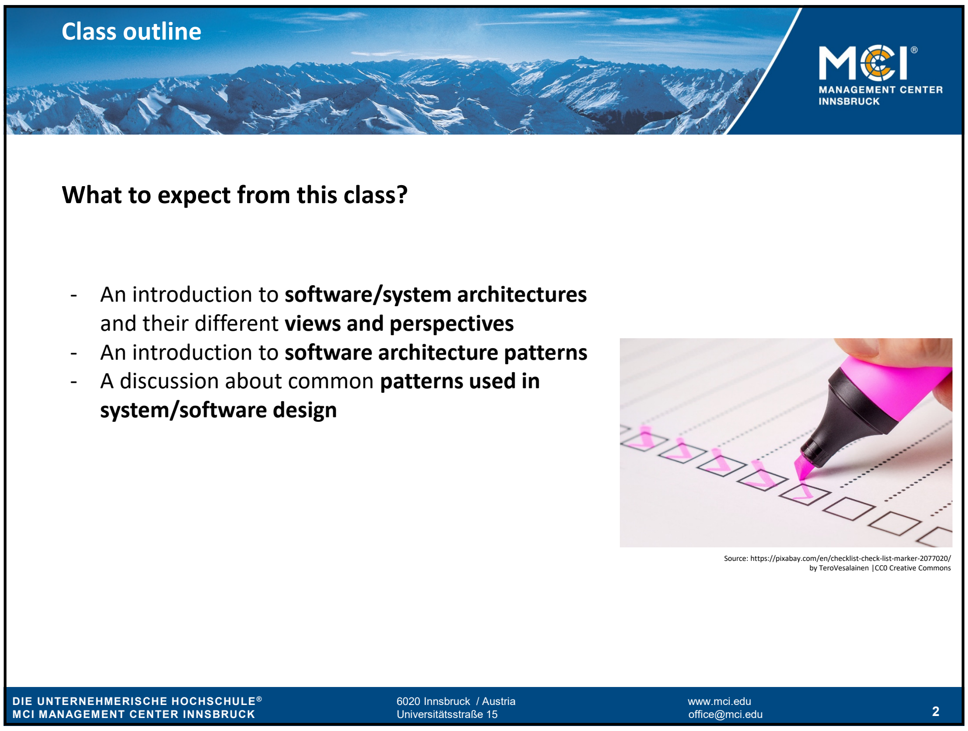
MANAGEMENT CENTER INNSBRUCK

SYSTEMS PLANNING

Architectural Design



Class outline

MANAGEMENT CENTER INNSBRUCK

What to expect from this class?

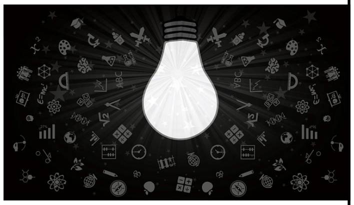
- An introduction to **software/system architectures** and their different **views and perspectives**
- An introduction to **software architecture patterns**
- A discussion about common **patterns used in system/software design**

Source: https://pixabay.com/en/checklist-check-list-marker-2077020/
by TeroVesalainen |CC0 Creative Commons

Class learning goals

What should be your learning outcome?

- Understand **the goal of software architecture**
- Know the **purpose of an architectural pattern**
- Know **common software/system patterns**


Source: https://pixabay.com/en/learning-hint-school-subject-3245792/
by harishs |CC0 Creative Commons

Architectural Design

What is architectural design?

- Concerned with **understanding how a system should be organized**
- The **overall structure** of that system
- Critical **link between design and requirements engineering**
- **Output** is an **architectural model**

In agile processes
- Early stage development is concerned with an **overall system architecture**
- **Incremental development** of an architecture is **not usually successful**

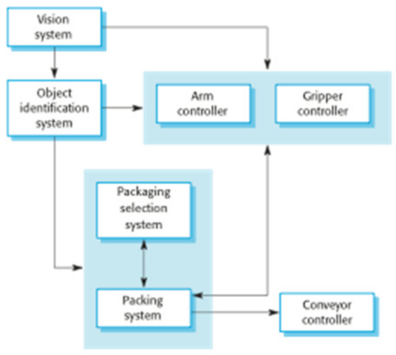
Note: Refactoring of components is easy, refactoring a system architecture is more difficult!

Source: https://pixabay.com/en/urban-sky-buildings-city-205986/
by Ichigo121212 |CC0 Creative Commons

Architectural Design

Example: Packing robot system

- **Abstract architecture**
- System can **pack different kinds of objects**
- Uses a **vision component** to:
  pick out objects
  identify the **type** of object
  select the right kind of **packaging**
- System **moves objects from the delivery conveyor to be packaged**



Source: Sommerville, I. (2011). Software Engineering, 9th Ed. Boston, MA, USA: Pearson Education Inc.

Architectural Design

Requirements Engineering vs. Architectural Design

- In practice there is a **significant overlap**
- Ideally a **system specification** should **not include any design information**
- Yet, this is **unrealistic** except for **very small systems**
- **Architectural decomposition** is usually necessary in order to **discuss requirements and features of a system with stakeholders**

Source: https://pixabay.com/en/smoothies-fruits-colorful-vitamins-2253423/
by silviarita |CC0 Creative Commons

Architectural Design

MANAGEMENT CENTER INNSBRUCK

Architectures at two levels of abstraction

Architecture in the small
- Architecture of individual programs
- How are these programs decomposed into components

Architecture in the large
- Architecture of complex enterprise systems, programs and program components
- Systems are usually distributed over different computers which may be owned and managed by different companies

Source: https://pixabay.com/en/monitor-binary-binary-system-1307227/
by geralt | CC0 Creative Commons

Architectural Design

MANAGEMENT CENTER INNSBRUCK

The Importance of Software Architecture

Software Architecture is important because it affects:
- Performance
- Robustness
- Distributability
- Maintainability

Source: https://pixabay.com/en/important-stamp-imprint-1705212/
by geralt | CC0 Creative Commons

Note: Individual components implement the functional system requirements; the non-functional requirements depend on the system architecture (Bosch, 2000)!

Architectural Design

MCI MANAGEMENT CENTER INNSBRUCK

Three advantages of explicit software architecture (Bass et al., 2003)

- **Stakeholder communication**
  may be used as a focus for discussion by a range of different stakeholders
- **System analysis**
  Making the system architecture explicit early, helps if the development requires some analysis.
  Architectural decisions may have profound effects on critical requirements (performance, reliability, etc.)
- **Large-scale reuse**
  An explicit system architecture can often support large-scale reuse

Source: https://pixabay.com/en/like-facebook-social-media-icon-3000958/ by raphaelsilva |CC0 Creative Commons

Architectural Design

MCI MANAGEMENT CENTER INNSBRUCK

Hofmeister et al., 2003

Software architecture can serve:
1) As a design plan for negotiations of **system requirements**
2) As a means of **structuring discussions with clients**, **developers**, and **managers**

- Tool for **complexity management**
- **Hides details** and allows the designers to focus on the key system abstractions

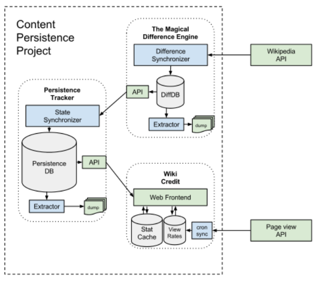
Source: https://pixabay.com/en/workplace-team-business-meeting-1245776/ by Free-Photos |CC0 Creative Commons

Architectural Design

Notations

- System architectures often use simple **block diagrams**
- Each **box** in the diagram represents a **component**
- **Boxes within boxes** indicate **decompositions**
- **Arrows** represent **data** of control signals

Note: This is usually a high-level picture of the structure, which people can easily understand. It is also often the only architectural documentation that exists.



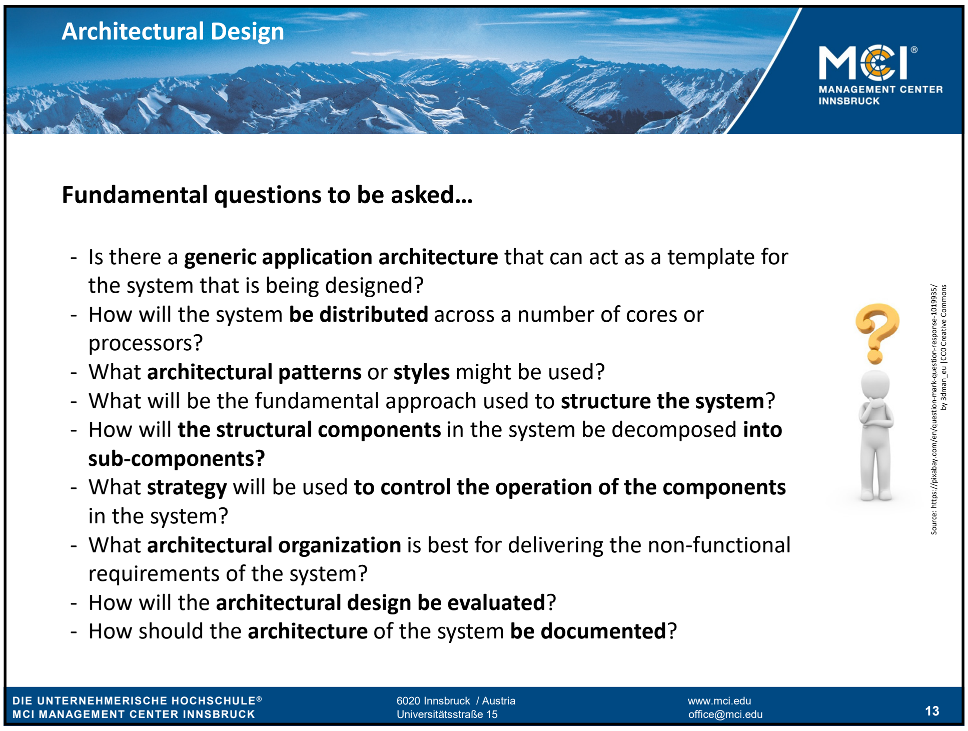
Source: https://commons.wikimedia.org/wiki/File:Content_persistence.system_architecture.diagram.svg
by EpochFail |CC0 Creative Commons

Architectural Design

Architectural Design Decisions

- **Creative process** where you design a system organization that will satisfy **functional and non-functional requirements**
- Depends on the **type of system being developed**, **the background and experience of the system architect**, and the **specific requirements of the system**
- During the design process system architects make a **number of decisions that affect the system** and its development process

Source: https://pixabay.com/en/office-home-glasses-workspace-820390/
by fancycrave1 |CC0 Creative Commons



Architectural Design

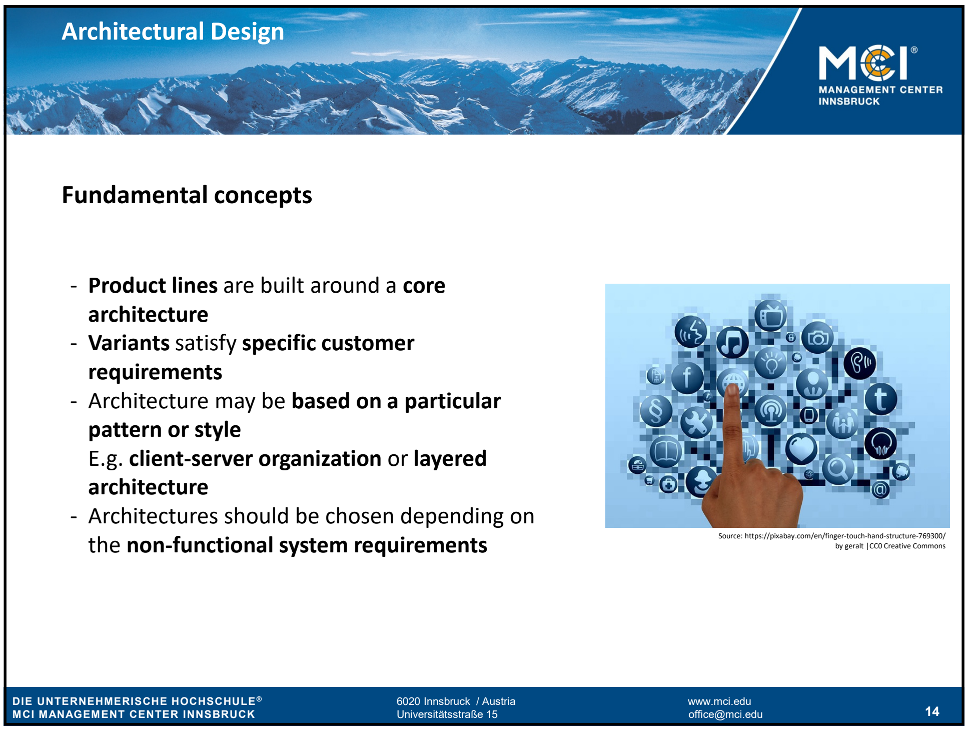
MCI® MANAGEMENT CENTER INNSBRUCK

Fundamental questions to be asked…

- Is there a **generic application architecture** that can act as a template for the system that is being designed?
- How will the system **be distributed** across a number of cores or processors?
- What **architectural patterns** or **styles** might be used?
- What will be the fundamental approach used to **structure the system**?
- How will **the structural components** in the system be decomposed **into sub-components?**
- What **strategy** will be used **to control the operation of the components** in the system?
- What **architectural organization** is best for delivering the non-functional requirements of the system?
- How will the **architectural design be evaluated**?
- How should the **architecture** of the system **be documented**?

Source: https://pixabay.com/en/question-mark-question-response-1019935/ by 3dman_eu |CC0 Creative Commons

DIE UNTERNEHMERISCHE HOCHSCHULE®
MCI MANAGEMENT CENTER INNSBRUCK
6020 Innsbruck / Austria
Universitätsstraße 15
www.mci.edu
office@mci.edu
13



Architectural Design

MCI® MANAGEMENT CENTER INNSBRUCK

Fundamental concepts

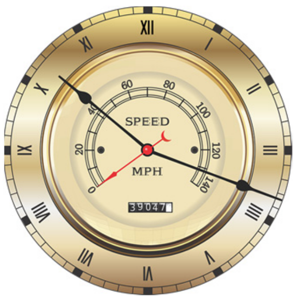
- **Product lines** are built around a **core architecture**
- **Variants** satisfy **specific customer requirements**
- Architecture may be **based on a particular pattern or style**
  E.g. **client-server organization** or **layered architecture**
- Architectures should be chosen depending on the **non-functional system requirements**

Source: https://pixabay.com/en/finger-touch-hand-structure-769300/ by geralt |CC0 Creative Commons

DIE UNTERNEHMERISCHE HOCHSCHULE®
MCI MANAGEMENT CENTER INNSBRUCK
6020 Innsbruck / Austria
Universitätsstraße 15
www.mci.edu
office@mci.edu
14

Non-functional Requirements and Software Architecture

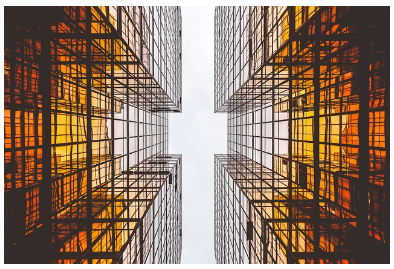
- **Performance**
  Deploy critical components on the same computer; larger components with local communication; multi-core processors; etc.
- **Security**
  Protect critical structure through layers;
- **Safety**
  Put safety-related operations together;
- **Availability**
  Integrate redundancy;
- **Maintainability**
  Use fine-grained self-contained components that may readily be changed;



Source: https://pixabay.com/en/speedometer-vintage-clock-3578085/
by peteyp8 |CC0 Creative Commons

Architectural Views (Krutchen, 1995)

- **Logical view**
  Key abstractions in the system as objects
- **Process view**
  Shows the interacting processes of the system at runtime. Useful for making judgments about non-functional system characteristics e.g. performance
- **Development view**
  Who develops which component
- **Physical view**
  System hardware and software components and how they are distributed



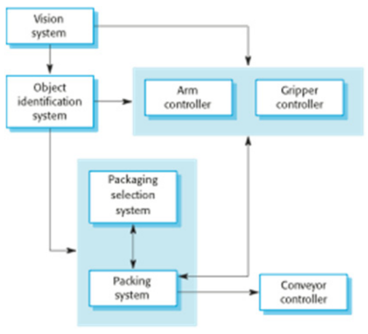
Source: https://pixabay.com/en/architecture-skyscraper-urban-city-768432/
by Free-Photos |CC0 Creative Commons

Architectural Design

MCI® MANAGEMENT CENTER INNSBRUCK

Hofmeister et al., 2000

- Use a Conceptual view as a basis for **decomposing high-level requirements into more detailed specifications**
  e.g. the packing robot from the beginning of this class
- Conceptual views are almost always **developed during the design process**
- They are used to **support decision making**
- **Communicate** the essence of a system to different **stakeholders**



Source: Sommerville, I. (2011). Software Engineering, 9th Ed. Boston, MA, USA: Pearson Education Inc.

Architectural Design

MCI® MANAGEMENT CENTER INNSBRUCK

Architectural Patterns

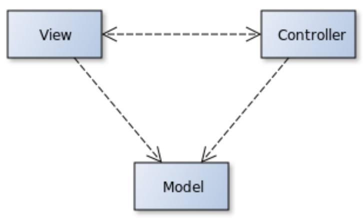
- Way of **presenting, sharing, and reusing knowledge** about software systems
- Origin **Alexander et al., 1977**
- In Software Engineering Gamma et al., 1995
- **Stylized, abstract description** of good practice
- An architectural pattern describes **a system organization that has been successful in previous systems**
- It should include **strengths** and **weaknesses**

Source: https://pixabay.com/en/stones-wall-background-quarry-stone-770264/
by meineresterampe |CC0 Creative Commons

Architectural Design

Example: Model-View-Controller pattern

- Basis for interaction management in many **web-based systems**
- Pattern description includes **name, brief description, example** of a system where the pattern is used
- Graphical models present the **architecture from different views**



Source: https://de.m.wikipedia.org/wiki/Datei:Model_View_Controller.svg by MovGP0 |CC0 Creative Commons

Architectural Design

Example: Model-View-Controller pattern

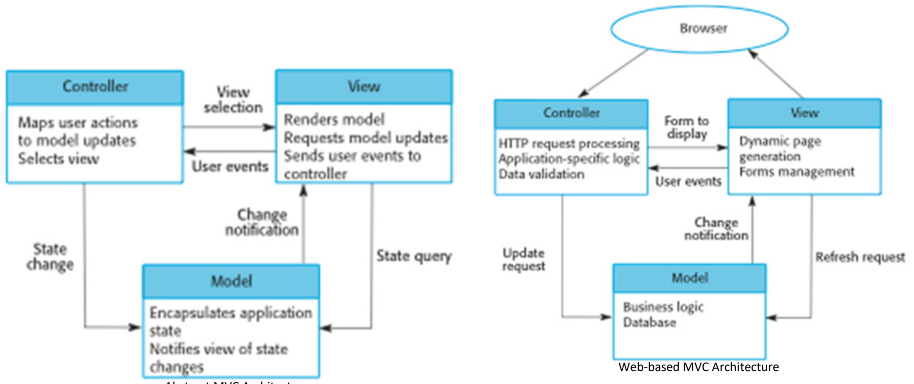
Name	MVC (Model-View-Controller)
Description	Separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines and manages how the data is presented to the user. The Controller component manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model. See Figure 6.3.
Example	Figure 6.4 shows the architecture of a web-based application system organized using the MVC pattern.
When used	Used when there are multiple ways to view and interact with data. Also used when the future requirements for interaction and presentation of data are unknown.
Advantages	Allows the data to change independently of its representation and vice versa. Supports presentation of the same data in different ways with changes made in one representation shown in all of them.
Disadvantages	Can involve additional code and code complexity when the data model and interactions are simple.

Source: Sommerville, I. (2011). Software Engineering, 9th Ed. Boston, MA, USA: Pearson Education Inc.

Architectural Design

MANAGEMENT CENTER INNSBRUCK

Example: Model-View-Controller pattern



Abstract MVC Architecture

Web-based MVC Architecture

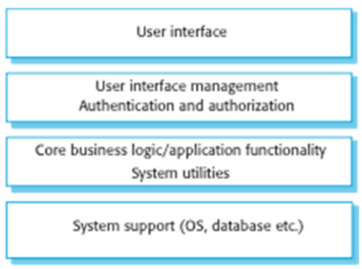
Source: Sommerville, I. (2011). Software Engineering, 9th Ed. Boston, MA, USA: Pearson Education Inc.

Architectural Design

MANAGEMENT CENTER INNSBRUCK

Example: Layered Architecture

- **Separation and independence** are fundamental as they allow changes to be localized
- **MVC** allows for **changing views without changing the underlying data model**
- Similar the layered architecture which allows for **replacing one layer by another**
- If a layer **changes** only the **adjacent layers are affected**
- Supports **multi-platform implementation**


User interface
User interface management
Authentication and authorization
Core business logic/application functionality
System utilities
System support (OS, database etc.)

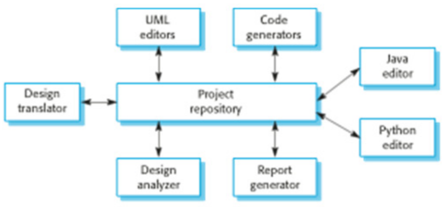
Source: Sommerville, I. (2011). Software Engineering, 9th Ed. Boston, MA, USA: Pearson Education Inc.

Architectural Design

MCI® MANAGEMENT CENTER INNSBRUCK

Example: Repository Architecture

- Describes how a **set of interacting components** can **share data**
- Suited for applications in which **data is generated by one component and used by another**
- Tools are **organized around a repository**
- There is **no need to transmit data** explicitly from one component to another
- **Alternative** approach from **AI systems**: **'blackboard'** model



Source: Sommerville, I. (2011). Software Engineering, 9th Ed. Boston, MA, USA: Pearson Education Inc.

Architectural Design

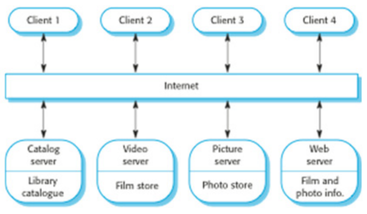
MCI® MANAGEMENT CENTER INNSBRUCK

Example: Client-Server Pattern

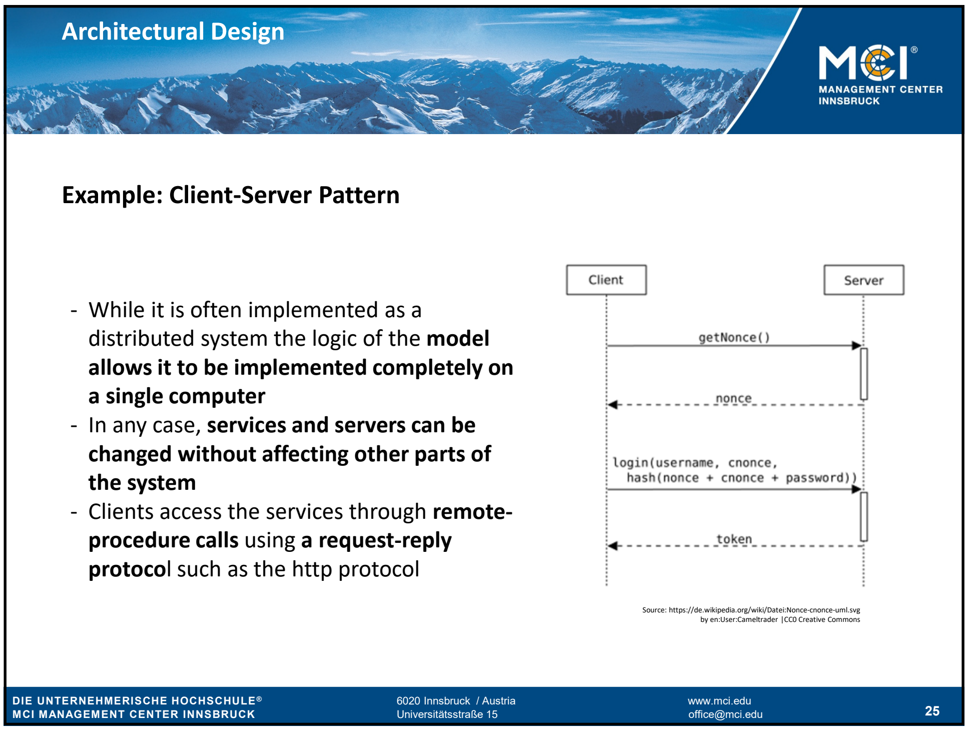
- Used for the run-time organization of distributed systems
- Organized as a set of services and associated servers, and clients accessing these services

The major components are

- Servers: e.g. print server, file server, compile server;
- Clients: call services
- Network: connects servers and clients often implemented as distributed systems



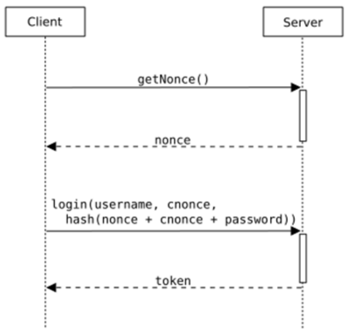
Source: Sommerville, I. (2011). Software Engineering, 9th Ed. Boston, MA, USA: Pearson Education Inc.



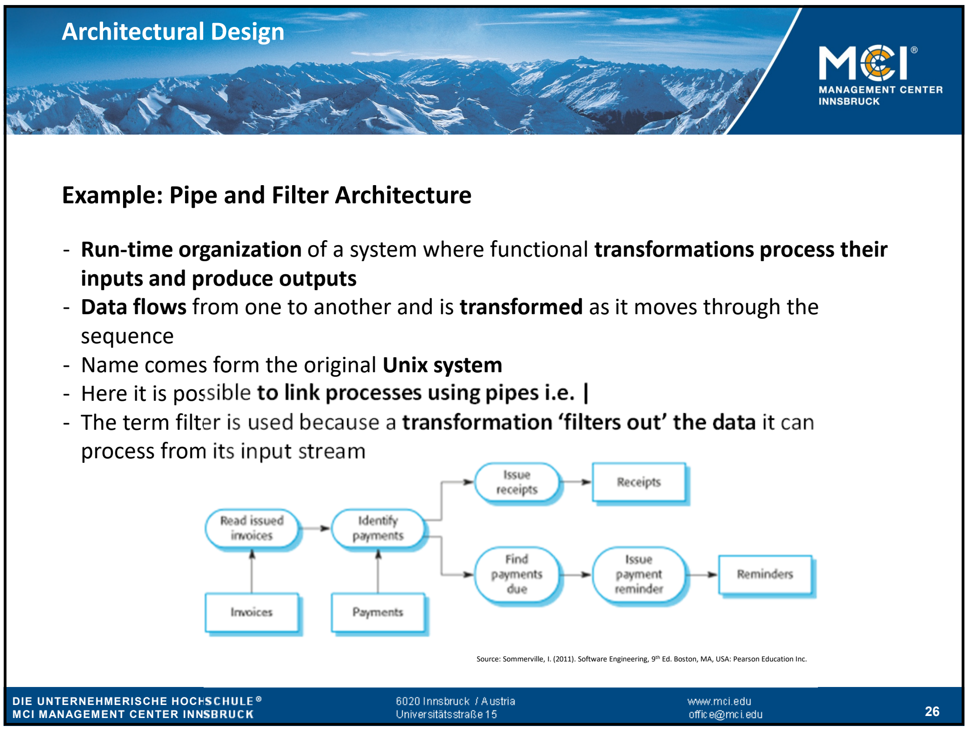
Architectual Design

Example: Client-Server Pattern

- While it is often implemented as a distributed system the logic of the **model allows it to be implemented completely on a single computer**
- In any case, **services and servers can be changed without affecting other parts of the system**
- Clients access the services through **remote-procedure calls** using **a request-reply protoco**l such as the http protocol



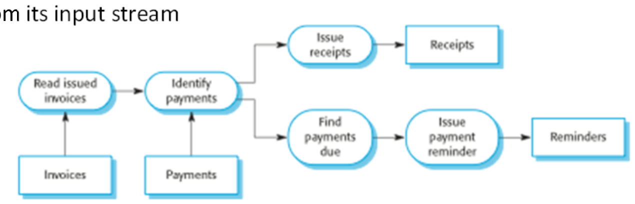
Source: https://de.wikipedia.org/wiki/Datei:Nonce-cnonce-uml.svg by en:User:Cameltrader |CC0 Creative Commons



Architectual Design

Example: Pipe and Filter Architecture

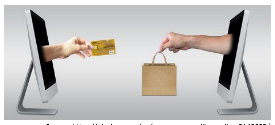
- **Run-time organization** of a system where functional **transformations process their inputs and produce outputs**
- **Data flows** from one to another and is **transformed** as it moves through the sequence
- Name comes form the original **Unix system**
- Here it is possible **to link processes using pipes i.e. |**
- The term filter is used because a **transformation 'filters out' the data** it can process from its input stream



Source: Sommerville, I. (2011). Software Engineering, 9th Ed. Boston, MA, USA: Pearson Education Inc.

Architectural Design

Application Architectures


Source: https://pixabay.com/en/ecommerce-selling-online-2140603/ by Mediamodifier |CC0 Creative Commons

Transaction Processing Systems
- Database systems using transactions
  e.g. ATM transaction
- Interactive systems in which users make asynchronous requests
- Transaction manager ensures that transaction is properly completed

Information Systems
- Type of transaction processing system
- Allows controlled access to a large base of information
- Increasingly web-based and accessed through a web browser

Language Processing Systems
- Translate a natural or artificial language into another representation of that language
  e.g. a compiler translates an artificial programming language into machine code; natural language processing systems may translate from French to German

Summary

What you should have taken away from this class:

- A software architecture is a **description of how a software system is organized**. Properties of a system such as **performance, security, and availability** are influenced by the architecture used.
- Architectural design decisions include **decisions on the type of application**, the **distribution of the system**, the **architectural styles** to be used, and the **ways in which the architecture should be documented** and evaluated.
- Architectures may be documented from **several different perspectives or views**. Possible views include a **conceptual view, a logical view, a process view, a development view, and a physical view.**
- **Architectural patterns** are a **means of reusing knowledge about generic system architectures**. They describe the architecture, explain when it may be used, and discuss its **advantages and disadvantages**.
- Commonly used architectural patterns include **Model-View-Controller, Layered Architecture, Repository, Client-server, and Pipe and Filter.**
- Transaction processing systems are **interactive systems** that allow information in a database to be remotely accessed and modified by a number of users. Information systems and resource management systems are examples of transaction processing systems.
- **Language processing** systems are used to **translate texts from one language into another** and to carry out the instructions specified in the input language. They include a translator and an abstract machine that executes the generated language.

References

- Alexander, C.; Ishikawa, S.; & Silversrtein, M. (1977). A Pattern Language: Towns, Buildings, Construction. Oxford, UK: Oxford University Press.
- Bass, L., Clements, P., & Kazman, R. (2003). *Software Architecture in Practice 2nd Ed.* Boston, MA, USA: Addison-Wesley.
- Berczuk, S. P., & Appleton, B. (2002). *Software Configuration Management Patterns: Effective Teamwork, Practical Integration.* Boston, MA, USA: Addison-Wesley.
- Bosch, G. (2000). *Design and Use of Software Architecture.* Harlow, UK: Addison-Wesley.
- Coplien, J. H., & Harrison, N. B. (2004). *Organizational Patterns of Agile Software Development.* Englewood Cliffs, NJ, USA: Prentice Hall.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* Reading, MA, USA: Addison-Wesley.
- Garlan, D., & Shaw, M. (1993). An Introduction to Software Architecture. *Advances in Software Engineering and Knowledge Engineering 1* , 1-39.
- Hofmeister, C., Nord, R., & Soni, D. (2000). *Applied Software Architecture.* Boston, MA, USA: Addison-Wesley.
- Hofmeister, C., Nord, R., & Soni, D. (2000). *Applied Software Architecture.* Boston, MA, USA: Addison-Wesley.
- Krutchen, P. (1995). The 4+1 View Model of Software Architecture. *IEEE Software 12(6)* , 42-50.
- Lewis, P. M., Bernstein, A. J., & Kifer, M. (2003). *Databases and Transaction Processing: An Application-oriented Approach.* Boston, MA, USA: Addison-Wesley.
- Martin, D., & Sommerville, I. (2004). Patterns of Interaction: Linking ethnomethodology and design. *ACM Transactions on Computer-Human Interaction 11(1)* , 59-89.

Thank you for your attention!

mentoring the motivated.