Inf2-SEPP
Lecture 7:
Introduction to Design. Architectural Design

Adriana Sejfia

School of Informatics
University of Edinburgh

Previous lectures

- Requirements engineering:
  - In general, with its concepts and sub-activities
  - Using use cases and use case diagrams
  - In different types of systems and software development processes
  - Use of personas, scenarios and user stories in product engineering

This lecture

- Design
  - Concept
  - Outputs of the design process
  - Criteria for good design
  - Levels of design
    - Architectural design
      - Concept and importance of an architecture
      - Considerations for architectural design: system decomposition, distribution, technologies
      - Some important architectures

What is design?

Design is the process of deciding how software will meet requirements.

What is design?

Design is the process of deciding how software will meet requirements.

Usually excludes detailed coding level.

Outputs of design process

Outputs include:

- **models.**
  - E.g. using UML or Simulink
  - Often graphical
  - Can be executable

Outputs of design process

Outputs include:

- **models.**
  - E.g. using UML or Simulink
  - Often graphical
  - Can be executable
- **written documents**
  - Important that these record reasons for decisions

(Some) criteria for a good design

- It can meet the known requirements
  (functional and non-functional)

(Some) criteria for a good design

- It can meet the known requirements
  (functional and non-functional)
- It is maintainable:
  i.e. it can be adapted to meet future requirements

(Some) criteria for a good design

- It can meet the known requirements (functional and non-functional)
- It is maintainable: i.e. it can be adapted to meet future requirements
- It is straightforward to explain to implementors

(Some) criteria for a good design

- It can meet the known requirements
  (functional and non-functional)
- It is maintainable:
  i.e. it can be adapted to meet future requirements
- It is straightforward to explain to implementors
- It makes appropriate use of existing technology,
  e.g. reusable components

(Some) criteria for a good design

- It can meet the known requirements
  (functional and non-functional)
- It is maintainable:
  i.e. it can be adapted to meet future requirements
- It is straightforward to explain to implementors
- It makes appropriate use of existing technology,
  e.g. reusable components

Notice the human angle and the situation-dependency, e.g.

- Who will implement the design? OO programmers or functional programmers?
- What kind of future changes do we expect?

Levels of design

Design occurs at different levels, e.g. someone must decide:

- how is your system split up into subsystems?
  (high-level, or architectural, design)
- what are the classes in each subsystem?
  (low-level, or detailed, design)

What is an architecture?

An **architecture** is the **fundamental organisation 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** (IEEE)

What is an architecture?

An **architecture** is the **fundamental organisation 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** (IEEE)

- Pervasive, hence hard to change.
- An alternative definition: "**what stays the same**"
  - as the system develops
  - between related systems.

Other important definitions: component, service, module

A **service** is a "**coherent unit of functionality**" (Sommerville ESP)

Other important definitions: component, service, module

A **service** is a "**coherent unit of functionality**" (Sommerville ESP)

A **component** is "a named **software unit** that **offers one or more services** to other software components or to end-users of the software". It "can be anything from a program (large scale) to an object (small scale)". (Sommerville ESP)

Other important definitions: component, service, module

A **service** is a "**coherent unit of functionality**" (Sommerville ESP)

A **component** is "a named **software unit** that **offers one or more services** to other software components or to end-users of the software". It "can be anything from a program (large scale) to an object (small scale)". (Sommerville ESP)

A **module** is a "named **set of components**" which "should have **something in common**. For example, they may provide a set of related services" (Sommerville ESP)

Why is architecture important?

- Because it has a fundamental influence on non-functional (very important!) characteristics of the system:
  - Non-functional attributes may not all be optimizable
  - E.g. two components sharing or not a database has different cost vs. maintainability and resilience effects

Why is architecture important?

- Because it has a fundamental influence on non-functional (very important!) characteristics of the system:
  - Non-functional attributes may not all be optimizable
  - E.g. two components sharing or not a database has different cost vs. maintainability and resilience effects

- Because it affects the complexity of the software:
  - The more complex, the less maintainable, more error prone, less secure.
  - Minimising complexity important goal for architectural design

Architectural design

Involves creating a description of the architecture showing components and their relationships.

Architectural design

Involves creating a description of the architecture showing components and their relationships.

Important architectural design issues to consider:

- Non-functional requirements

Architectural design

Involves creating a description of the architecture showing components and their relationships.

Important architectural design issues to consider:

- Non-functional requirements
- Product lifetime: if long-lived, architecture should be able to evolve

Architectural design

Involves creating a description of the architecture showing components and their relationships.

Important architectural design issues to consider:

- Non-functional requirements
- Product lifetime: if long-lived, architecture should be able to evolve
- Software reuse: saves time, constrains architectural choices

Architectural design

Involves creating a description of the architecture showing components and their relationships.

Important architectural design issues to consider:

- Non-functional requirements
- Product lifetime: if long-lived, architecture should be able to evolve
- Software reuse: saves time, constrains architectural choices
- Number of users: if very variable, architecture should allow quickly scaling up and down

Architectural design

Involves creating a description of the architecture showing components and their relationships.

Important architectural design issues to consider:

- Non-functional requirements
- Product lifetime: if long-lived, architecture should be able to evolve
- Software reuse: saves time, constrains architectural choices
- Number of users: if very variable, architecture should allow quickly scaling up and down
- Software compatibility: constrains architectural choices

Architectural design

Involves creating a description of the architecture showing components and their relationships.

Important architectural design issues to consider:

- Non-functional requirements
- Product lifetime: if long-lived, architecture should be able to evolve
- Software reuse: saves time, constrains architectural choices
- Number of users: if very variable, architecture should allow quickly scaling up and down
- Software compatibility: constrains architectural choices
- Planned schedule, team capabilities, budget etc.

Architectural design: trade-offs

- **Maintainability vs performance**: having fine-grained components with individual responsibilities and own data structures is good for maintainability, but affects performance due to communication and data transfer overheads

Architectural design: trade-offs

- **Maintainability vs performance**: having fine-grained components with individual responsibilities and own data structures is good for maintainability, but affects performance due to communication and data transfer overheads

- **Security vs usability**: layers of components can help with security, but affect usability as multiple authentication layers frustrate users.

Architectural design: trade-offs

- **Maintainability vs performance**: having fine-grained components with individual responsibilities and own data structures is good for maintainability, but affects performance due to communication and data transfer overheads

- **Security vs usability**: layers of components can help with security, but affect usability as multiple authentication layers frustrate users.

- **Availability vs time to market and cost**: redundant components help with availability, but at increased cost, complexity, error proneness.

Architectural design: main questions

1. How should the system be decomposed into a set of components?

Architectural design: main questions

1. How should the system be decomposed into a set of components?

2. (web-based systems) How should the components be distributed and how should they communicate?

Architectural design: main questions

1. How should the system be decomposed into a set of components?

2. (web-based systems) How should the components be distributed and how should they communicate?

3. What technologies should be used in developing the system?

1. Decomposing the system into architectural components

Identifying large-scale components, then analysing and splitting them up into smaller components.

1. Decomposing the system into architectural components

Identifying large-scale components, then analysing and splitting them up into smaller components.

Concerns:

- Some non-functional requirements (e.g. security, performance, reliability) may be cross-cutting
- Complexity (major concern) due to the number of components and their relationships (exponential).

1. Decomposing the system into architectural components

Design guidelines for controlling complexity:

- Separation of concerns: components doing only one thing; grouping components with related functionality.
- Implement once: not duplicating functionality
- Stable interfaces: hiding a component's implementation details behind a component interface (API) so that dependant components do not need to change when this component changes

1. Decomposing the system into architectural components

Design guidelines for controlling complexity:

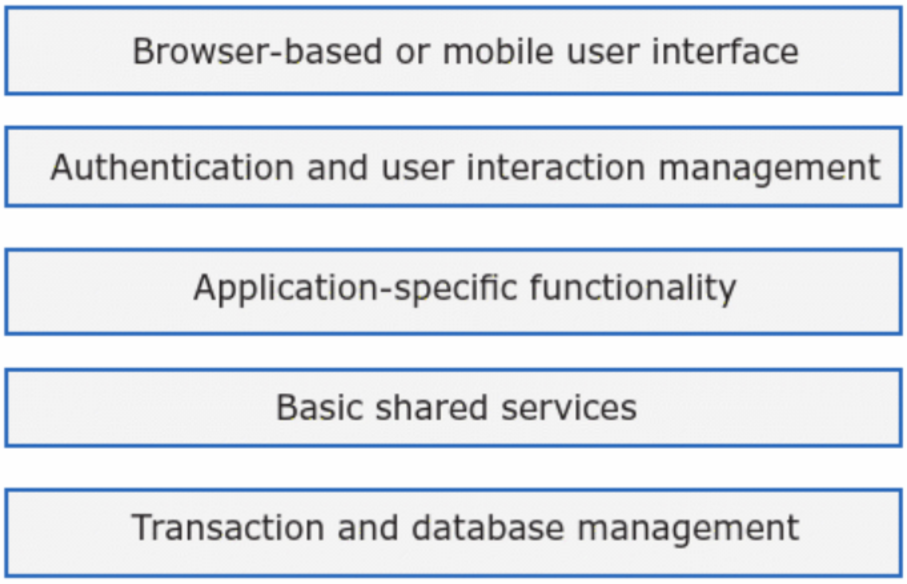
- **Separation of concerns:** components doing only one thing; grouping components with related functionality.
- **Implement once:** not duplicating functionality

1. Decomposing the system into architectural components

Design guidelines for controlling complexity:

- **Separation of concerns:** components doing only one thing; grouping components with related functionality.
- **Implement once:** not duplicating functionality
- **Stable interfaces:** hiding a component's implementation details behind a component interface (API) so that dependent components do not need to change when this component changes

Example: A generic layered architecture for a web-based application



Taken from: Sommerville, I., 2020. Engineering Software Products. Pearson.

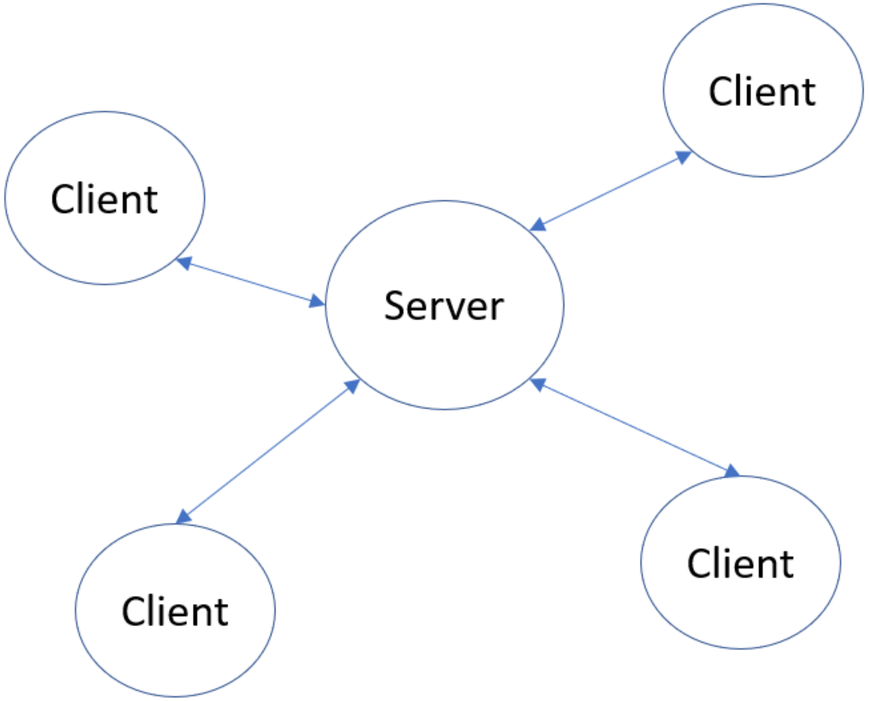
2. The distribution architecture (for web-based systems)

Defines how components are distributed online.

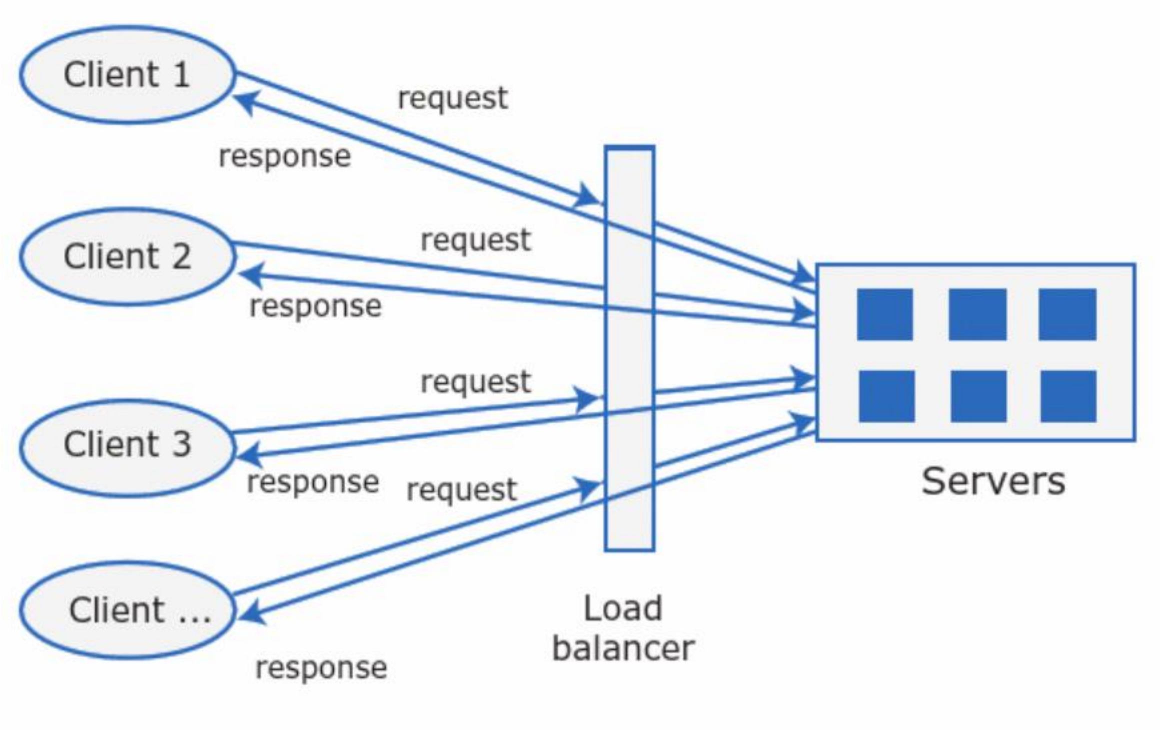
Some well-known architectures:

- Client-server architecture, with some variations:
  - Multi-tier client server architecture
  - Service-oriented architecture
- Peer to peer architecture
- Message bus architecture

Client-server architecture: high-level view with one server



Client-server architecture: logical view for web-based and mobile software systems



- Taken from: Sommerville, I., 2020. Engineering Software Products. Pearson.

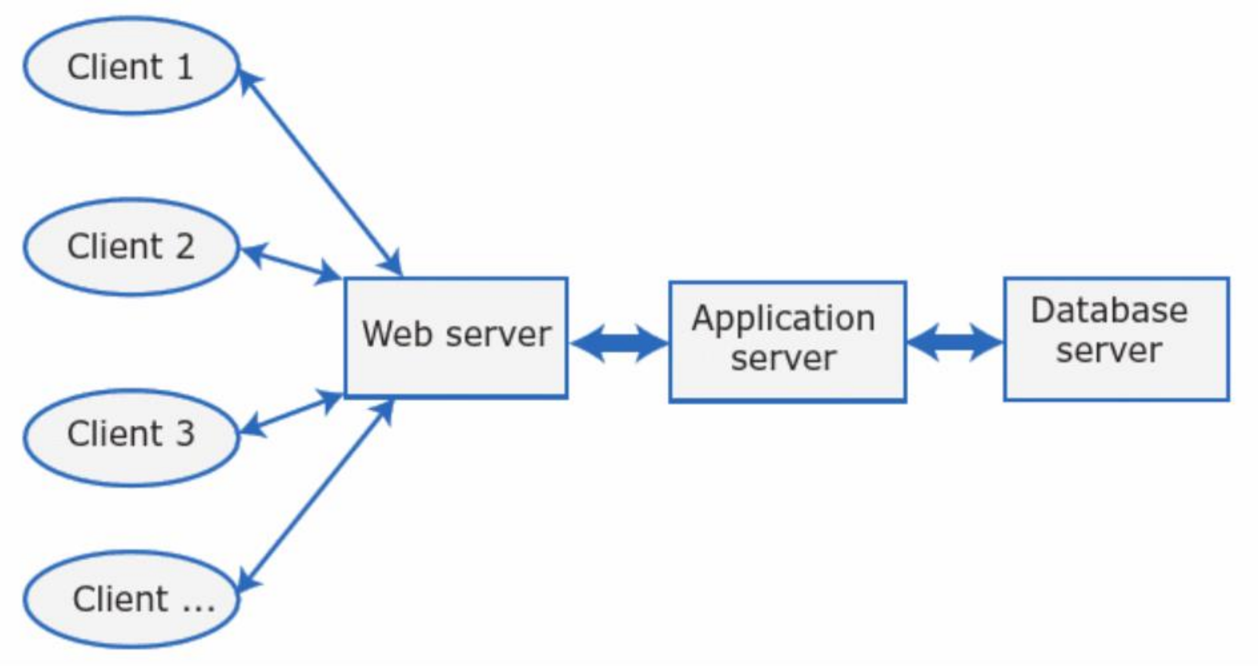
Client-server architecture: logical view for web-based and mobile software systems

- Clients send requests to servers, which process these requests and provide a response
- Client responsible for user interaction, based on the data moving between it and the server
- Servers initially conducted all processing, now clients are computers or mobile devices with large processing power so significant processing on clients
- Several servers e.g. web and database
- Load balancer distributes requests to servers, ensures even load
- Organised frequently using the Model-View-Controller (MVC) pattern.

Client-server architecture variation 1: The multi-tier client-server architecture

- Use of an object-oriented approach (from the 1990s)
- Single "monolithic" system with a shared database
- Several communicating servers with different individual responsibilities and running large software components
- Good if using structured data with concurrent updates, and for business systems running on local servers.

Client-server architecture variation 1: The multi-tier client-server architecture

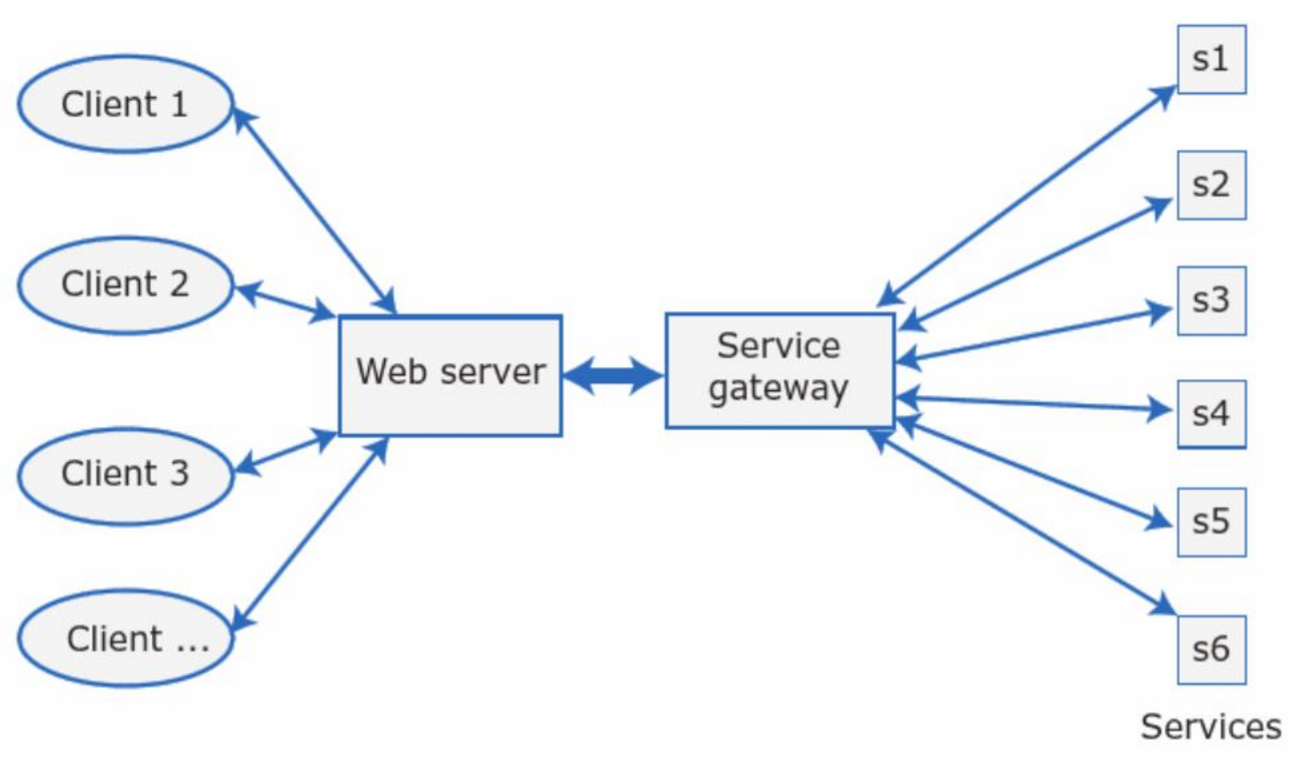


Taken from: Sommerville, I., 2020. Engineering Software Products. Pearson.

Client-server architecture variation 2: The service-oriented architecture

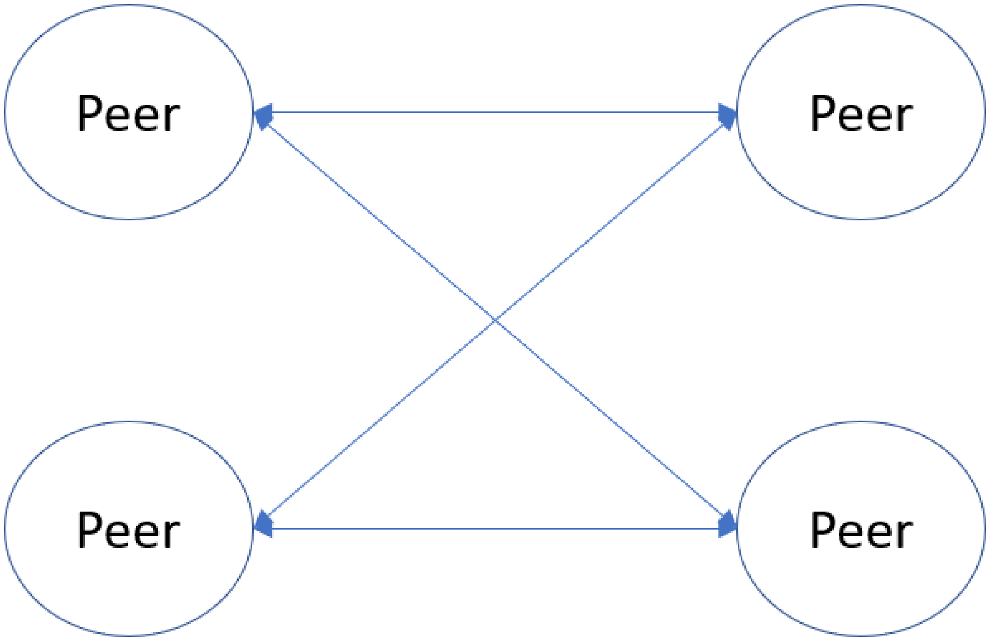
- More modern, becoming the norm
- Fine-grained services that may be provided by many servers
- Services are stateless, so independent and can be replicated, distributed, migrated between servers
- Good if system components need to be updated often, or there is a need for scalability (e.g. use on the cloud) and resilience to failure

Client-server architecture variation 2: The service-oriented architecture

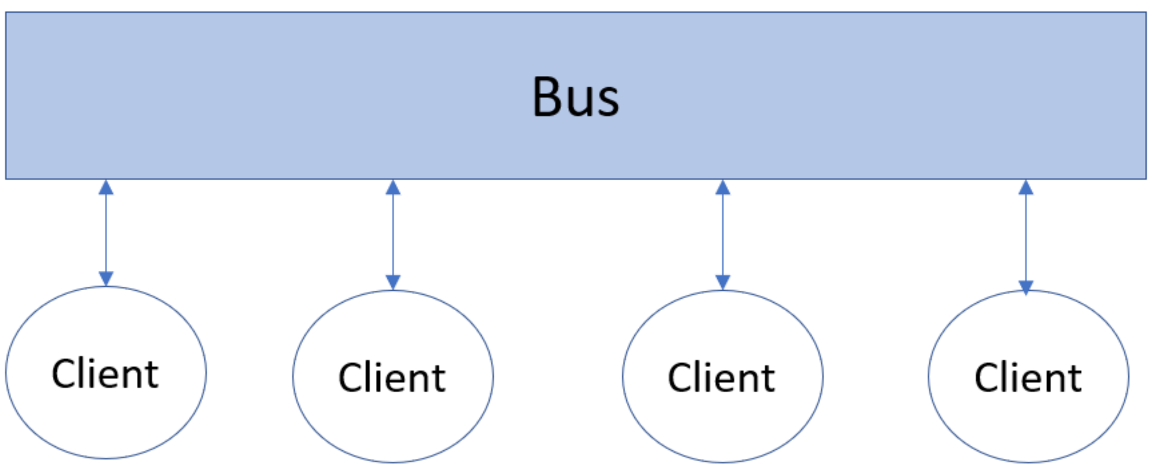


Taken from: Sommerville, I., 2020. Engineering Software Products. Pearson.

Peer to peer architecture



Message bus architecture



3. Technological considerations

Technologies need to be decided since architectural design, as changing them later is difficult and expensive

3. Technological considerations

Technologies need to be decided since architectural design, as changing them later is difficult and expensive

Technologies to consider:

- Database: relational or NoSQL?

3. Technological considerations

Technologies need to be decided since architectural design, as changing them later is difficult and expensive

Technologies to consider:

- Database: relational or NoSQL?
- Delivery platform: browser-based or mobile?
- Server: using the cloud and, if so, what cloud provider?

3. Technological considerations

Technologies need to be decided since architectural design, as changing them later is difficult and expensive

Technologies to consider:

- Database: relational or NoSQL?
- Delivery platform: browser-based or mobile?
- Server: using the cloud and, if so, what cloud provider?
- Use of open source software? Proprietary software?

3. Technological considerations

Technologies need to be decided since architectural design, as changing them later is difficult and expensive

Technologies to consider:

- Database: relational or NoSQL?
- Delivery platform: browser-based or mobile?
- Server: using the cloud and, if so, what cloud provider?
- Use of open source software?
- Development technology: mobile development toolkits, web application frameworks advantageous?

Resources

- Essential: Sommerville ESP Chapter 4
- Essential: Sommerville SE 6.1, 6.3.3