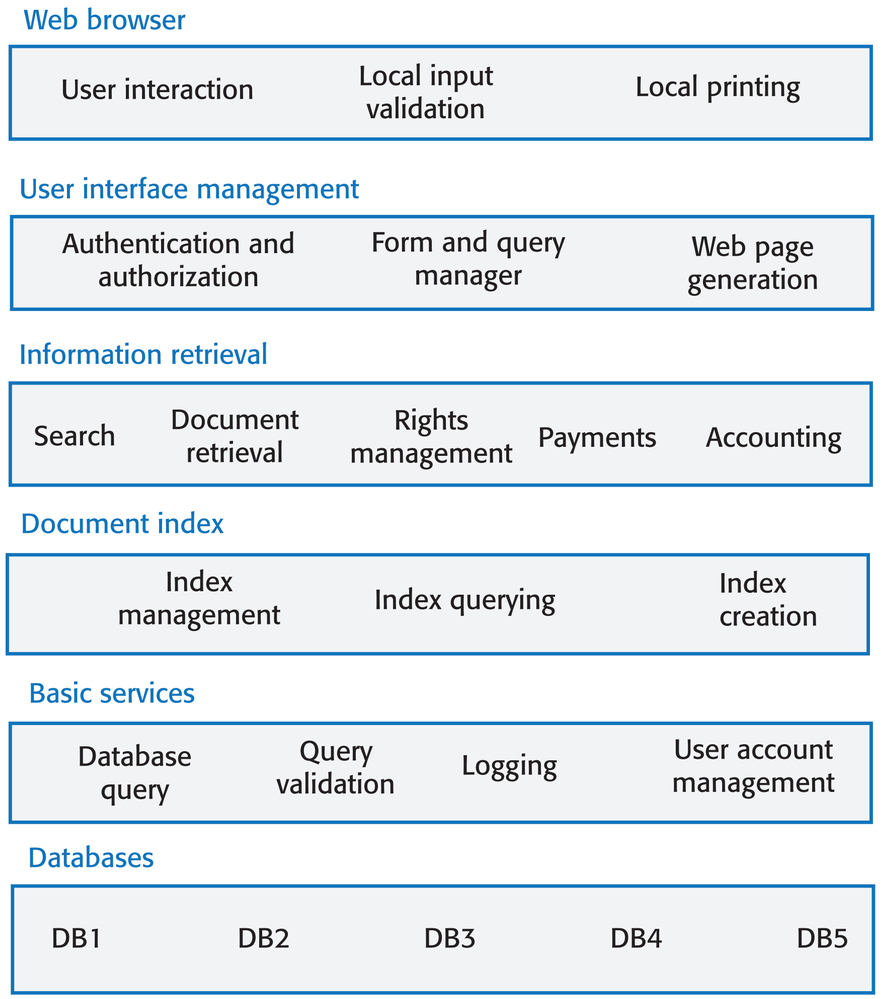


Figure 4.6: Architectural model of a document retrieval system

Component relationships & Separation of concerns

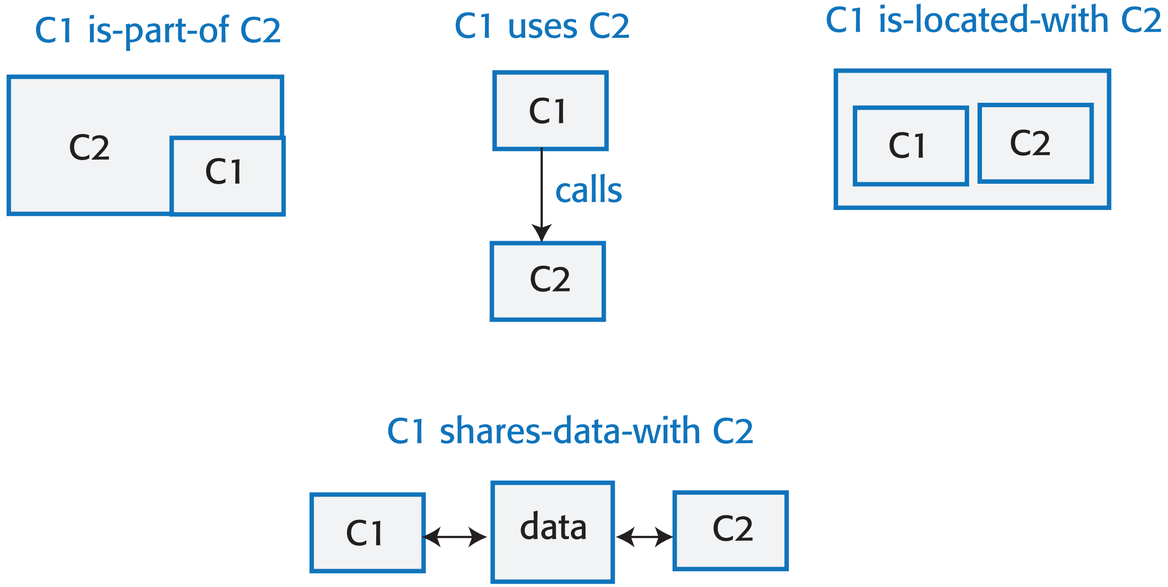


Figure 4.7: Component relationship examples

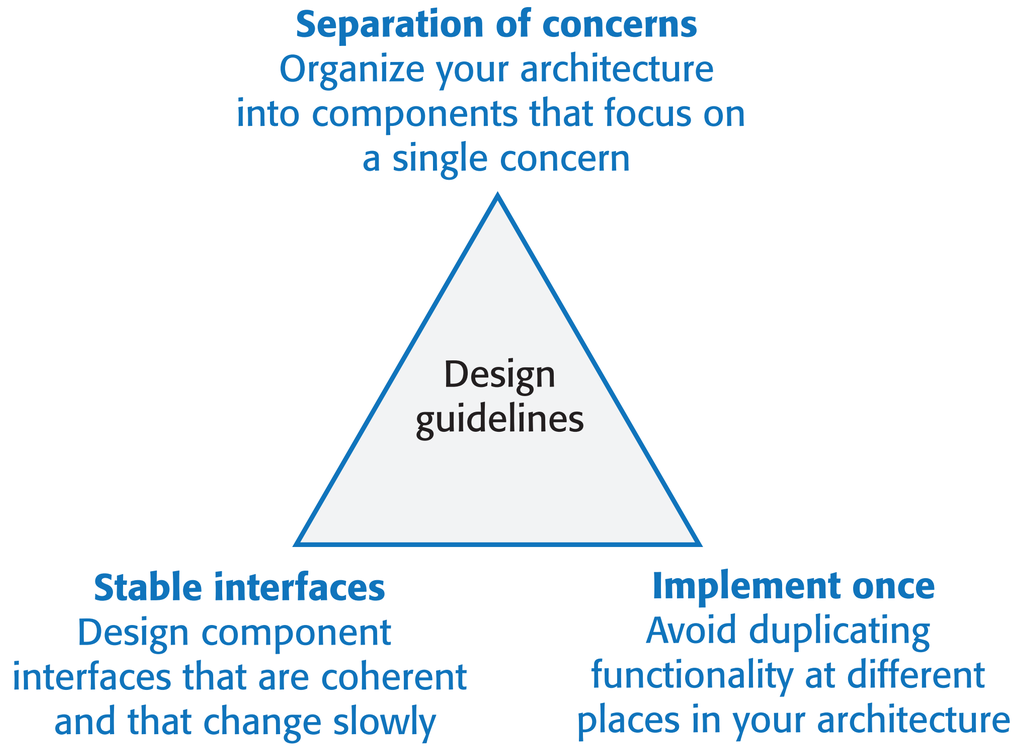


Figure 4.8: Architectural design guidelines

Layered architectures: guidelines

Each layer is considered **separately**!

- **Top layer**: user interaction, **top-1:** user interface management, **top-2**: with information retrieval…

Within each layer: components are **independent** and **do not overlap in functionality**

- **Lower layers**: include components that provide **general functionality**… no need to replicate this in the components in a higher level…

Architectural model is a **high-level model** that **does not include implementation information**…

- Components at level X should **only interact with the APIs** of the components in level $X\text{-}1$. *i.e.* interactions should be **between layers and not across layers**

Layered architectures: Cross-cutting concerns

Systemic: they affect the whole system!

Affect **all layers** in the system and how people use the system

Completely **different from the functional concerns** represented by layers in a software architecture

Every layer has to take them into account…

The existence of cross-cutting concerns is the reason **why modifying a system after it has been designed to improve its security is often difficult**.

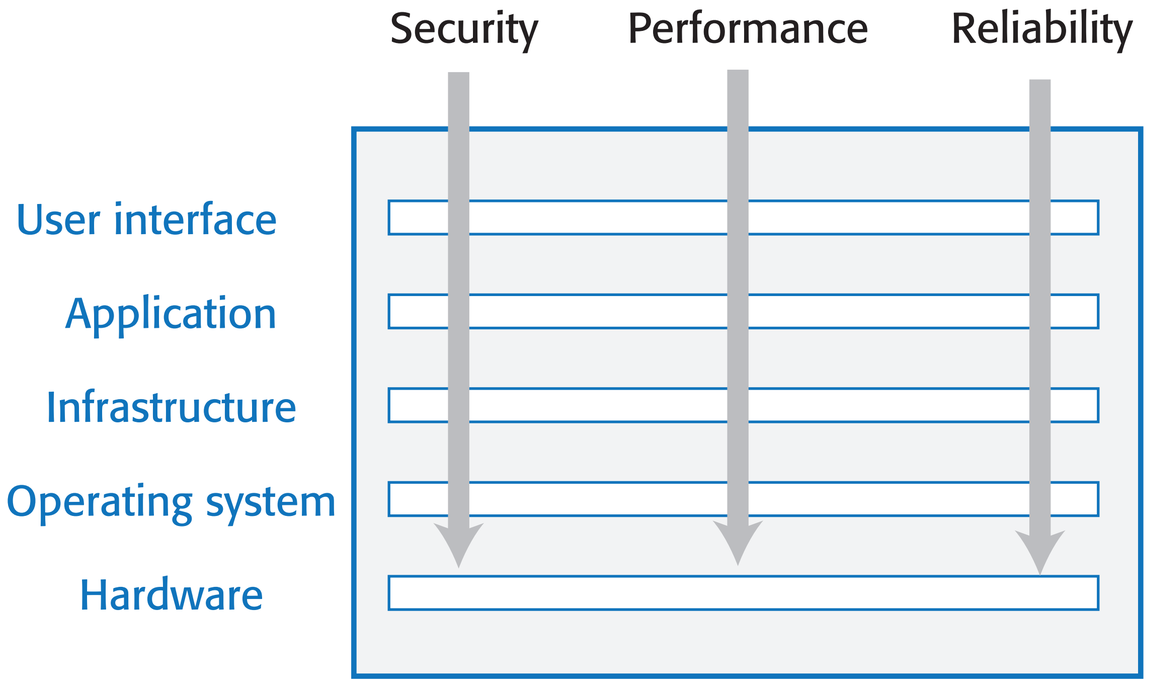


Figure 4.9: Cross-cutting concerns

Service-oriented architectures (SOA)

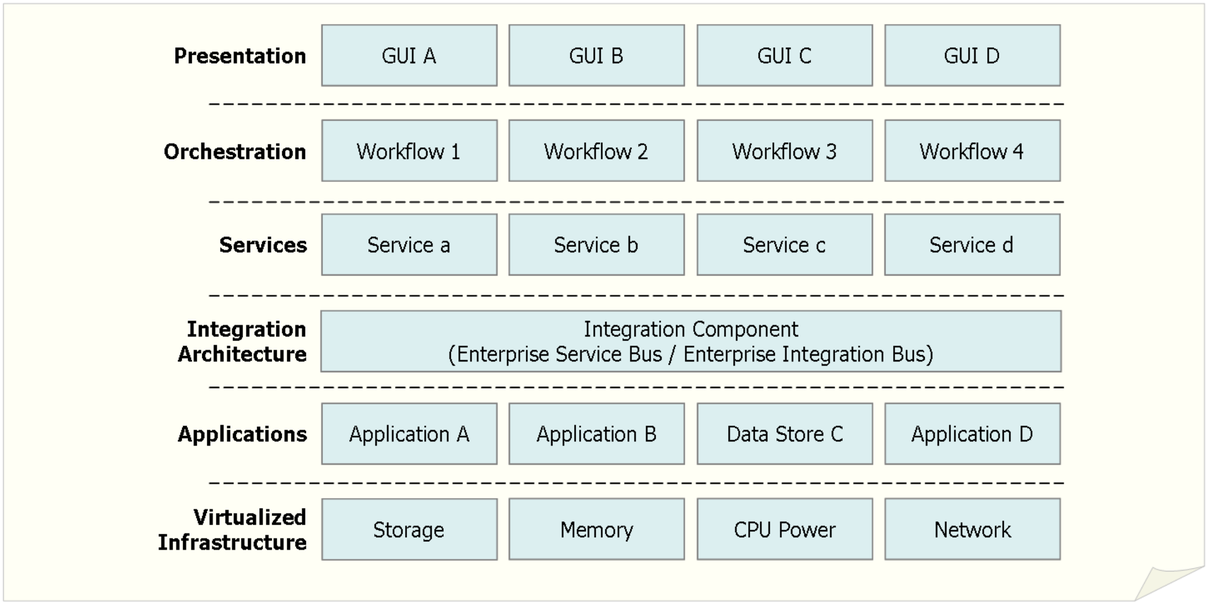
Every component in the system is a service! Any service is (possibly) replaceable and new services can be created by combining existing services

Full service integration: Services are **aware of** and can **communicate** with other services **through their APIs**.

Partial integration: Services **may share service components** and **databases** but are **not aware of** and **cannot communicate** directly with other application services

Independent: Services do not use any shared system services or databases and they are unaware of any other services in the system. They can be replaced by any other comparable service.

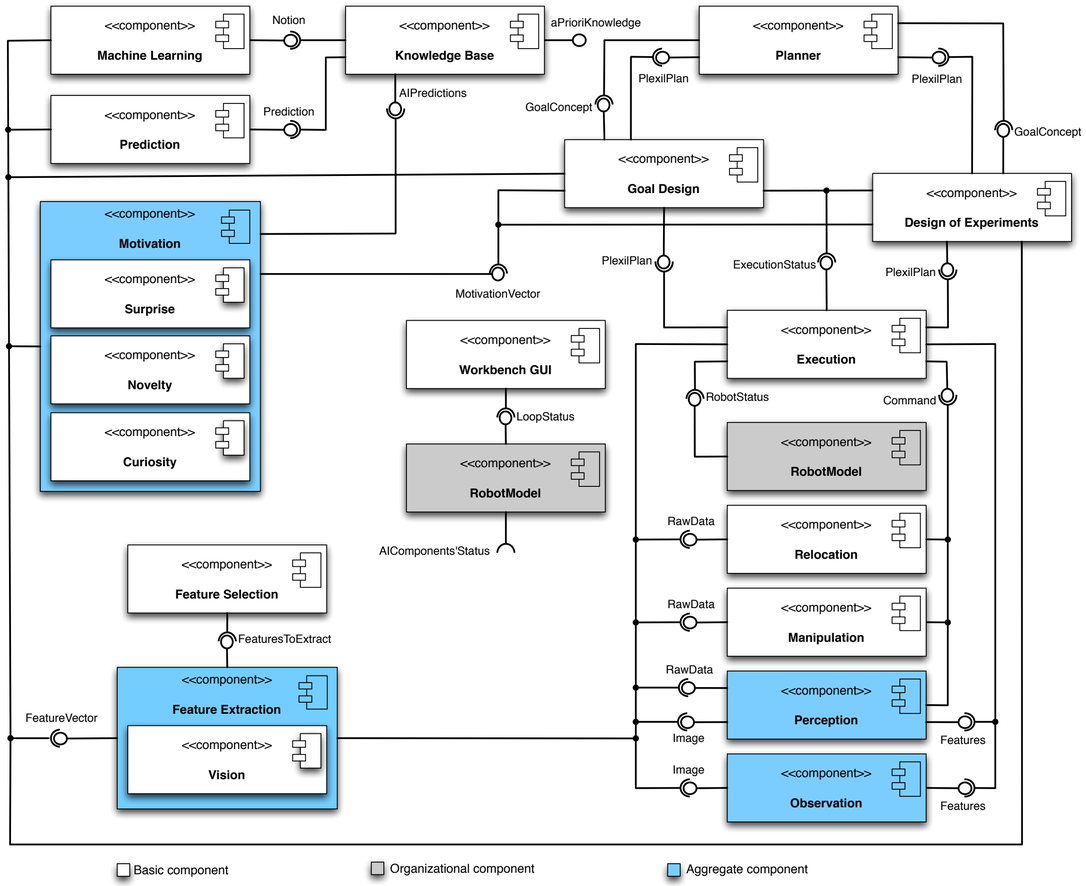
Loosely-coupled, autonomous components that offer services through their well-defined interfaces and form a service-oriented architecture.

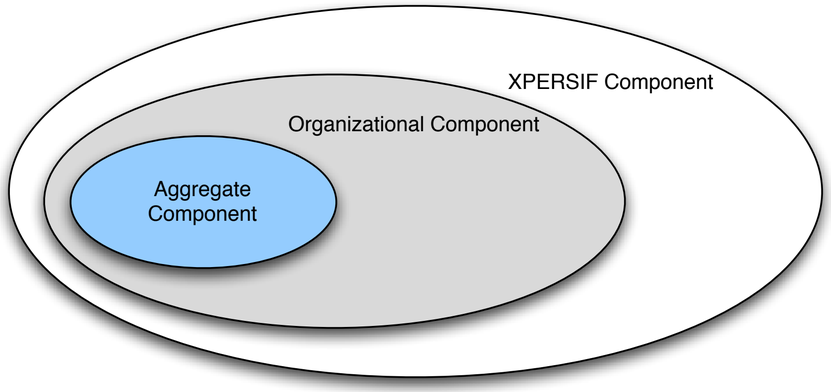


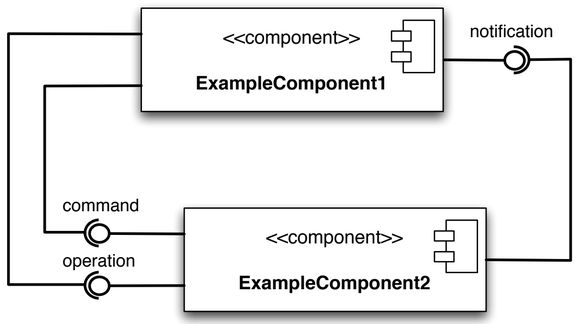
Reproduced from P. Welkenbach, M. Heinisch, D. Liebhart, M.I Knings, G.Schmutz, M. Kliiker, and P. Pakull. Architecture Blueprints. Carl Hanser, 2006.

Service-oriented architectures (SOA)

Every component in the system is a service! Any service is (possibly) replaceable and new services can be created by combining existing services







Service-oriented architectures (SOA)

Stateless components: they can be replicated and can migrate from one computer to another

Many servers may be involved in providing services

SOAs are usually easier to scale as demand increases and are resilient to failure

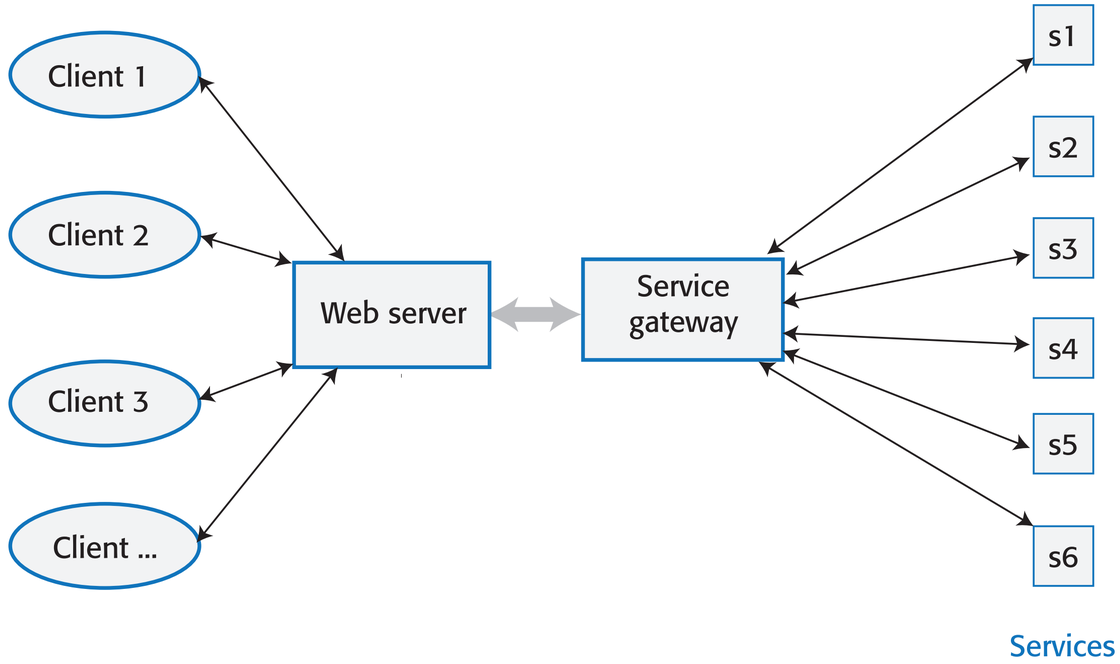


Figure 4.15: Service-oriented architecture

Distribution architecture: Client/server arch.

…defines the **servers** in the system and the **allocation of components to these servers**

Client-server architectures:a type of distribution architecture suited to applications where clients access a shared DB and business logic operations on that data

User interface implemented on the user's own computer or mobile device.

Functionality is **distributed** between the **client** and one or more **server** computers

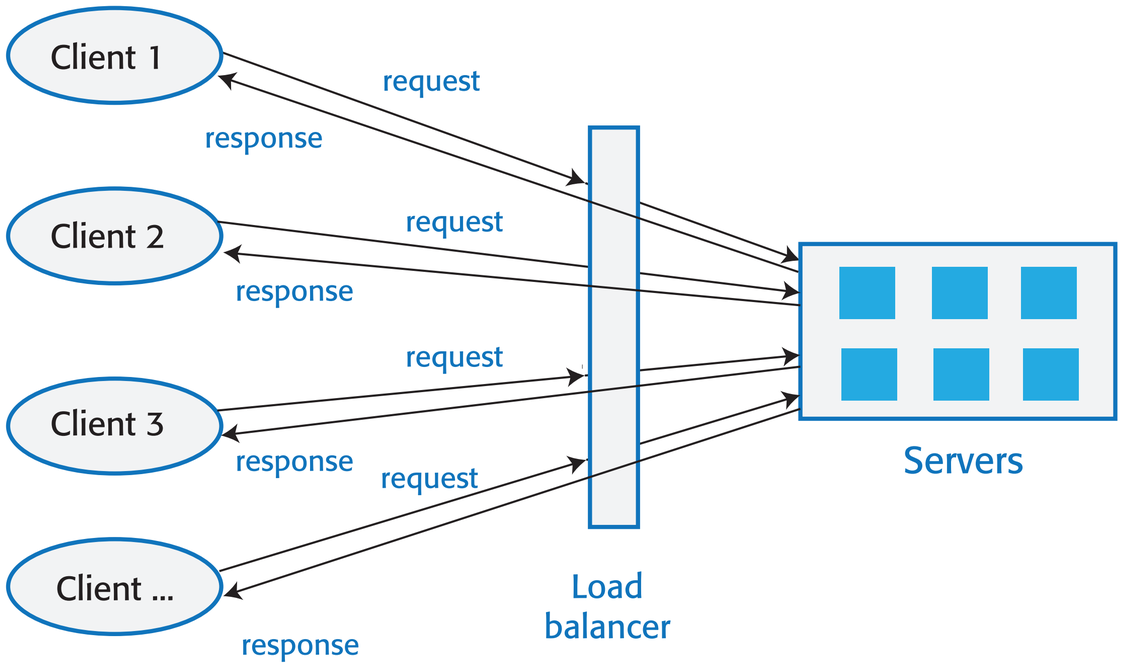


Figure 4.12: Client-server architecture

Client/server architecture: Model-View-Controller pattern

…is a software architectural **pattern** that **separates** **an application into three interconnected components**:

1. **Model** (data and business logic)
2. **View** (user interface)
3. **Controller** (handles user input and interacts with the Model and View)

Client/server architecture: Model-View-Controller pattern

The **Model component** **manages** the system data and associated operations on that data

e.g. Customer info is required from the DB, the Controller will **request** the Model to **retrieve** the **data**. The data is then sent to the View component (either directly or through the controller)

The **View component** **defines** and **manages** **how** the data is presented to the user (GUI).

e.g. The view will include all the UI components like text boxes, drop downs,…

The **Controller component** **manages** **user interaction** (*e.g.* key presses, mouse clicks,…) and **passes** these interactions to the View and the Model

e.g. Saving a specific document → updates the model

e.g. scrolling a particular document→ change the view

Client/server architecture: Model-View-Controller pattern

Communication using the **HTTP protocol**. Client sends a message to the server that includes an instruction such as **GET** or **POST** along with the identifier of a resource (usually a **URL**) on which that instruction should operate… message may also include more information, *e.g.* from a form on a page

HTTP: text-only protocol, so structured data has to be represented as text. Two (widely-used) ways to represent it:

1. **XML**: markup language with tags used to identify each data item
2. **JSON**: simpler representation based on the representation of objects in the Javascript language

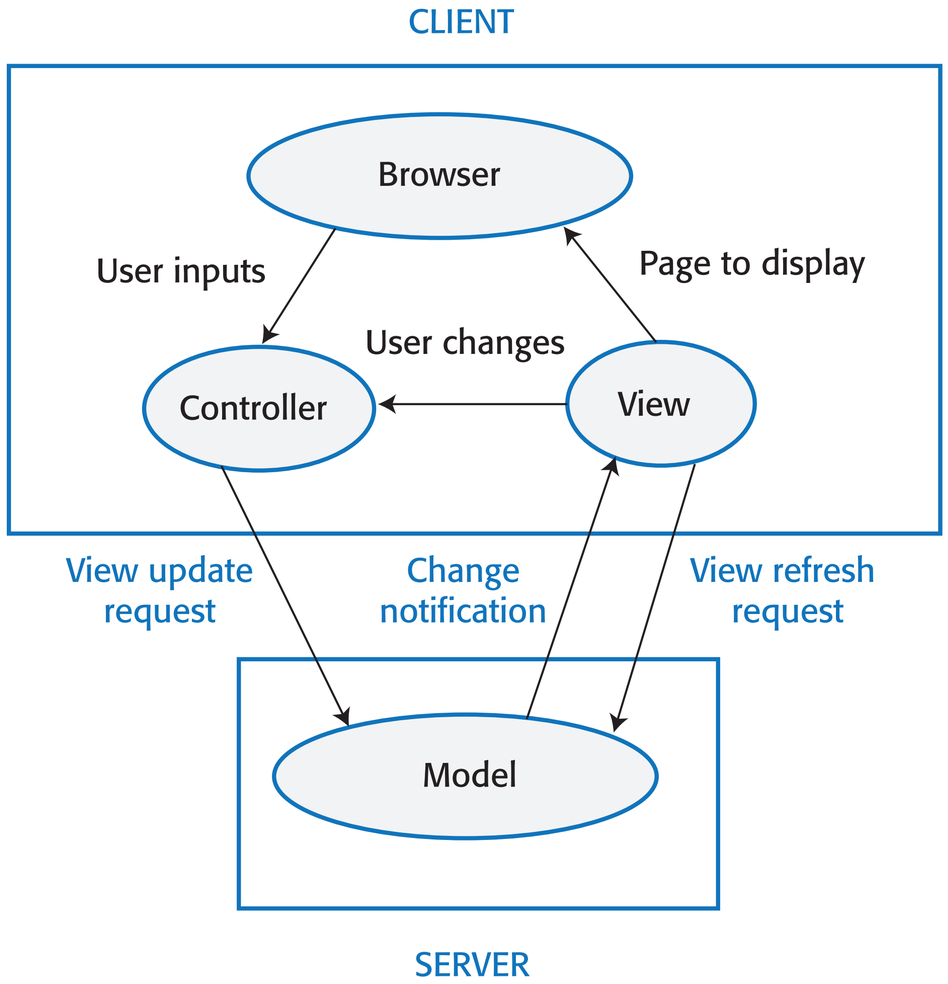
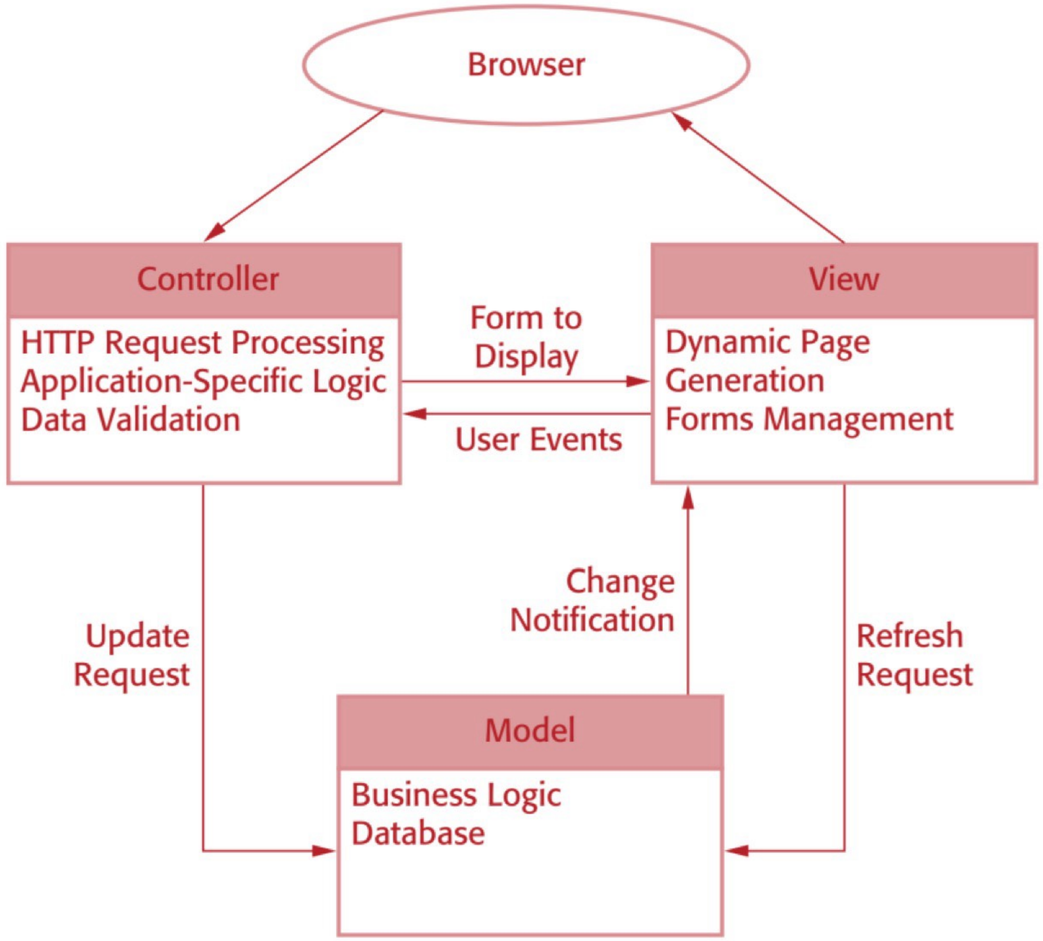


Figure 4.13: Model-View-Controller pattern

Client/server architecture: Model-View-Controller pattern



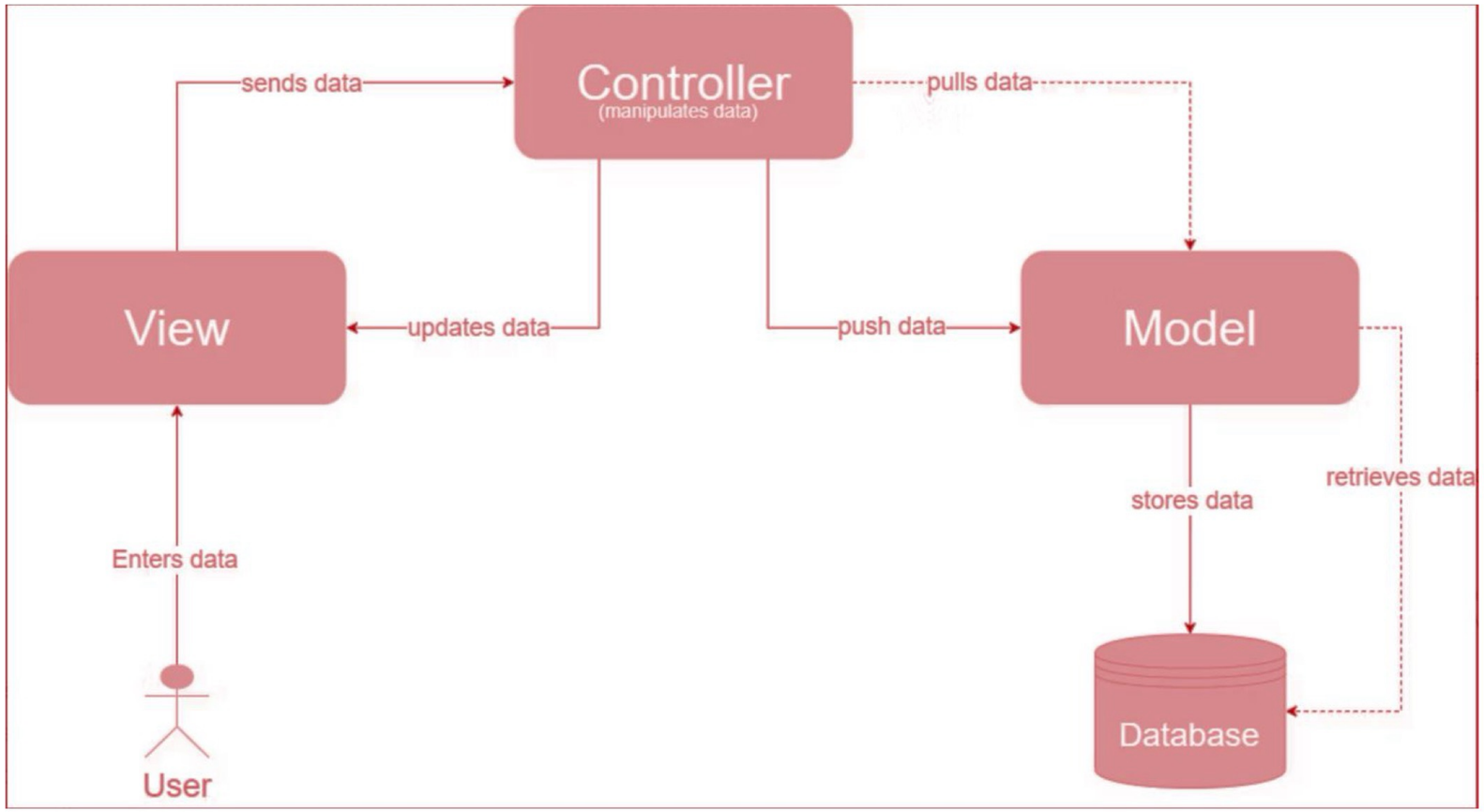
DI. JOHN ZARI

Model
Manages the data, logic, and rules of the application

View
Responsible for the presentation layer and what the user sees

Controller
Acts as the intermediary between the Model and the View

Client/server architecture: Model-View-Controller pattern



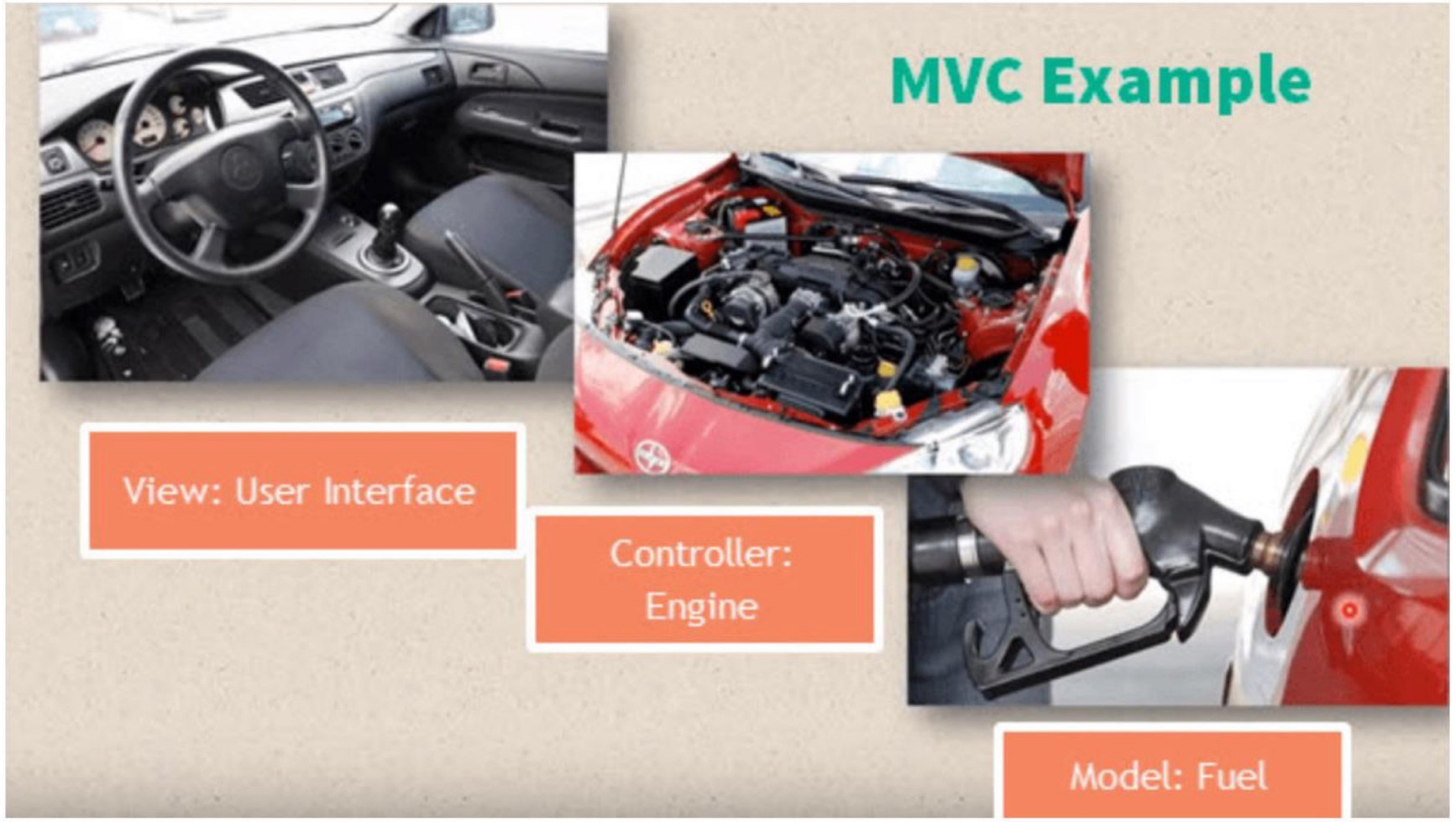
Model
Manages the data, logic, and rules of the application

View
Responsible for the presentation layer and what the user sees

Controller
Acts as the intermediary between the Model and the View

Dr. JOHN ZAKI

Client/server architecture: Model-View-Controller pattern

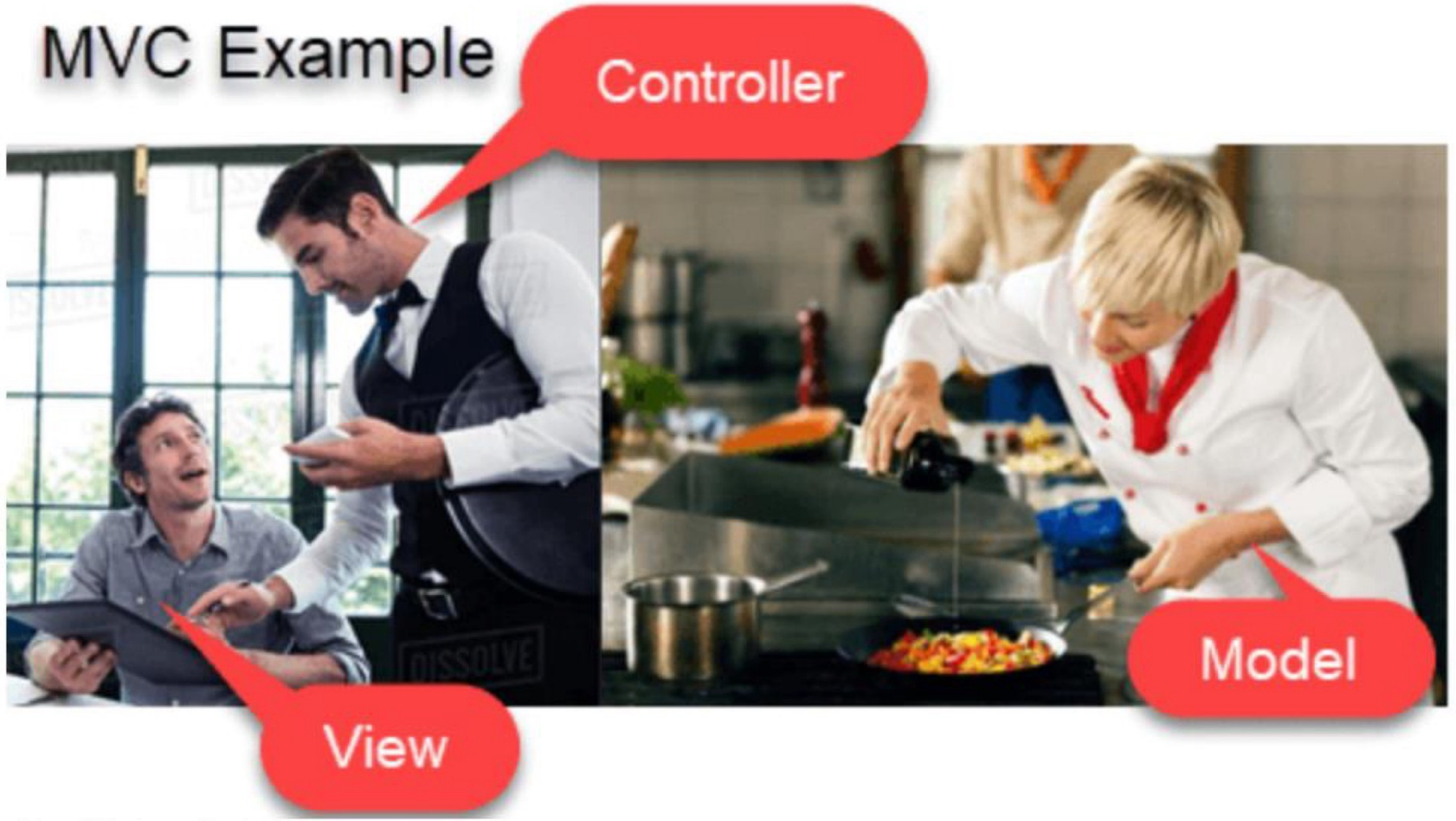


Model
Manages the data, logic, and rules of the application

View
Responsible for the presentation layer and what the user sees

Controller
Acts as the intermediary between the Model and the View

Client/server architecture: Model-View-Controller pattern



Picture credit: https://www.guru99.com/mvc-tutorial.html

Model
Manages the data, logic, and rules of the application

View
Responsible for the presentation layer and what the user sees

Controller
Acts as the intermediary between the Model and the View

Client/server architecture: Model-View-Controller pattern

Advantages:

Improved scalability

Maintainability (components are reasonably independent)

Reusability (Models can be reused by multiple views)

Adoption of MVC makes an application easier to understand

Easier to extend & test (each component can be tested separately)

Disadvantage:

Can involve additional code and code complexity even when the data model and interactions are simple

Technology questions

1. **Database**: Should we use a relational SQL database or an unstructured NOSQL database?
2. **Platform**: Should we deliver our product on a mobile app and/or a web platform?
3. **Server**: Should we use dedicated in-house servers or design our system to run on a public cloud? If a public cloud, should we use Amazon, Google, Microsoft, or some other option?
4. **Open source**: Are there suitable open-source components that we could incorporate into your products?
5. **Development tools**: Do our development tools embed architectural assumptions about the software being developed that limit our architectural choices?

Technology Qs: Web-based, mobile app-based or both

Mobile app issues:

Intermittent connectivity: you must be able to provide a limited service without network connectivity

Processor power: Mobiles have less powerful processors…need to minimize computationally-intensive operations

Power management: Battery life is limited so you should try to minimize the power used by your application

On-screen keyboard: Slow and error-prone… minimize input using the keyboard to reduce user frustration

To deal with these differences, you usually **need** separate browser-based and mobile **versions** of your **product front-end**!

May need a completely **different decomposition architecture** in these different versions to ensure that performance and other characteristics are maintained…

Technology Qs: Servers

Key decision: whether to design the system to run on customer **servers** or to run on the **cloud**

For consumer products that are not mobile apps: almost always makes sense to **develop for the cloud**.

For business products: a more difficult decision…

Some businesses worry about **cloud security** and prefer to run their systems on in-house servers.

May have a **predictable pattern of system usage** so less need to design the system to cope with large changes.

If cloud, **which cloud provider** to use?

Technology Qs: Open-source

…is software that is **available freely**, which **you can modify** as you like

Advantage: reuse rather than implement new software (reduces development costs and time to market)

Disadvantage: constrained by that software and have no control over its evolution…

Decision also depends on **availability**, **maturity** and **continuing support** of the components.

Open source **license issues** may impose constraints on how you use the software

In the end, the decision depends on the **type of product** that you are developing, **your target market** and the **development team's expertise**

Technology Qs: Development tools

Development technologies, *e.g.* a **mobile development toolkit** or a **web application framework**, **influence the architecture** of your software.

They have built-in assumptions about system architectures and you have to conform to these assumptions to use the development system.

The development technology that you use may also have an indirect influence on the system architecture.

Developers usually favour architectural choices that use familiar technologies that they understand. For example, if your team have a lot of experience of relational databases, they may argue for this instead of a NoSQL database.

Modern software architectures

Microservices: collection of small, independent **services**, each with its own business function. (communication via APIs)

Event-Driven Architecture: Components **react** to **events**, allowing for **asynchronous processing** and **loose coupling** between different parts of the system. This pattern is agile and highly performant, ideal for real-time systems

Serverless: **Cloud-native pattern** where the cloud provider manages the underlying infrastructure, allowing developers to focus on writing code without managing servers

Microkernel: An architecture with a **core system** and **plug-in modules**. The microkernel handles the basic functionality and acts as the entry point, allowing interchangeable parts to be added as needed

See https://tecnovy.com/en/top-10-software-architecture-patterns for an interesting read…

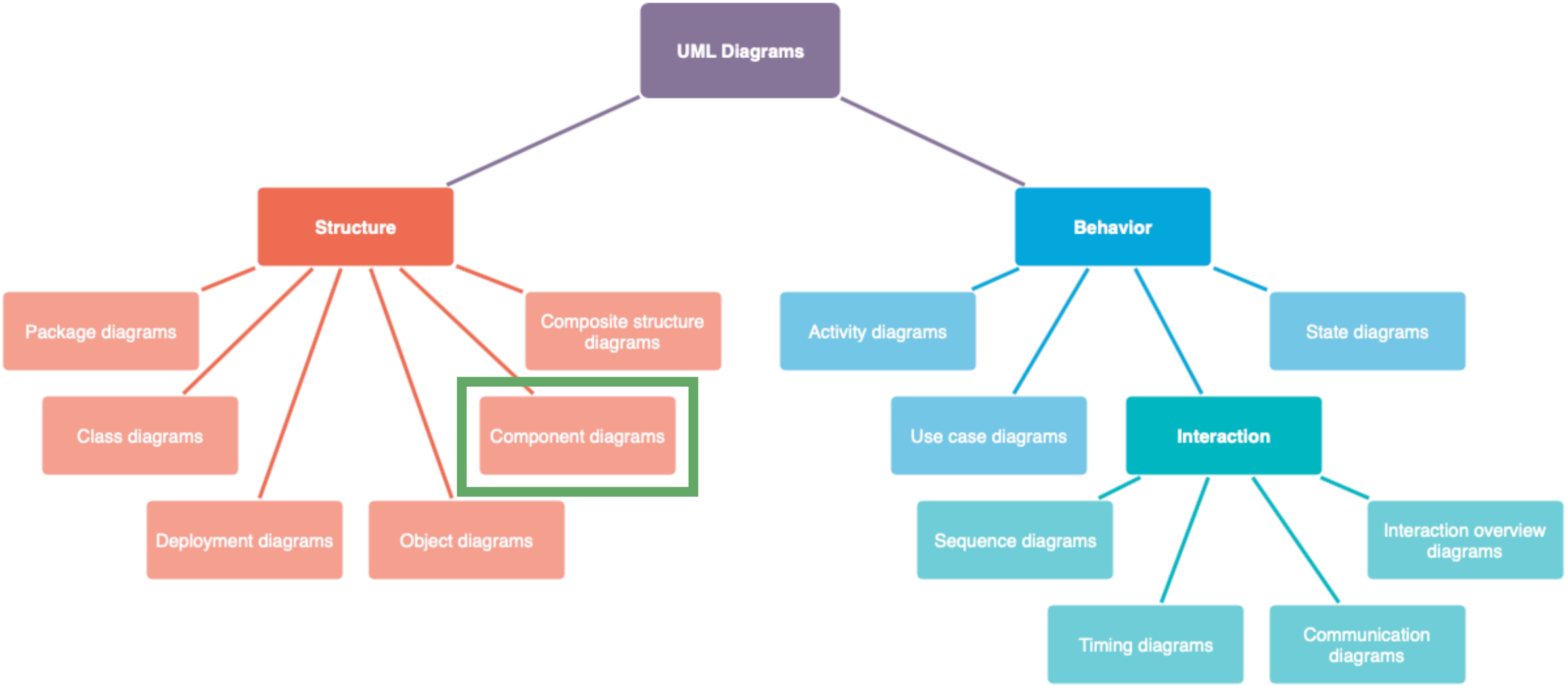
How do you document your software architecture?

Unified Modeling Language (UML)!!

Use case	Sequence	Class	State	Activity
Interactions between a **system** and its **environment**	Interactions between **actors** and **system** and between **system components**	Object classes in the system and **associations between those classes**	How the system reacts to **internal** and **external events**	**Activities** involved in a **process** or in **data processing**

https://drawio-app.com/blog/uml-diagrams/

Different models for different uses…



https://drawio-app.com/blog/uml-diagrams/


TEST THE CODE