Software Engineering

Lesson #12 - Lecture

Lesson #12 - Lecture

Your KBTU 202309 Software Engineering class information is updating …

Lesson #12 update is in progress

This will take around 2 hours to complete

Please, don’t turn off your computer

Software Engineering

System Dependability and Security

COMPLETED

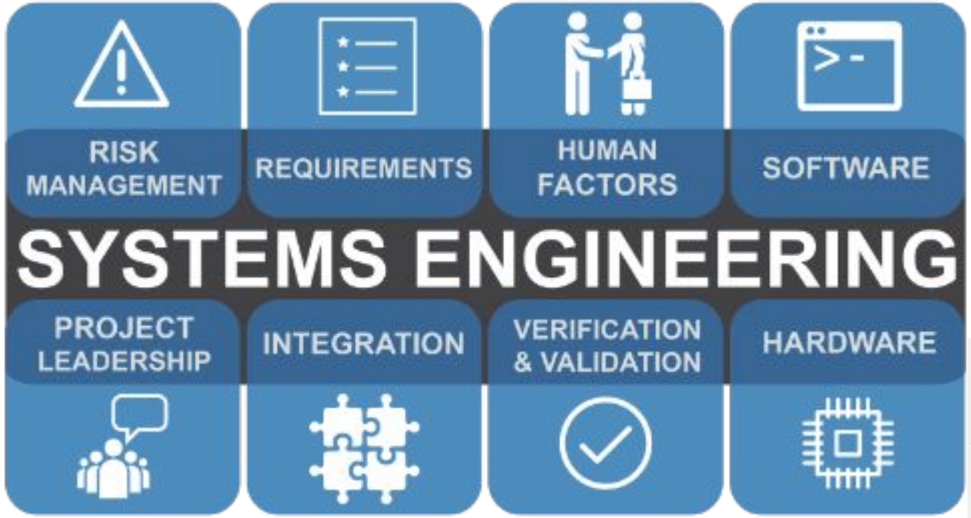
Part #02

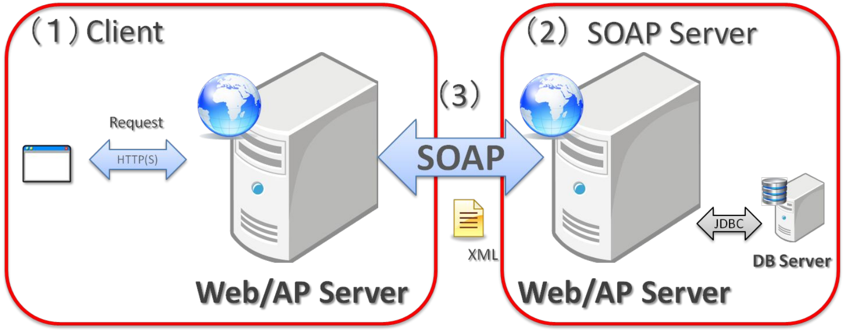
Software Engineering

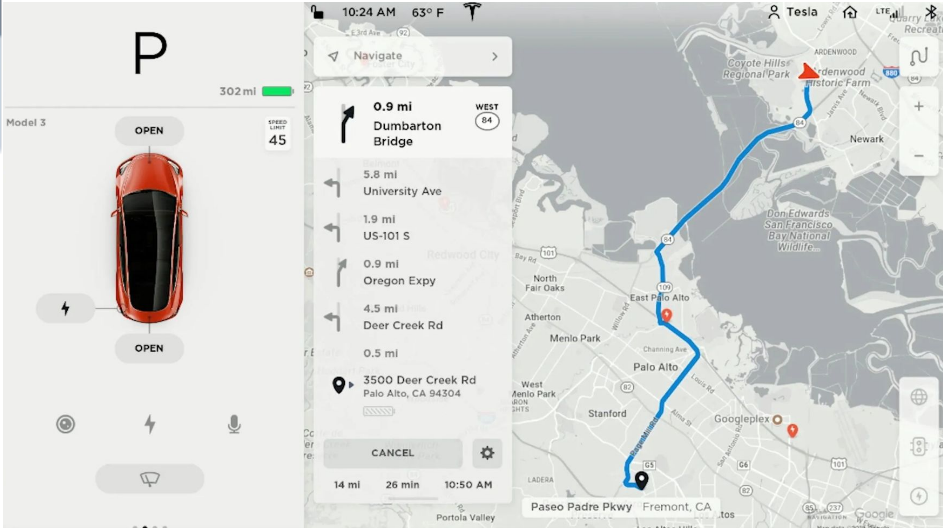
Advanced Software Engineering

Part #03

Part 3: Advanced Software Engineering







Agenda: Lesson #12 - Software Engineering - Lecture

Software Reuse

1. The reuse landscape
2. Application frameworks
3. Software product lines
4. Application system reuse

Agenda: Lesson #12 - Software Engineering - Lecture

Component based software engineering

1. Components and component models
2. CBSE processes
3. Component composition
4. Q & A

Agenda: Lesson #12 - Software Engineering - Lecture

Distributed software engineering

1	Distributed systems
2	Client–server computing
3	Architectural patterns for distributed systems
4	Software as a service

Part 3. Advanced Software Engineering

```
#include<iostream>
Using namespace std;

int main()
{
	cout << “Software Reuse” << endl;

	return 0;
}
```

Software Reuse

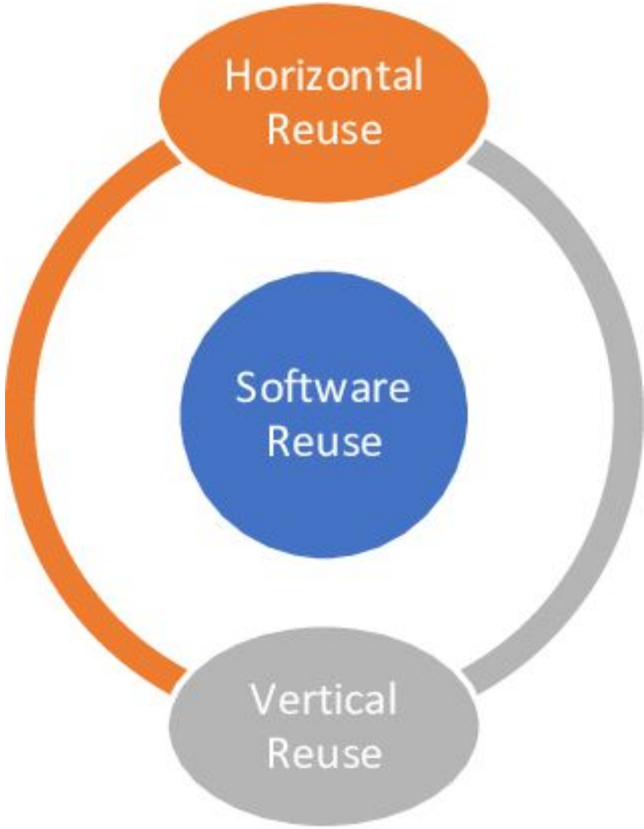
Software Reuse

In most engineering disciplines, systems are designed by composing existing components that have been used in other systems

Software engineering has been more focused on original development but it is now recognised that to achieve better software, more quickly and at lower cost, we need a design process that is based on systematic software reuse

Software Reuse

There has been a major switch to reuse-based development over the past 10 years



Software Reuse

Software -> High quality

Software -> faster dev.

Software -> Low cost

Reuse-based software engineering

System reuse

- Complete systems, which may include several application programs may be reused

Application reuse

- An application may be reused either by incorporating it without change into other or by developing application families

Reuse-based software engineering

Component reuse

- Components of an application from sub-systems to single objects may be reused

Object and function reuse

- Small-scale software components that implement a single well-defined object or function may be reused

Software Reuse

Benefits of software reuse

- Accelerated development
- Effective use of specialists
- Increased dependability
- Lower development costs
- Reduced process risk
- Standards compliance

Problems with reuse

- Creating, maintaining, and using a component library
- Finding, understanding, and adapting reusable components
- Increased maintenance costs
- Lack of tool support
- Not-invented-here syndrome

Agenda: Lesson #12 - Software Engineering - Lecture

Software Reuse

The reuse landscape

Although reuse is often simply thought of as the reuse of system components, there are many different approaches to reuse that may be used

Reuse is possible at a range of levels from simple functions to complete application systems

The reuse landscape covers the range of possible reuse techniques

Software Reuse

Reuse Landscape

https://www.youtube.com/watch?v=8Io6IRiwYio

Reuse planning factors

- The development schedule for the software
- The expected software lifetime
- The background, skills and experience of the development team
- The criticality of the software and its non-functional requirements
- The application domain
- The execution platform for the software

Agenda: Lesson #12 - Software Engineering - Lecture

Software Reuse

Framework definition

“..an integrated set of software artefacts (such as classes, objects and components) that collaborate to provide a reusable architecture for a family of related applications.”

Application frameworks

Frameworks are moderately large entities that can be reused. They are somewhere between system and component reuse

Frameworks are a sub-system design made up of a collection of abstract and concrete classes and the interfaces between them

The sub-system is implemented by adding components to fill in parts of the design and by instantiating the abstract classes in the framework

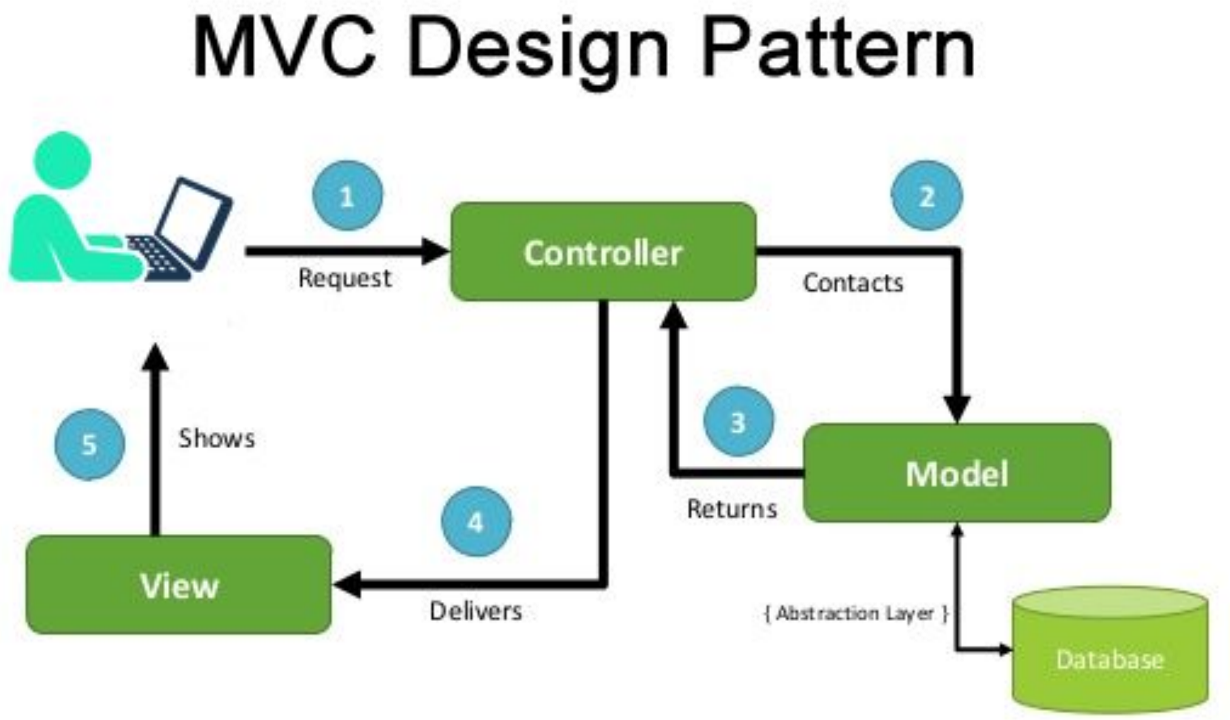
Web application frameworks

Support the construction of dynamic websites as a front-end for web applications

WAFs are now available for all of the commonly used web programming languages e.g. Java, Python, Ruby, etc.

Interaction model is based on the Model-View-Controller composite pattern

Web application frameworks



Top 10 Web Application Frameworks

React.js

jQuery

Angular

Vue.js

ASP.NET

Express.js

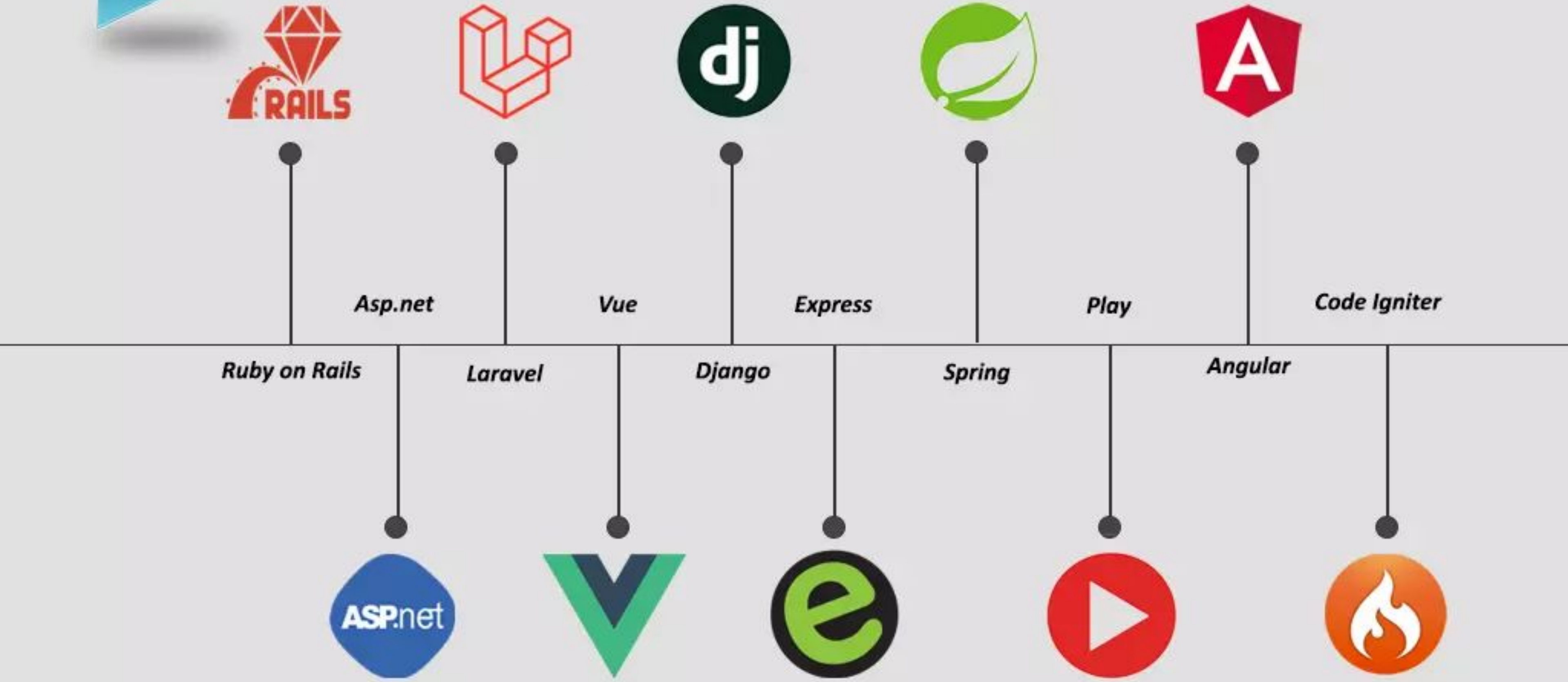
django

ASP.NET Core

Spring

Flask

Top 10 Frameworks for Web Application Development



Agenda: Lesson #12 - Software Engineering - Lecture

Software Reuse

Software product lines

Software product lines or application families are applications with generic functionality that can be adapted and configured for use in a specific context

A software product line is a set of applications with a common architecture and shared components, with each application specialized to reflect different requirements

Software product lines

Adaptation may involve:

- Component and system configuration;
- Adding new components to the system;
- Selecting from a library of existing components;
- Modifying components to meet new requirements.

Base applications

Core components that provide infrastructure support. These are not usually modified when developing a new instance of the product line

Configurable components that may be modified and configured to specialize them to a new application. Sometimes, it is possible to reconfigure these components without changing their code by using a built-in component configuration language

Specialized, domain-specific components some or all of which may be replaced when a new instance of a product line is created

Application frameworks & product lines

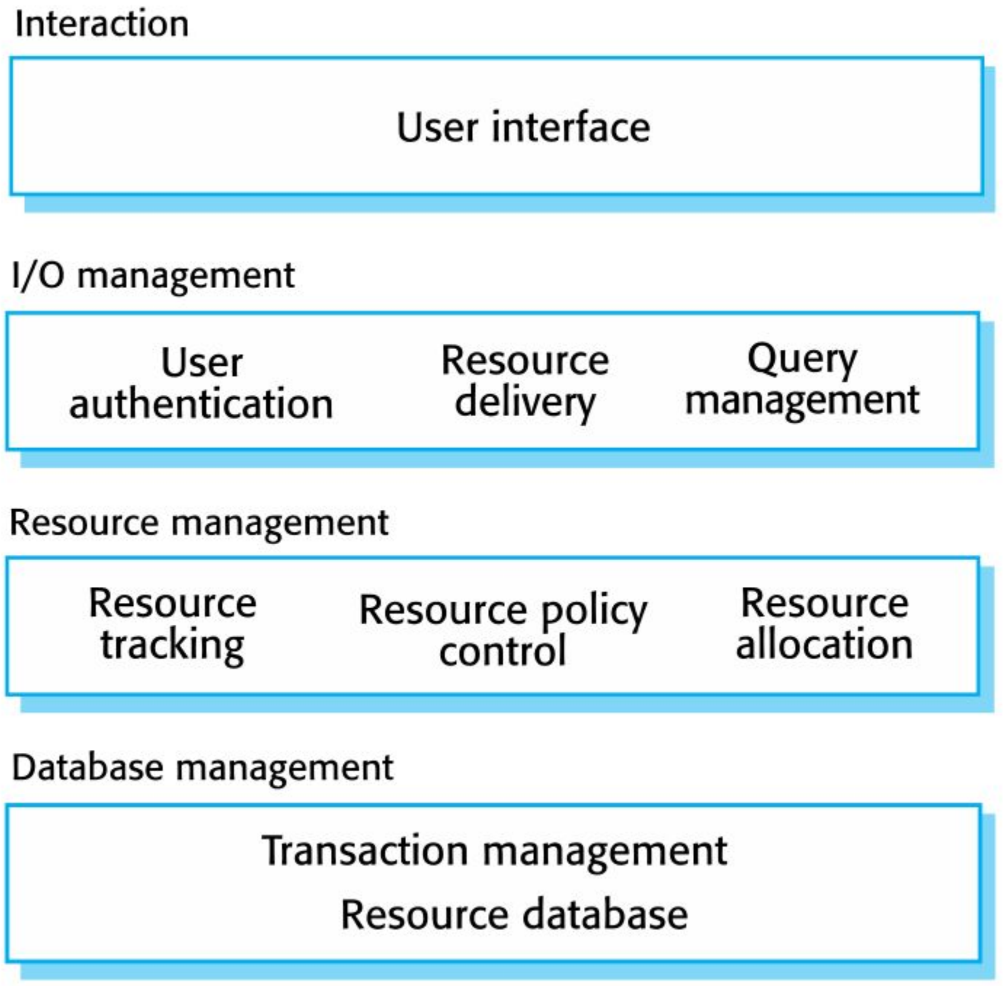
- Application frameworks rely on object-oriented features such as polymorphism to implement extensions. Product lines need not be object-oriented (e.g. embedded software for a mobile phone)
- Application frameworks focus on providing technical rather than domain-specific support. Product lines embed domain and platform information
- Product lines often control applications for equipment
- Software product lines are made up of a family of applications, usually owned by the same organization

Product line architectures

Architectures must be structured in such a way to separate different subsystems and to allow them to be modified

The architecture should also separate entities and their descriptions and the higher levels in the system access entities through descriptions rather than directly

The architecture of a resource allocation system



Product line architectures

Architectures must be structured in such a way to separate different subsystems and to allow them to be modified

The architecture should also separate entities and their descriptions and the higher levels in the system access entities through descriptions rather than directly

Android Auto	Android OS	Android TV	Calendar	Cardboard	Chrome
Chrome Web Store	Chromebook	Chromecast	Connected Home	Contacts	Docs
Drawings	Drive	Earth	Finance	Forms	Gboard
Gmail	Google Alerts	Google Assistant	Google Cast	Google Chat	Google Classroom
Google Cloud Print	Google Duo	Google Expeditions	Google Express	Google Fi	Google Fit
Google Flights	Google Fonts	Google Groups	Google Input Tools	Google Meet	Google One

Microsoft

Bookings	Calendar	Delve	Excel	Forms	Kaizala	Lists	MyAnalytics	OneDrive
OneNote	Outlook	People	Planner	Power Apps	Power Autom...	Power BI	PowerPoint	Project
SharePoint	Stream	Sway	Teams	To Do	Visio	Whiteboard	Word	Zoom for Outl...

Agenda: Lesson #12 - Software Engineering - Lecture

Software Reuse

Application system reuse

An application system product is a software system that can be adapted for different customers without changing the source code of the system

Application systems have generic features and so can be used/reused in different environments

Application system reuse

Application system products are adapted by using built-in configuration mechanisms that allow the functionality of the system to be tailored to specific customer needs

- For example, in a hospital patient record system, separate input forms and output reports might be defined for different types of patient

Benefits of application system reuse

As with other types of reuse, more rapid deployment of a reliable system may be possible

It is possible to see what functionality is provided by the applications and so it is easier to judge whether or not they are likely to be suitable

Some development risks are avoided by using existing software. However, this approach has its own risks, as I discuss below

Benefits of application system reuse

Businesses can focus on their core activity without having to devote a lot of resources to IT systems development

As operating platforms evolve, technology updates may be simplified as these are the responsibility of the COTS product vendor rather than the customer

Problems of application system reuse

Requirements usually have to be adapted to reflect the functionality and mode of operation of the COTS product

The COTS product may be based on assumptions that are practically impossible to change

Choosing the right COTS system for an enterprise can be a difficult process, especially as many COTS products are not well documented

Problems of application system reuse

There may be a lack of local expertise to support systems development

The COTS product vendor controls system support and evolution

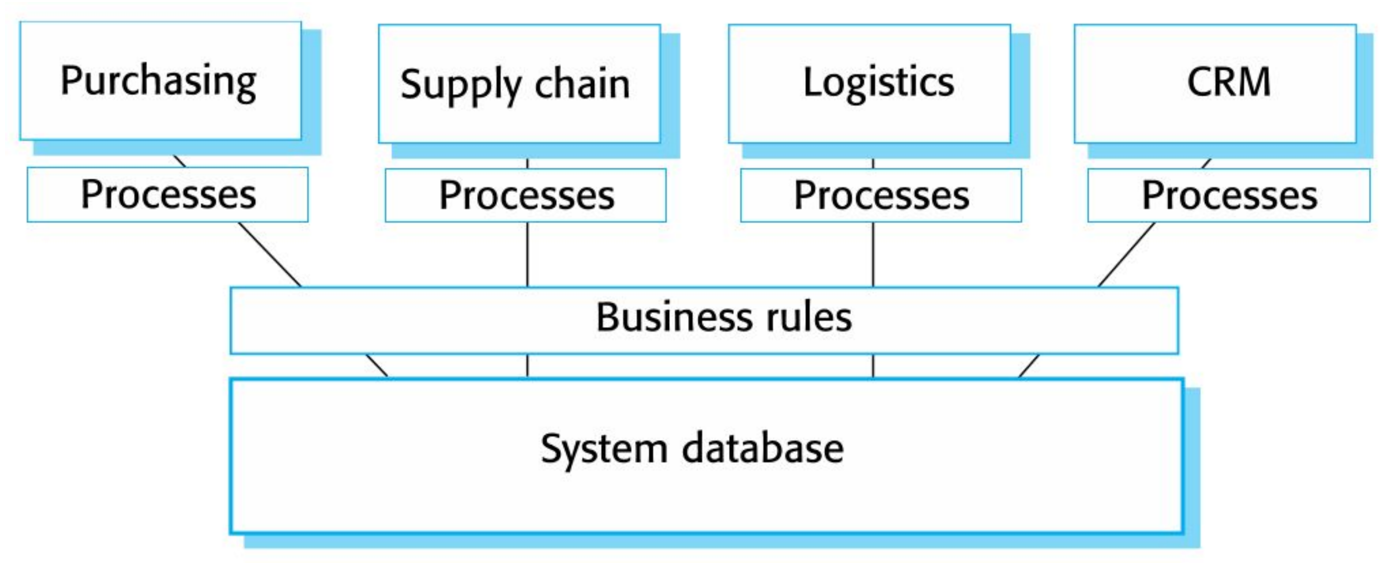
ERP systems

An Enterprise Resource Planning (ERP) system is a generic system that supports common business processes such as ordering and invoicing, manufacturing, etc.

These are very widely used in large companies - they represent probably the most common form of software reuse

The generic core is adapted by including modules and by incorporating knowledge of business processes and rules

The architecture of an ERP system



ERP architecture

A number of modules to support different business functions

A defined set of business processes, associated with each module, which relate to activities in that module

A common database that maintains information about all related business functions

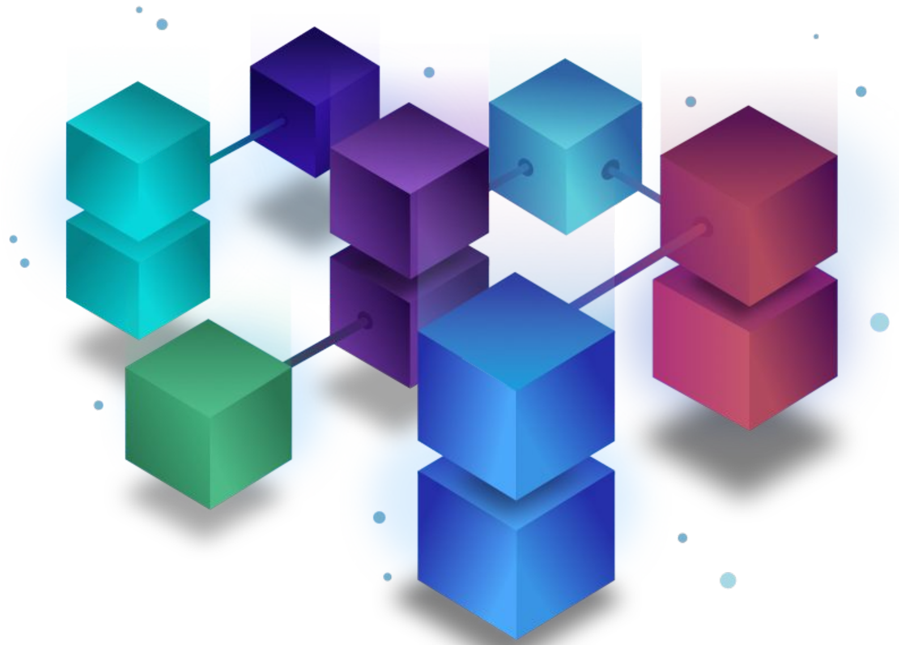
A set of business rules that apply to all data in the database

Part 3. Advanced Software Engineering

```
#include<iostream>
Using namespace std;

int main()
{
	cout << “Component based software engineering” << endl;

	return 0;
}
```

Agenda: Lesson #12 - Software Engineering - Lecture

Component based software engineering

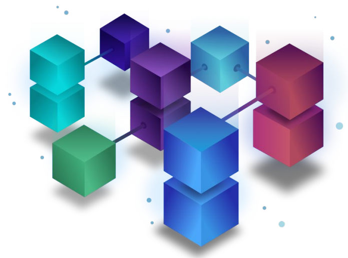


Component-based development

Component-based software engineering (CBSE) is an approach to software development that relies on the reuse of entities called 'software components'

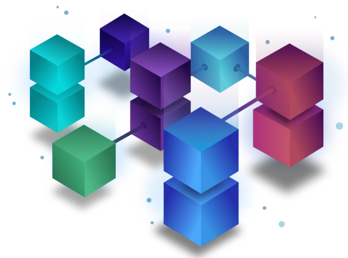
It emerged from the failure of object-oriented development to support effective reuse. Single object classes are too detailed and specific

Components are more abstract than object classes and can be considered to be stand-alone service providers. They can exist as stand-alone entities



CBSE essentials

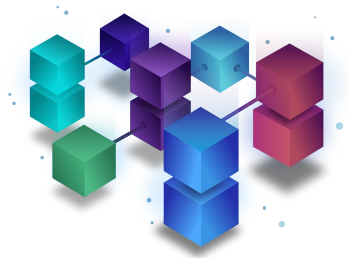
- Independent components specified by their interfaces
- Component standards to facilitate component integration
- Middleware that provides support for component interoperability
- A development process that is geared to reuse



CBSE and design principles

Apart from the benefits of reuse, CBSE is based on sound software engineering design principles:

- Components are independent so do not interfere with each other;
- Component implementations are hidden;
- Communication is through well-defined interfaces;
- One components can be replaced by another if its interface is maintained;
- Component infrastructures offer a range of standard services

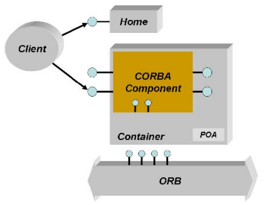


Component standards

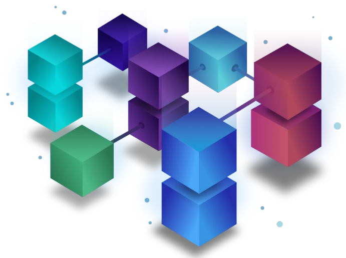
Standards need to be established so that components can communicate with each other and inter-operate

Unfortunately, several competing component standards were established:

- Sun's Enterprise Java Beans
- Microsoft's COM and .NET
- CORBA's CCM



In practice, these multiple standards have hindered the uptake of CBSE. It is impossible for components developed using different approaches to work together

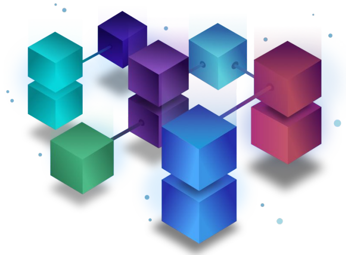


Service-oriented software engineering

An executable service is a type of independent component. It has a 'provides' interface but not a 'requires' interface

From the outset, services have been based around standards so there are no problems in communicating between services offered by different vendors

System performance may be slower with services but this approach is replacing CBSE in many systems



Agenda: Lesson #12 - Software Engineering - Lecture

Component based software engineering

Components

Components provide a service without regard to where the component is executing or its programming language

- A component is an independent executable entity that can be made up of one or more executable objects;

- The component interface is published and all interactions are through the published interface;

Component definitions

Councill and Heinmann:

- A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard

Component definitions

Szyperski:

- A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third-parties.

Component as a service provider

The component is an independent, executable entity. It does not have to be compiled before it is used with other components

The services offered by a component are made available through an interface and all component interactions take place through that interface

The component interface is expressed in terms of parameterized operations and its internal state is never exposed

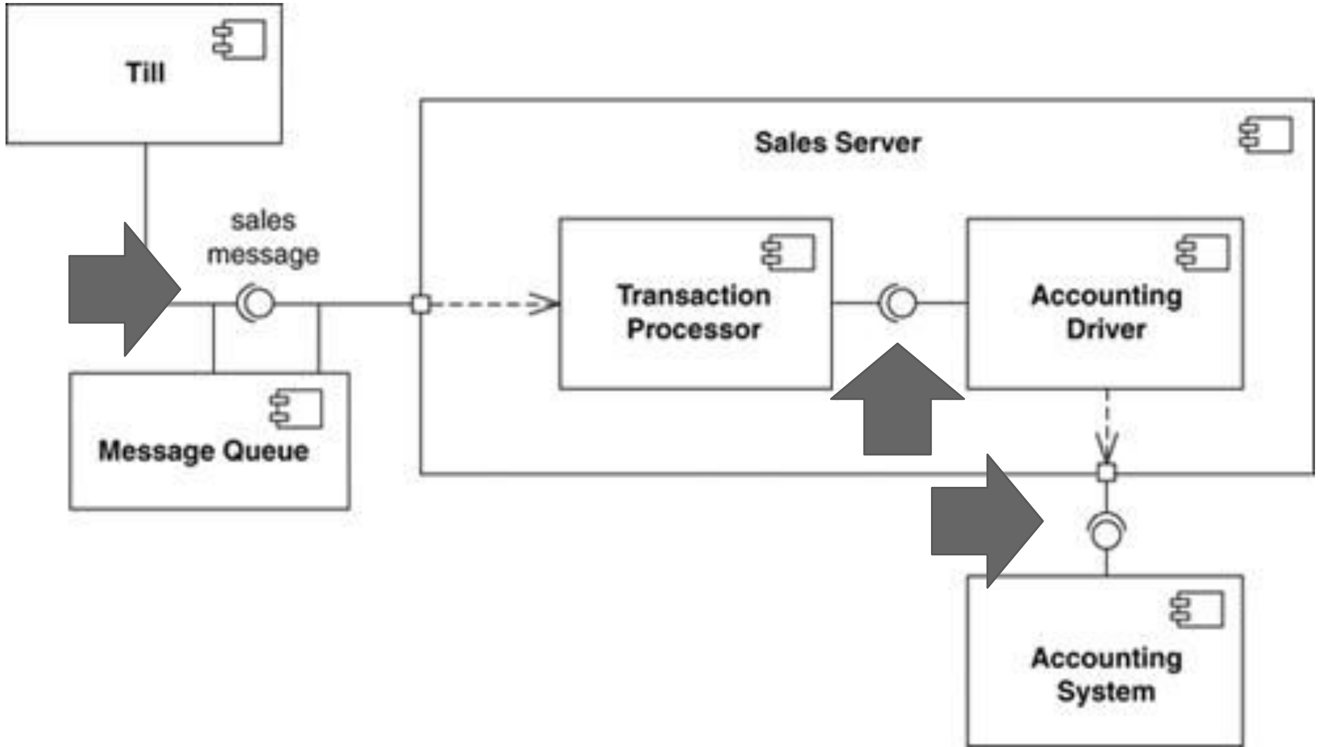
Component interfaces

- Provides interface
    - Defines the services that are provided by the component to other components
    - This interface, essentially, is the component API. It defines the methods that can be called by a user of the component.

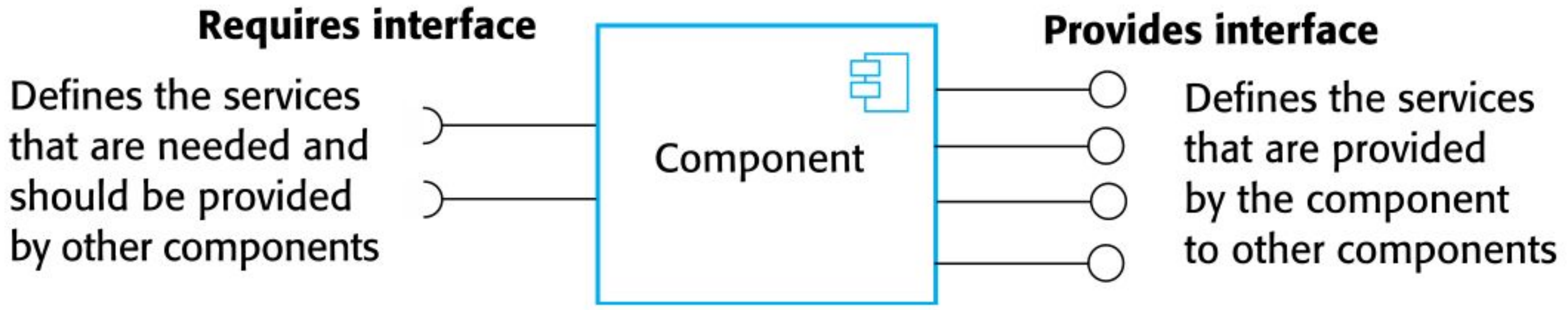
Component interfaces

- Requires interface
    - Defines the services that specifies what services must be made available for the component to execute as specified.
    - This does not compromise the independence or deployability of a component because the ‘requires’ interface does not define how these services should be provided

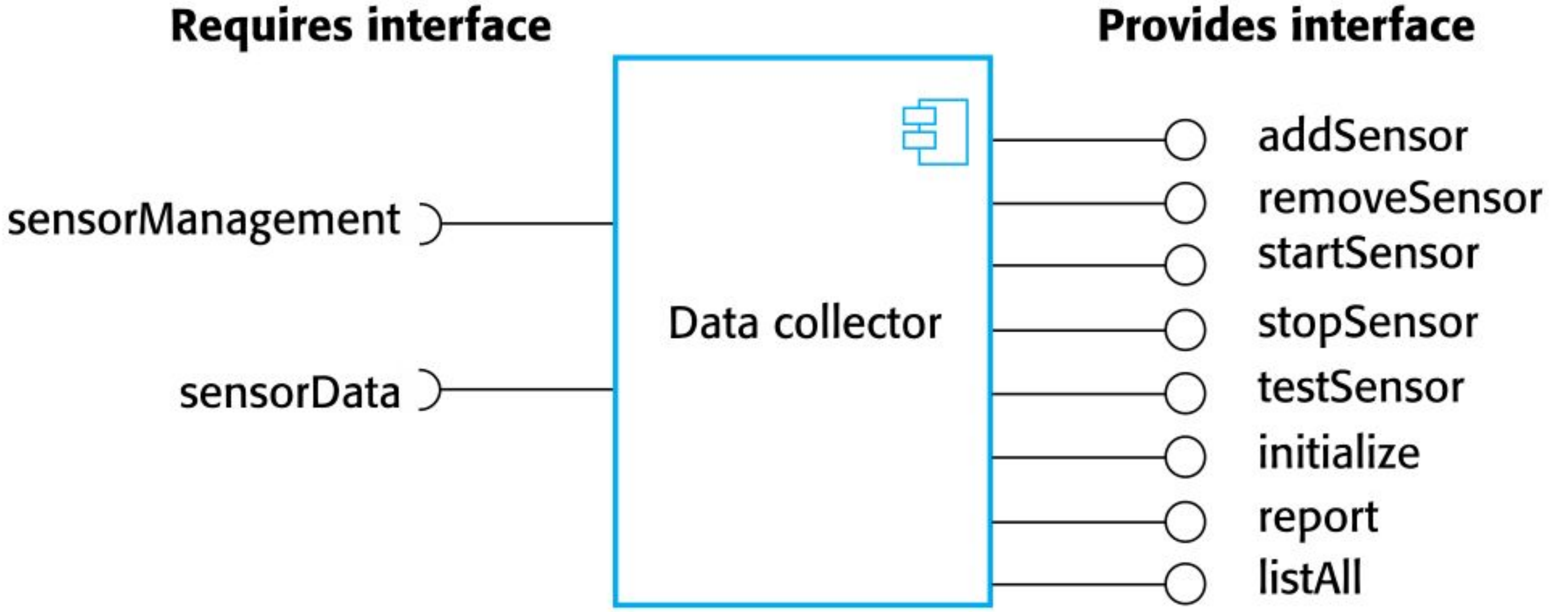
Component interfaces



Component interfaces


Requires interface
Defines the services that are needed and should be provided by other components
Component
Provides interface
Defines the services that are provided by the component to other components

A model of a data collector component



Component access

Components are accessed using remote procedure calls (RPCs)

Each component has a unique identifier (usually a URL) and can be referenced from any networked computer

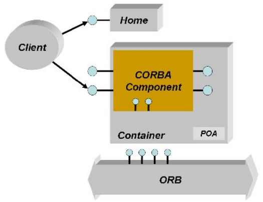
Therefore it can be called in a similar way as a procedure or method running on a local computer

Component models

A component model is a definition of standards for component implementation, documentation and deployment

Examples of component models

- EJB model (Enterprise Java Beans)
- COM+ model (.NET model)
- Corba Component Model



The component model specifies how interfaces should be defined and the elements that should be included in an interface definition

Agenda: Lesson #12 - Software Engineering - Lecture

Component based software engineering

1. Components and component models
2. **CBSE processes**
3. Component composition
4. Q & A

CBSE processes

CBSE processes are software processes that support component - based software engineering

- They take into account the possibilities of reuse and the different process activities involved in developing and using reusable components

CBSE processes

Development for reuse

- This process is concerned with developing components or services that will be reused in other applications. It usually involves generalizing existing components

Development with reuse

- This process is the process of developing new applications using existing components and services

CBSE for reuse

CBSE for reuse focuses on component development

Components developed for a specific application usually have to be generalised to make them reusable

A component is most likely to be reusable if it associated with a stable domain abstraction (business object)

For example, in a hospital stable domain abstractions are associated with the fundamental purpose - nurses, patients, treatments, etc.

Agenda: Lesson #12 - Software Engineering - Lecture

Component based software engineering

Component composition

The process of assembling components to create a system

Composition involves integrating components with each other and with the component infrastructure

Normally you have to write ‘glue code’ to integrate components

Types of composition

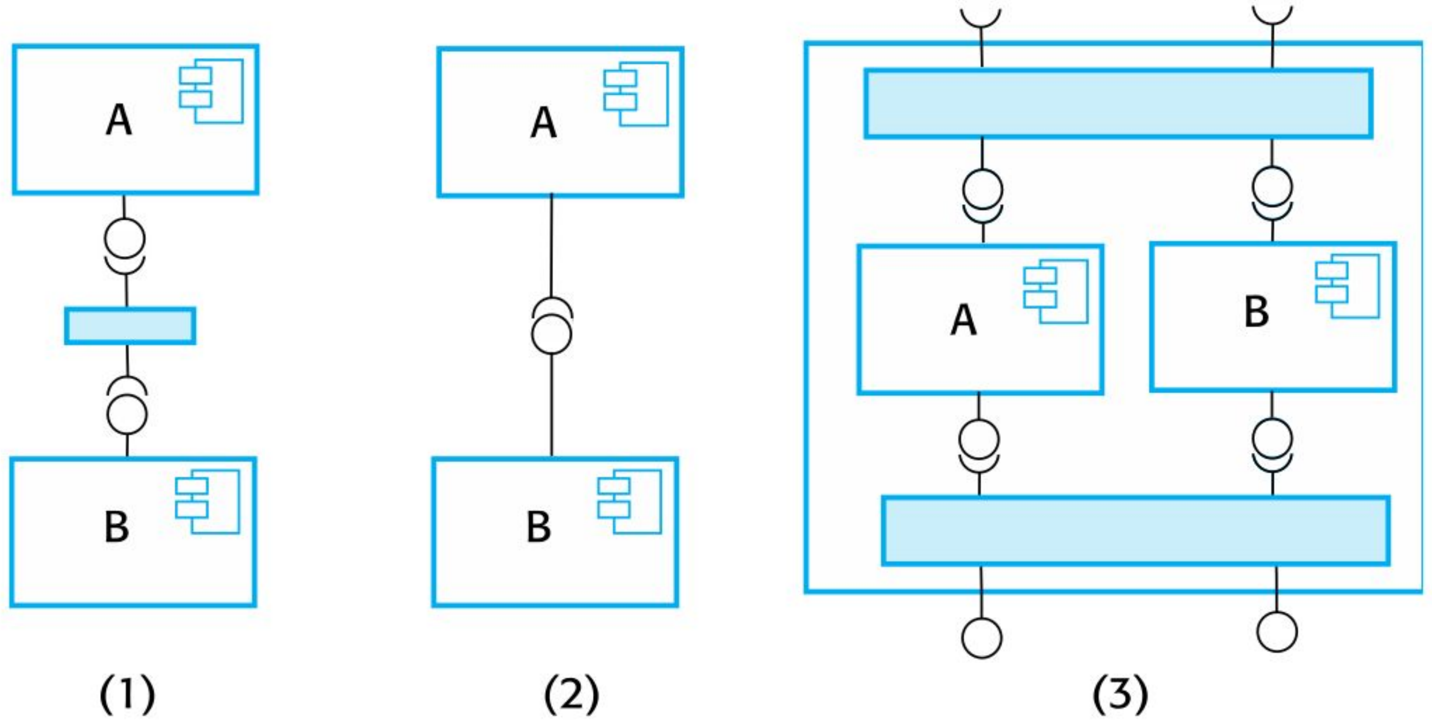
- Sequential composition (1) where the composed components are executed in sequence. This involves composing the provides interfaces of each component

- Hierarchical composition (2) where one component calls on the services of another. The provides interface of one component is composed with the requires interface of another

Types of composition

- Additive composition (3) where the interfaces of two components are put together to create a new component. Provides and requires interfaces of integrated component is a combination of interfaces of constituent components.

Types of composition

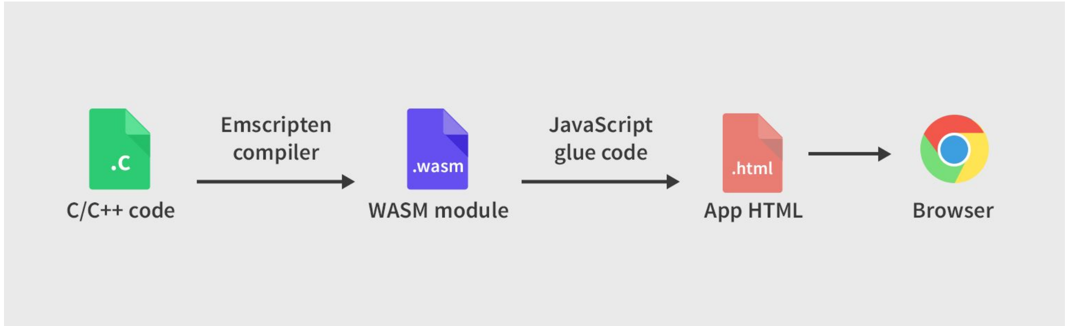


Glue code

Code that allows components to work together

If A and B are composed sequentially, then glue code has to call A, collect its results then call B using these results, transforming them into the format required by B

Glue code may be used to resolve interface incompatibilities



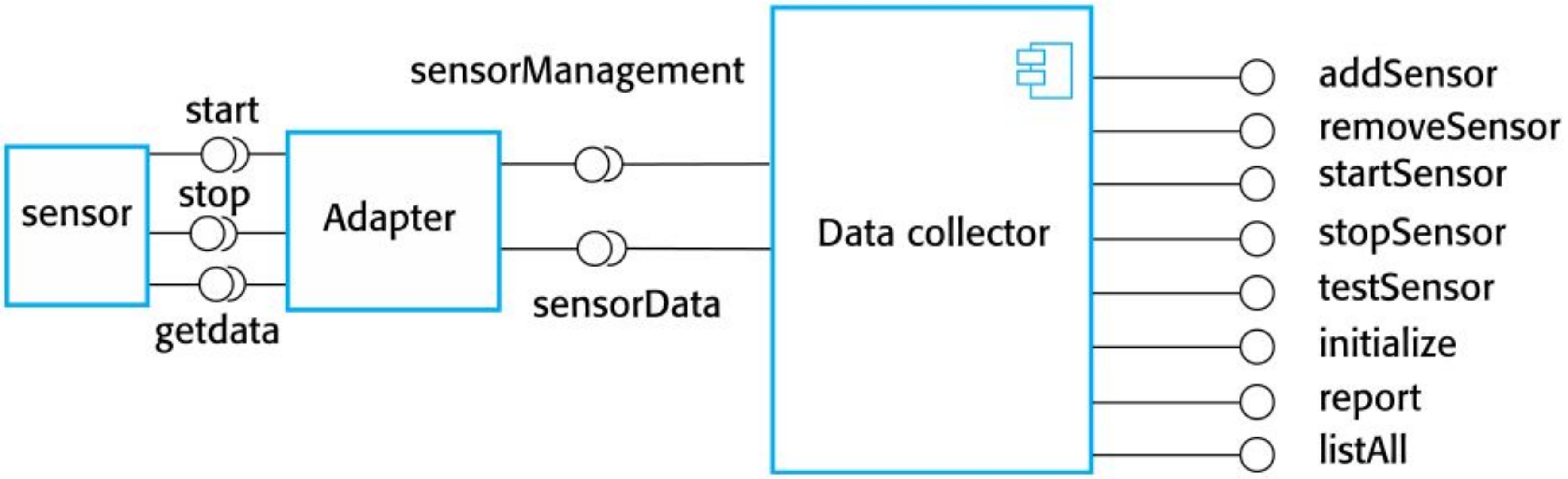
Adaptor components

Address the problem of component incompatibility by reconciling the interfaces of the components that are composed

Different types of adaptor are required depending on the type of composition

An addressFinder and a mapper component may be composed through an adaptor that strips the postal code from an address and passes this to the mapper component

An adaptor linking a data collector and a sensor

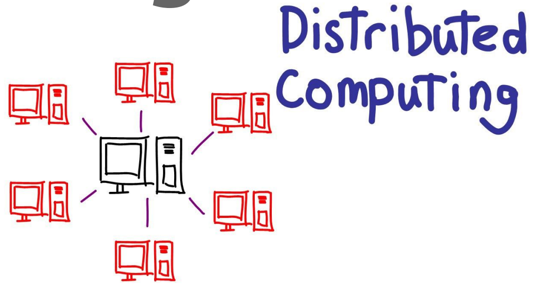


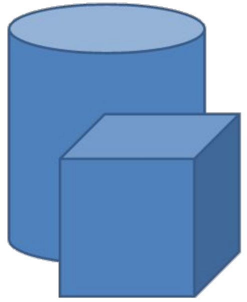
Part 3. Advanced Software Engineering

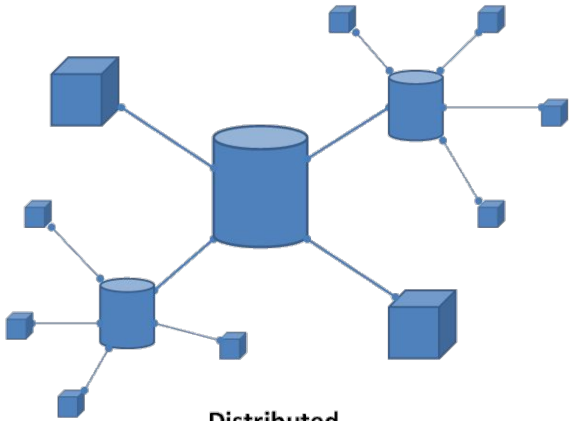
```
#include<iostream>
Using namespace std;

int main()
{
      cout << “Distributed software engineering” << endl;

      return 0;
}
```

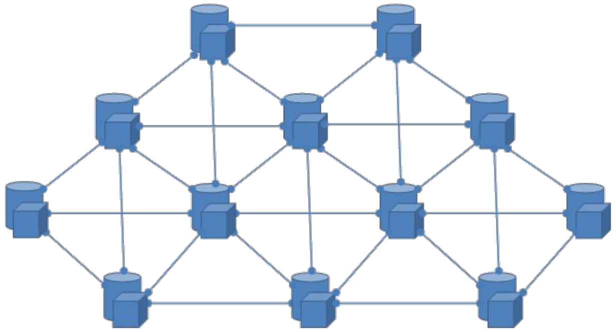
Agenda: Lesson #12 - Software Engineering - Lecture

Distributed software engineering




Centralized
one node does everything


Distributed
nodes distribute work to sub-nodes


Decentralized
nodes are only connected to peers

Distributed systems

Virtually all large computer-based systems are now distributed systems

- "… a collection of independent computers that appears to the user as a single coherent system."

Information processing is distributed over several computers rather than confined to a single machine

Distributed software engineering is therefore very important for enterprise computing systems

Distributed system characteristics

- Resource sharing
  - Sharing of hardware and software resources
- Openness
  - Use of equipment and software from different vendors
- Concurrency
  - Concurrent processing to enhance performance
- Scalability
  - Increased throughput by adding new resources
- Fault tolerance
  - The ability to continue in operation after a fault has occurred

Metaverse

TAMASHII NATIONS
GUNPLA
ANAHEIM ELECTRONICS
※画像はイメージです
©SOTSU・SUNRISE
©SOTSU・SUNRISE・MBS

Agenda: Lesson #12 - Software Engineering - Lecture

Distributed software engineering

Distributed systems issues

Distributed systems are more complex than systems that run on a single processor

Complexity arises because different parts of the system are independently managed as is the network

There is no single authority in charge of the system so top-down control is impossible

Agenda: Lesson #12 - Software Engineering - Lecture

Distributed software engineering

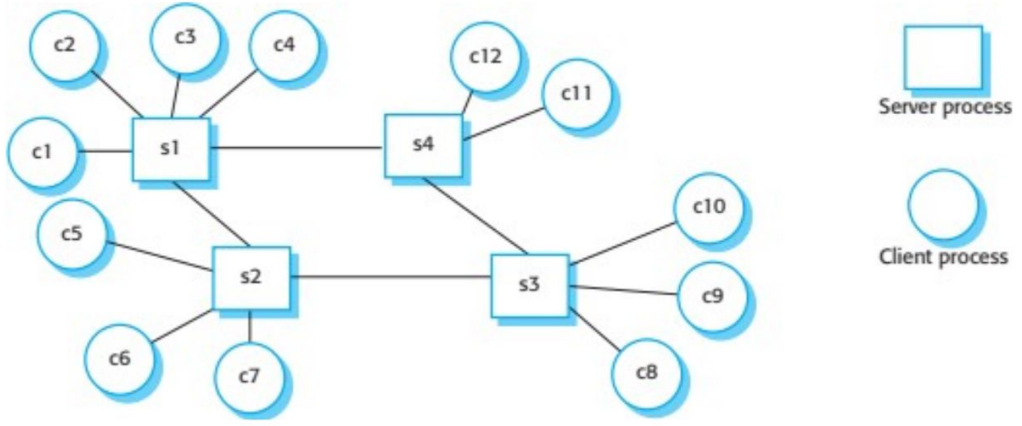
Client-server computing

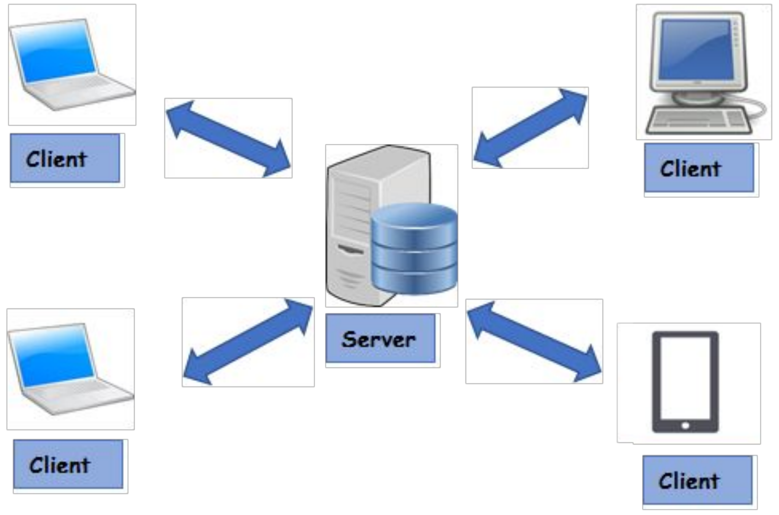
Distributed systems that are accessed over the Internet are normally organized as client-server systems

In a client-server system, the user interacts with a program running on their local computer (e.g. a web browser or mobile application). This interacts with another program running on a remote computer (e.g. a web server)

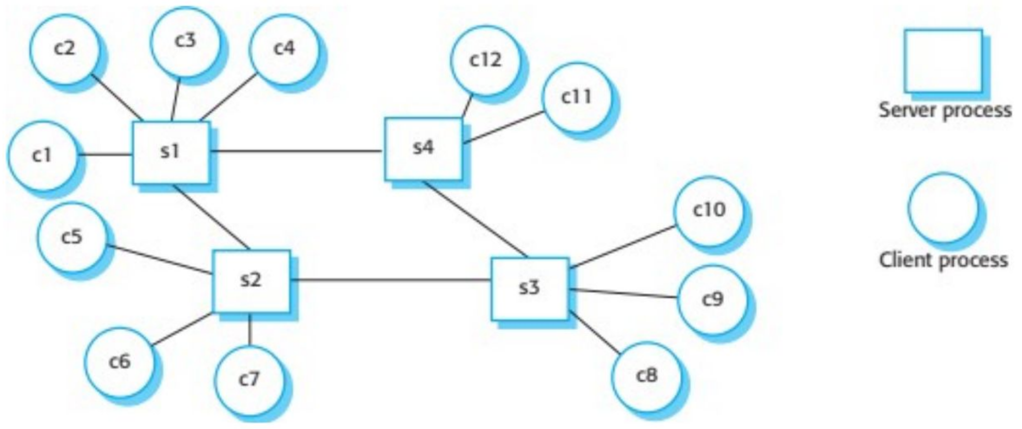
The remote computer provides services, such as access to web pages, which are available to external clients

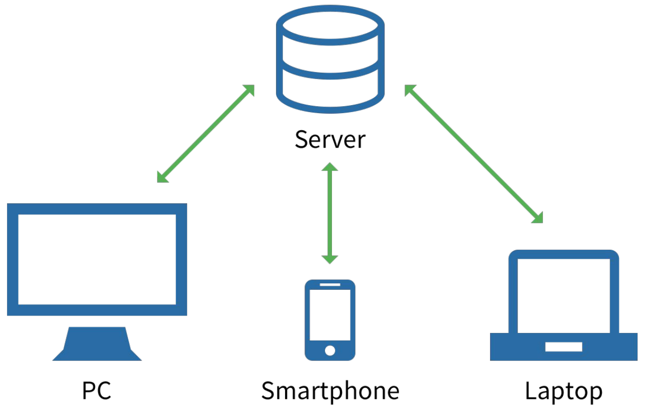
Client-server computing





Client-server computing





Agenda: Lesson #12 - Software Engineering - Lecture

Distributed software engineering

Architectural patterns for D. S.

Widely used ways of organizing the architecture of a distr. system:

- Master-slave architecture, which is used in real-time systems in which guaranteed interaction response times are required
- Two-tier client-server architecture, which is used for simple client-server systems, and where the system is centralized for security reasons

Architectural patterns for D. S.

Widely used ways of organizing the architecture of a distr. system:

- Multi-tier client-server architecture, which is used when there is a high volume of transactions to be processed by the server

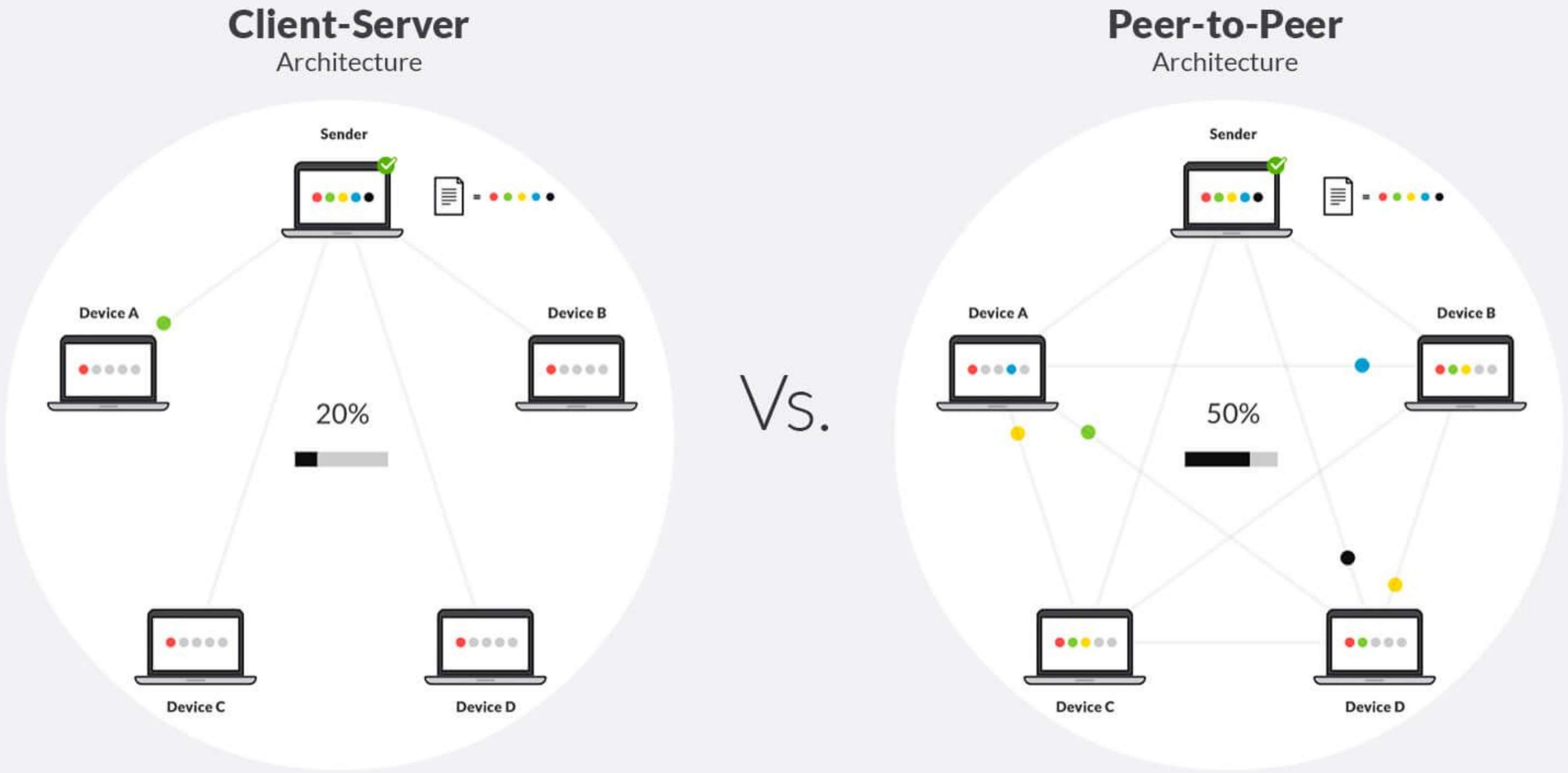
- Distributed component architecture, which is used when resources from different systems and databases need to be combined, or as an implementation model for multi-tier client-server systems

Architectural patterns for D. S.

Widely used ways of organizing the architecture of a distr. system:

- Peer-to-peer architecture, which is used when clients exchange locally stored information and the role of the server is to introduce clients to each other

Distributed software engineering

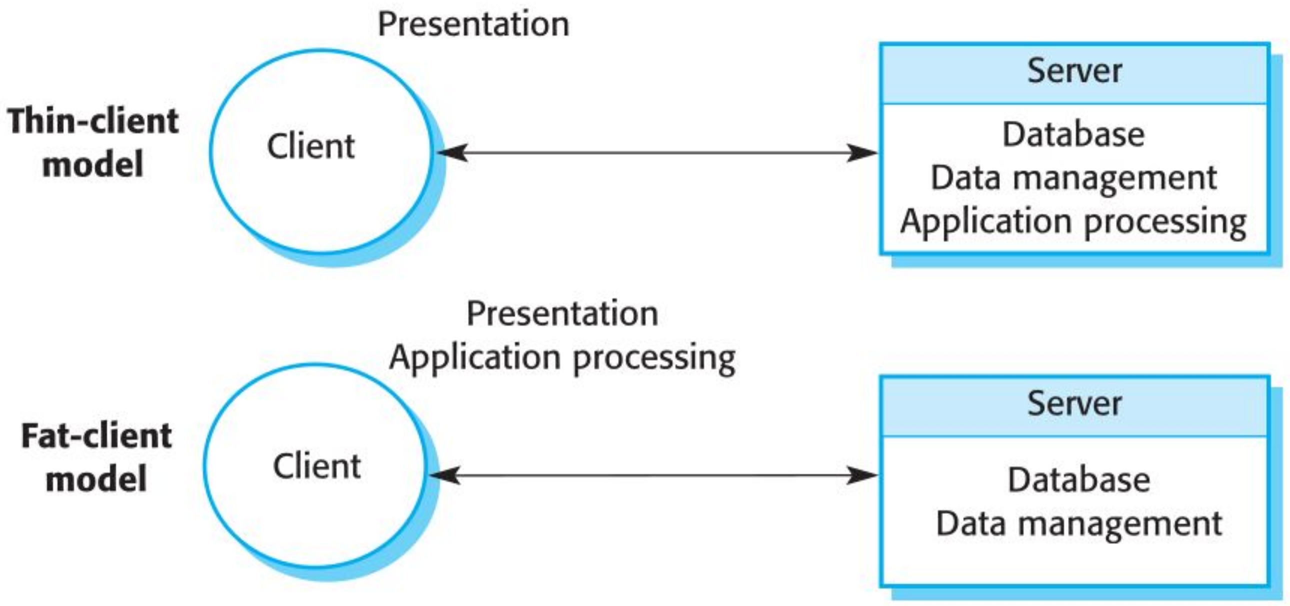


Two-tier client server architectures

In a two-tier client-server architecture, the system is implemented as a single logical server plus an indefinite number of clients that use that server

- Thin-client model, where the presentation layer is implemented on the client and all other layers (data management, application processing and database) are implemented on a server
- Fat-client model, where some or all of the application processing is carried out on the client. Data management and database functions are implemented on the server

Two-tier client server architectures



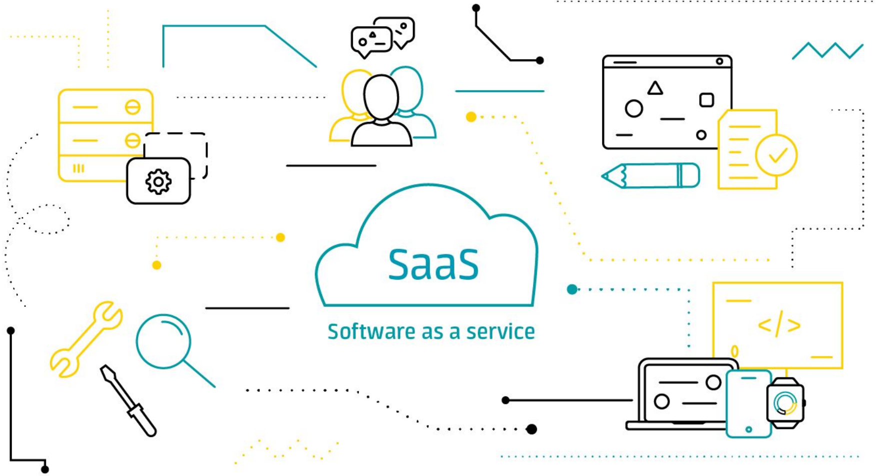
Agenda: Lesson #12 - Software Engineering - Lecture

Distributed software engineering

1	Distributed systems
2	Client–server computing
3	Architectural patterns for distributed systems
4	**Software as a service**

Software as a service

Software as a service (SaaS /sæs/) is a software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted



SaaS Software	VS	Traditional Software
Subscription Users subscribe to the software without paying any money up front		***One-time fee*** Users purchase the software up front and install it on their own computers
Multiple devices Applications can be used across multiple devices with a single login, the application can be updated online instantaneously		***Single device*** Licensed individually and usually limited to a single device and when updates come out, they must be downloaded or purchased and installed

Agenda: Lesson #12 - Software Engineering - Lecture

Q & A