Université de Bouira
Faculté des sciences
Département de Informatique

Module IHM

3ème Année Licence

IHM

Design Methods

BENAISSI Sellami s.benaissi@univ-bouira.dz

2023-2024

PART

1

Software Design Methods

Software Design Methods

There are many different software engineering design models, each with their own strengths and weaknesses. They can be classified into categories as in the table:

Type	Description	Example
Data-Driven Design	Models the flow of data through a system, including the system's inputs, outputs, and processes.	Data Flow Diagram (**DFD**), Entity-Relationship Diagram (**ERD**)
Object-Oriented Design	Models the system using objects and their interactions.	Unified Modeling Language (**UML**)
Model-Driven Design	Uses models to generate code and other artifacts.	Business Process Model and Notation (**BPMN**), **UML**
Event-Driven Design	Models the system using events.	Event-Driven Architecture (**EDA**), Message Queue Telemetry Transport (**MQTT**)
Agile Methods	Focus on delivering working software early and often.	**Scrum**, Dynamic Systems Development Method (**DSDM**)
Specific Methods	Models the system using a specific methodology.	**Merise, Waterfall model, V-model, Spiral model**

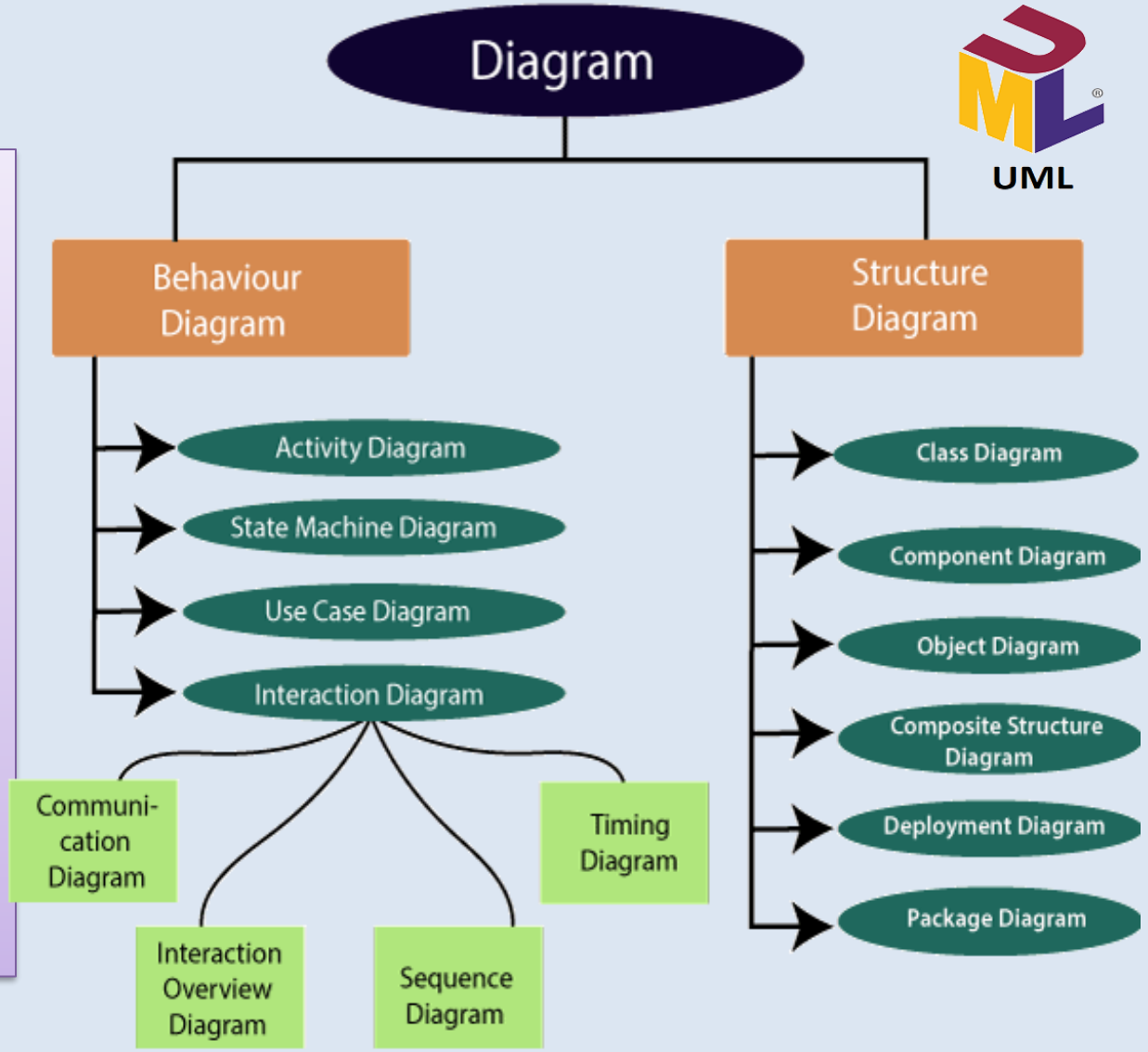
Software Design Methods

Type	Advantages	Disadvantages
Data-Driven Design	Easy to understand and use, well-suited for systems with complex data requirements.	Can be time-consuming to create and maintain, may not be suitable for systems with complex interactions between components.
Object-Oriented Design	Provides a modular and reusable approach to design, well-suited for systems with complex interactions between components.	Can be difficult to learn and apply, may not be suitable for systems with simple data requirements.
Model-Driven Design	Can automate much of the design and development process, well-suited for systems with complex or changing requirements.	Can be difficult to set up and use, may not be suitable for systems with simple requirements.
Event-Driven Design	Provides a scalable and flexible approach to design, well-suited for systems that need to respond to real-time events.	Can be difficult to design and implement, may not be suitable for systems with simple requirements.
Agile Methods	Well-suited for systems with changing requirements, provides a collaborative approach to development.	Can be difficult to manage and control, may not be suitable for systems with complex requirements.
Specific Methods	Well-suited for systems with well-defined requirements and a predictable development process.	Can be inflexible and difficult to adapt to changing requirements, may not be suitable for systems with complex or uncertain requirements.

Software Design Methods

Unified Modeling Language (UML)

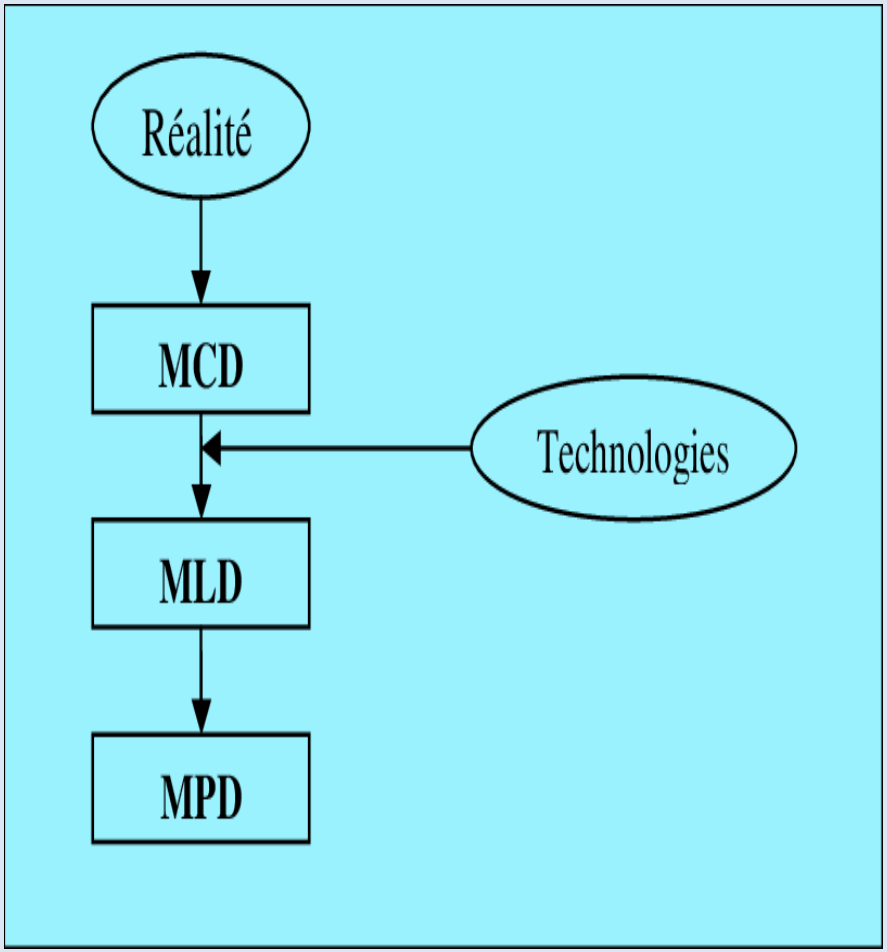
UML is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software system. It is a popular model-driven design method that can be used to create models of the system's structure, behavior, and interactions. UML models can be used to generate code and other artifacts, which can help to improve the quality and consistency of the system.



Software Design Methods

Merise

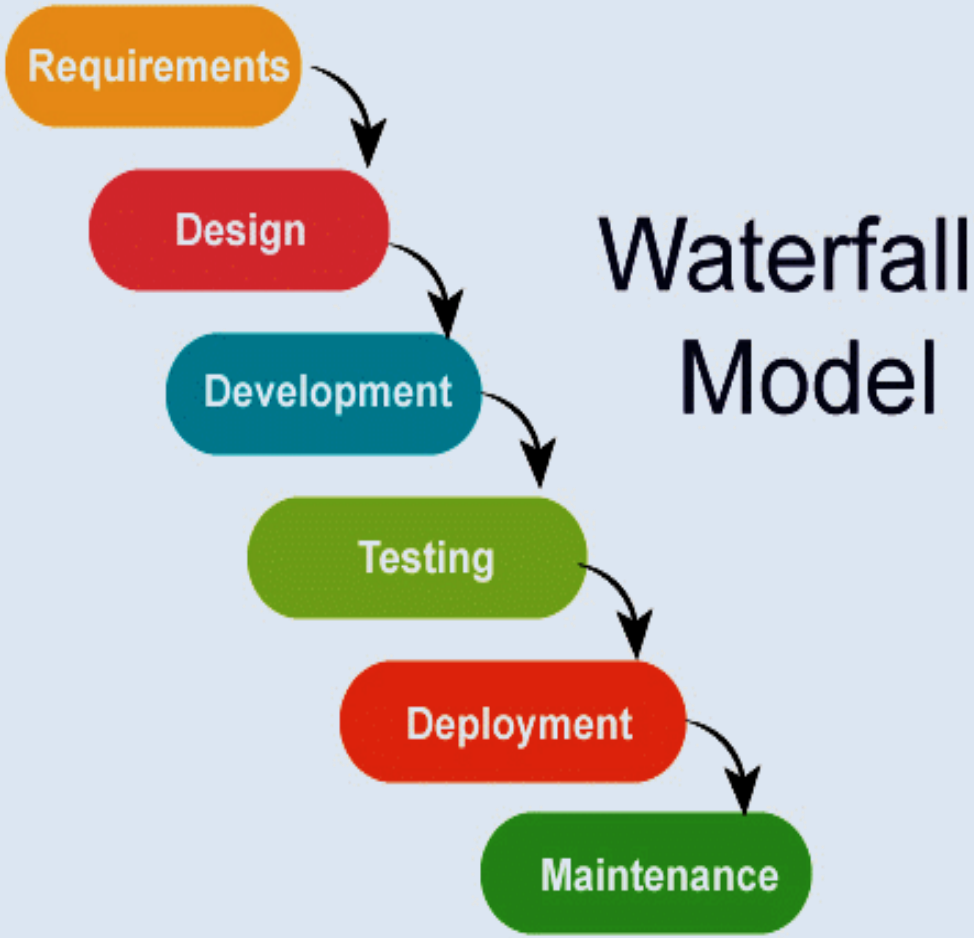
Merise is a structured and entity-relationship modeling method used in software engineering. It focuses on analyzing, designing, and implementing information systems by defining data structures, relationships, and business processes.



Software Design Methods

Waterfall

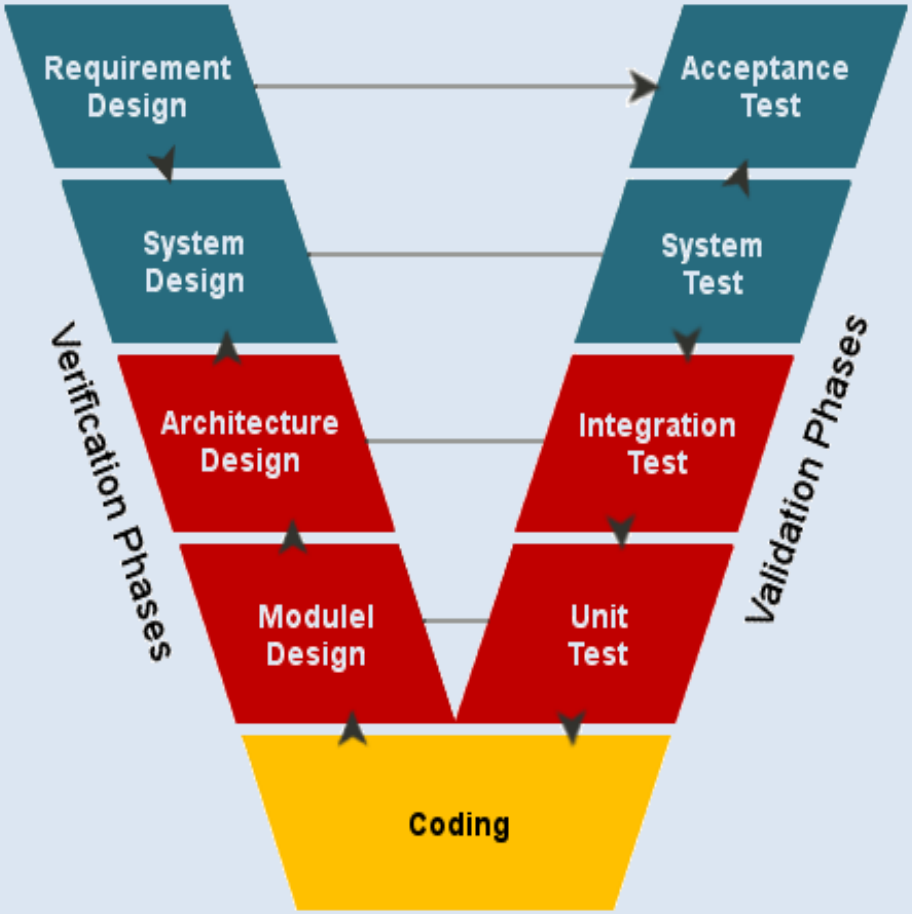
The waterfall model is a sequential software development methodology that is well-suited for projects with well-defined requirements. The waterfall model can help to improve the quality and predictability of the development process.



Software Design Methods

V-model

The V-model is a software development methodology that combines the waterfall model with a validation and verification phase at the end of each development phase. The V-model can help to improve the quality and reliability of the system.



Software Design Methods

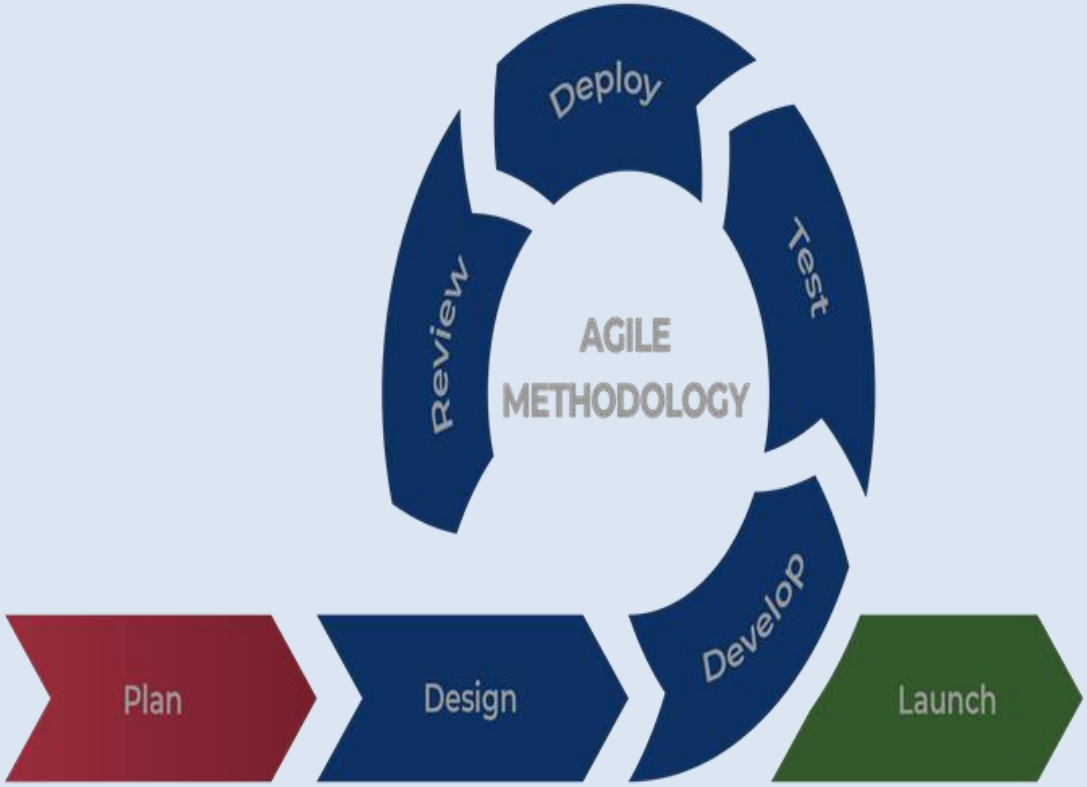
Spiral Model

The Spiral Model is an iterative and risk-driven software development approach. It combines elements from the waterfall model and the iterative model, focusing on managing risks through regular iterations, prototyping, and risk analysis. it is better suited for projects with changing requirements.

Software Design Methods

Agile Methods

Agile methods are iterative and incremental approaches to software development that prioritize collaboration, flexibility, and responsiveness to change. Examples of Agile methods include Scrum, which emphasizes iterative development and self-organizing teams, and DSDM (Dynamic Systems Development Method), which focuses on delivering business value quickly while maintaining quality.



Software Design Methods

Gaps and Shortcomings Related to the Interactive System:

- Limited User Involvement.
- System-centered (functional guarantee) to the detriment of users.
- Late Evaluation

It is necessary to define analysis and design methods in which aspects related to human-machine interfaces are more explicitly considered.

- ✓ Design carried out by a multidisciplinary team.

Why an HMI design method?

- ❖ Reduction of risks and maintenance costs
- ❖ Reduction of budget/training time
- ❖ Attractiveness of the application, productivity gain
- ❖ Reuse and improvement of basic components

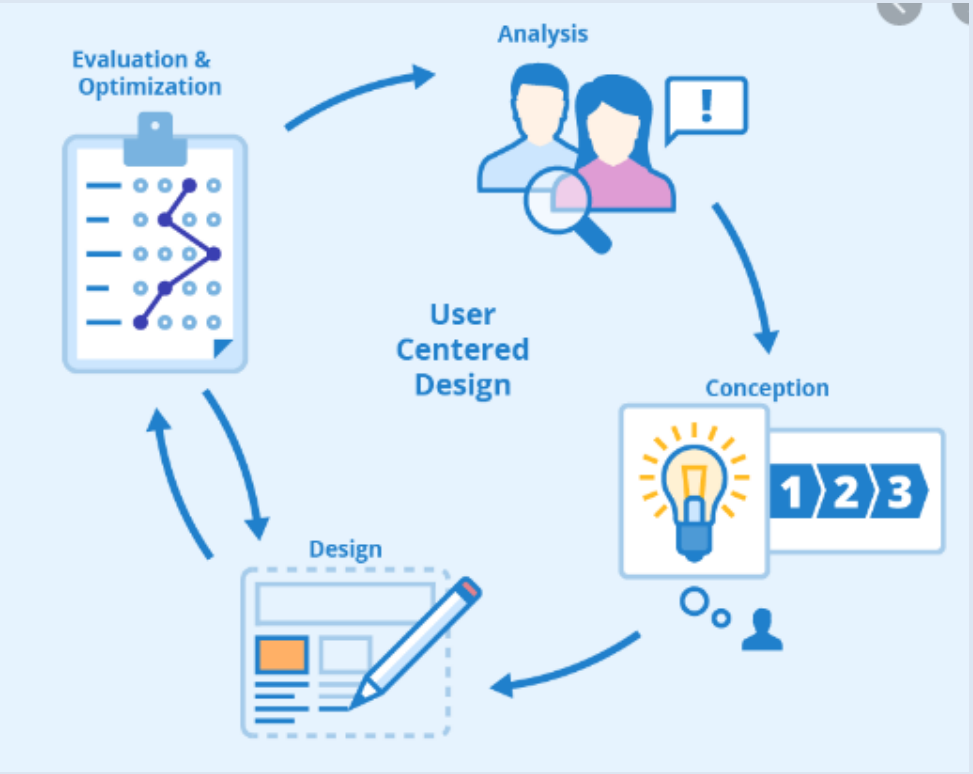
PART

2

User-Centered Design (UCD)

User-centered design (UCD)

User-centered design (UCD) is an iterative design process in which designers focus on the users and their needs in each phase of the design process. In UCD, design teams involve users throughout the design process via a variety of research and design techniques, to create highly usable and accessible products for them.



User-centered design (UCD)

User-centered design (UCD)

In user-centered design, designers use a mixture of investigative methods and tools (e.g., surveys and interviews) and generative ones (e.g., brainstorming) to develop an understanding of user needs.

The term was coined in the 1970s. Later, cognitive science and user experience expert Don Norman adopted the term in his extensive work on improving what people experience in their use of items.

And the term rose in prominence thanks to works such as User Centered System Design: "*New Perspectives on Human-Computer Interaction*" (which Norman co-authored with Stephen W. Draper) and Norman's "*The Design of Everyday Things*" (originally titled "*The Psychology of Everyday Things*").

User-centered design (UCD)

User-centered design (UCD)

User-Centered Design is a design process that focuses on users and their needs at every stage of the design and development process.

This goal is achieved by including users in all stages of design and development and focusing on the entire user experience.

This goal is achieved by identifying users' requirements and analyzing their behaviours, needs, expectations and preferences.

User-centered design is applied in many fields such as website design and mobile applications.

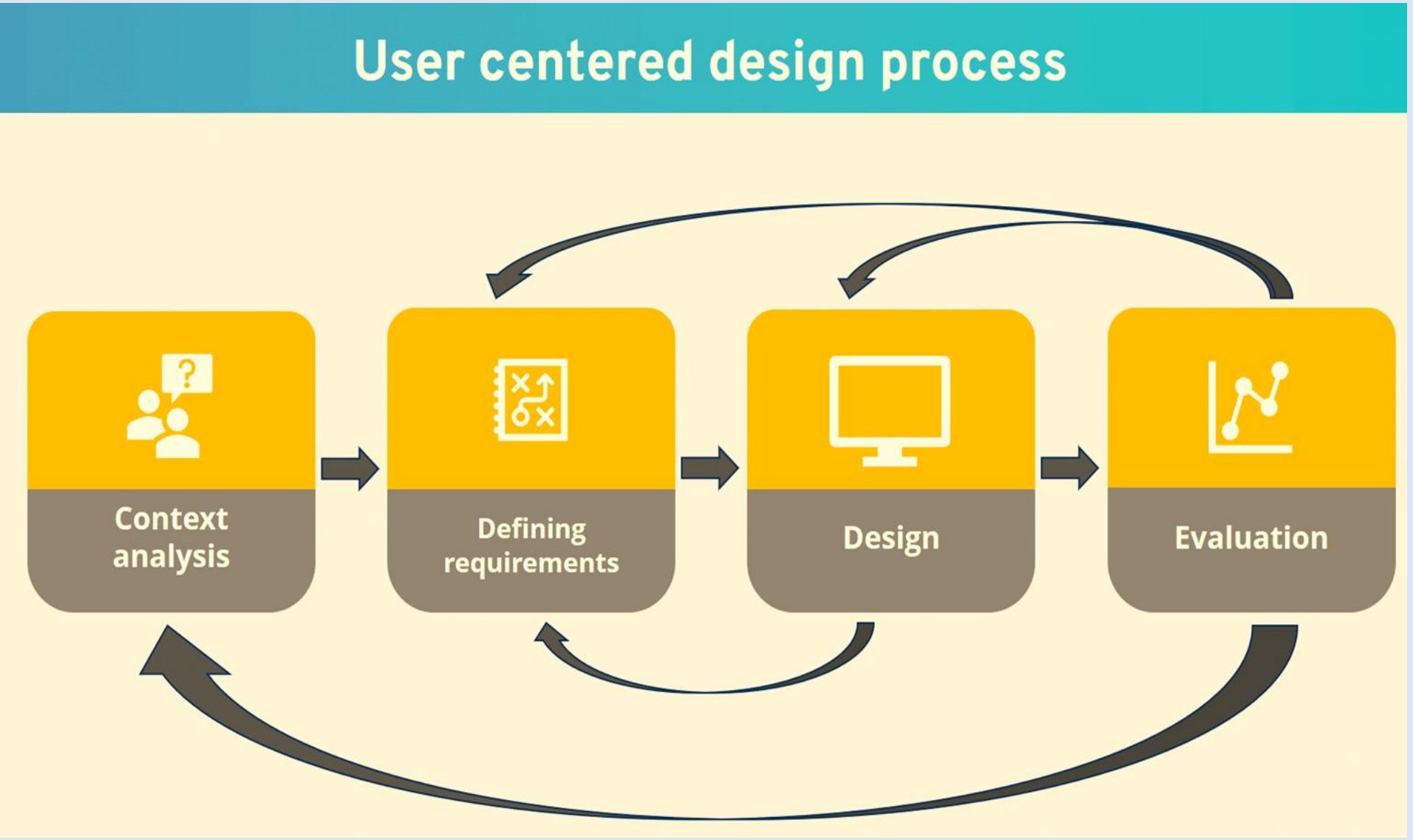
User-centered design (UCD)

Here are some principles that focus on user-centered design:

- Design based on a clear understanding of users, tasks and environments.
- Identifying users' requirements and analyzing their behaviours, needs, expectations and preferences.
- Including users in all stages of design and development.
- Apply evaluation frequently and continuously.
- Continuously improve the user experience.

User-centered design can be used in the design and development of interactive graphical user interfaces, and is used as a basic model for many other models in this field. This goal is achieved by providing order and organization of the user interface and defining the functions and tools necessary to achieve the specified goals.

User-centered design (UCD)



User-centered design (UCD)

Phase: Context analysis

User ? | Task ? | Context ?

- Specify the expectations and needs of users, know their tasks, and the context.
- **Answer questions such as**:
  - ✓ Who are my users?
  - ✓ What are their problems?
  - ✓ What are their characteristics and abilities (perception, cognition, motor)?
  - ✓ What is the context of use? etc.
- **Tools**:
  - ✓ Interview/Survey,
  - ✓ Persona,
  - ✓ Questionnaire,
  - ✓ Focus Group,
  - ✓ Observation,
  - ✓ Field study,
  - ✓ Documentation,etc.

User-centered design (UCD)

Phase: Context analysis

Personas

A personas are imaginary models that represent the users of your app or website. These personas are used to represent different groups of users and define their needs, expectations, and behaviors. User personas help improve the user experience and improve performance, effectiveness and satisfaction when using the application or website. Personas are based on research and data about real users and are used to understand user needs, behaviors, motivations, and goals.

Here are some important points about user personas:

- User personas are used to **represent different groups of users** and define their **needs**, **expectations**, and **behaviors**.
- User personas are created based on **users' actual** and **reference data**.
- User personas help improve the user **experience** and improve **performance**, **effectiveness** and **satisfaction** when using the application or website.
- Many templates available online can be used to create user personas.
- User personas must be **compatible** with the **target** user group and contain **detailed** and **accurate** information about the users.

User-centered design (UCD)

Phase: Context analysis

Personas

Buyer Persona Slide Template

PERSONA'S NAME

Position

Info 1: edit text

Info 2: edit text

Info 3: edit text

ABOUT

You can edit this text.

Text can be edited. You can edit this text. Text can be edited.

NEEDS

You can edit this text.

Text can be edited. You can edit this text.

PAIN POINTS

You can edit this text.

Text can be edited. You can edit this text.

PERSONALITY

Trait 1

Trait 2

Trait 3

GOALS

Goal 1

Goal 2

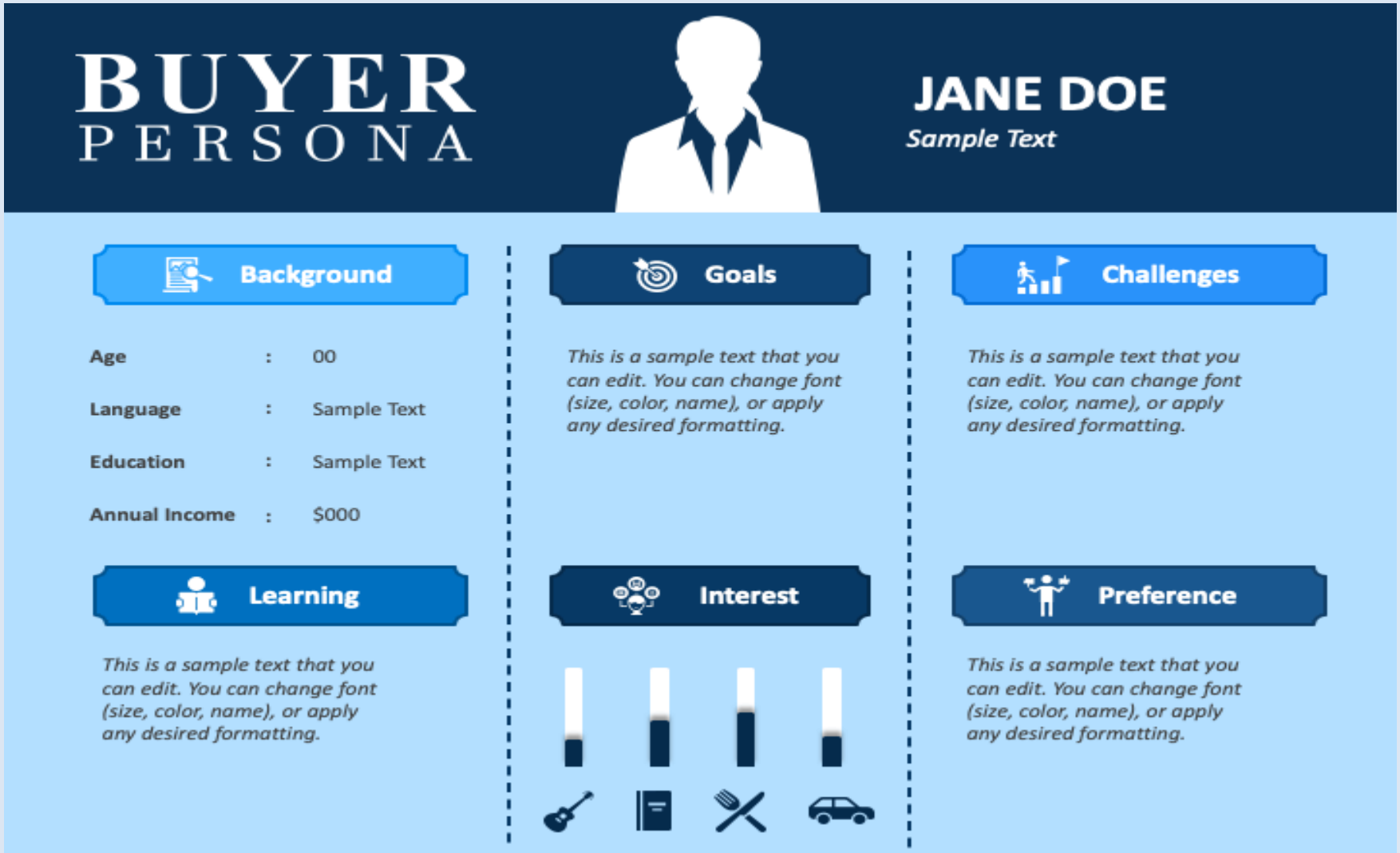
Goal 3

TAG 1 TAG 2 TAG 3 TAG 4 TAG 5 TAG 6 TAG 7 TAG 8

User-centered design (UCD)

Phase: Context analysis

Personas



ISO 9241-210

ISO 9241-210 is an international standard that provides guidelines for the design of human-centered interactive systems. It was first published in 2010 and has been revised twice since then. The latest revision, ISO 9241-210:2019, was published in 2019.

The key aspects covered in ISO 9241-210:

1. **Human-Centered Design:** The standard emphasizes the importance of involving end users throughout the design process. It promotes a user-centered approach that considers user characteristics, tasks, and needs, as well as the specific context of use.

2. **User Interface Design Principles:** ISO 9241-210 provides principles and guidelines for designing user interfaces that are intuitive, efficient, and effective. These principles cover aspects such as clarity, simplicity, consistency, feedback, error prevention, and user control.

ISO 9241-210

3. User Interface Components: The standard discusses various user interface components, such as menus, dialog boxes, forms, icons, and controls. It provides recommendations for their design, layout, and interaction to enhance usability and user experience.

4. Accessibility and Inclusivity: ISO 9241-210 emphasizes the importance of designing interfaces that are accessible to users with diverse abilities and needs. It encourages inclusive design practices to ensure that interactive systems can be used by a wide range of users.

5. Usability Evaluation: The standard also includes guidance on evaluating the usability of user interfaces. It provides methods and criteria for usability testing, expert evaluations, and user feedback collection to assess and improve the usability of interactive systems.

ISO 9241-210

ISO 9241-210 is intended to be used by designers, developers, and evaluators of interactive systems, as well as organizations concerned with human-centered design and usability. It provides a framework for designing user interfaces that prioritize user needs, enhance user satisfaction, and contribute to the overall usability of interactive systems.

It's worth noting that **ISO 9241-210** is just one part of the **ISO 9241** series, which covers various **aspects** of human-centered design for interactive systems. Other parts of the **ISO 9241** series address topics such as ergonomic requirements, visual display requirements, and dialogue principles.

PART

3

Model – View – Controller (MVC)

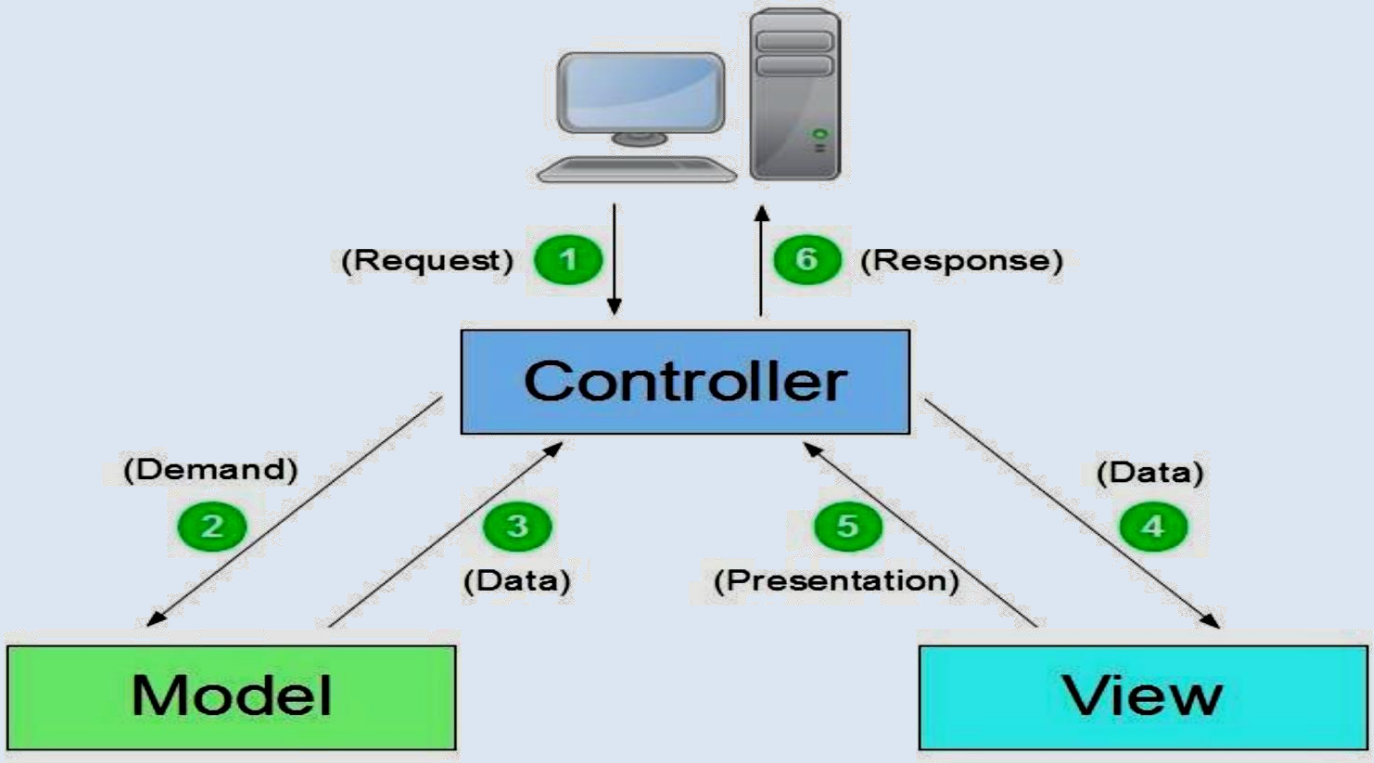
Model – View – Controller (MVC)

- **The MVC (Model-View-Controller)**

- Created by Trygve Reenskaug during his visit to the Palo Alto Research Center (PARC) in 1978.
- A software design and architecture pattern intended for graphical interfaces.
- Very popular for web applications

Model – View – Controller (MVC)

- **The MVC (Model-View-Controller)** pattern is a widely used architectural pattern in software development that separates the concerns of an application into three interconnected components: the model, the view, and the controller. It provides a structured approach to designing and organizing code for user interfaces and applications.



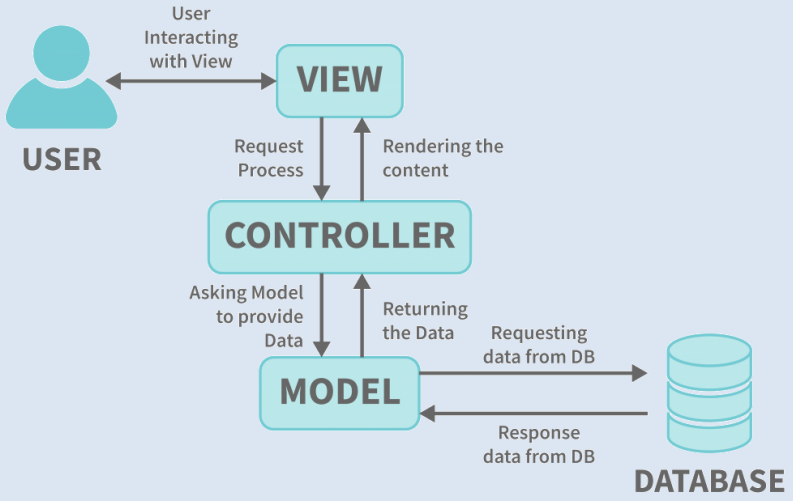
Model – View – Controller (MVC)

Components of the MVC pattern:

1. Model:

The model represents the **data** used by the application. It can be stored in a database, a file, or in memory. The model encapsulates the data, defines the rules and operations on that data, and notifies the view and controller of any changes.

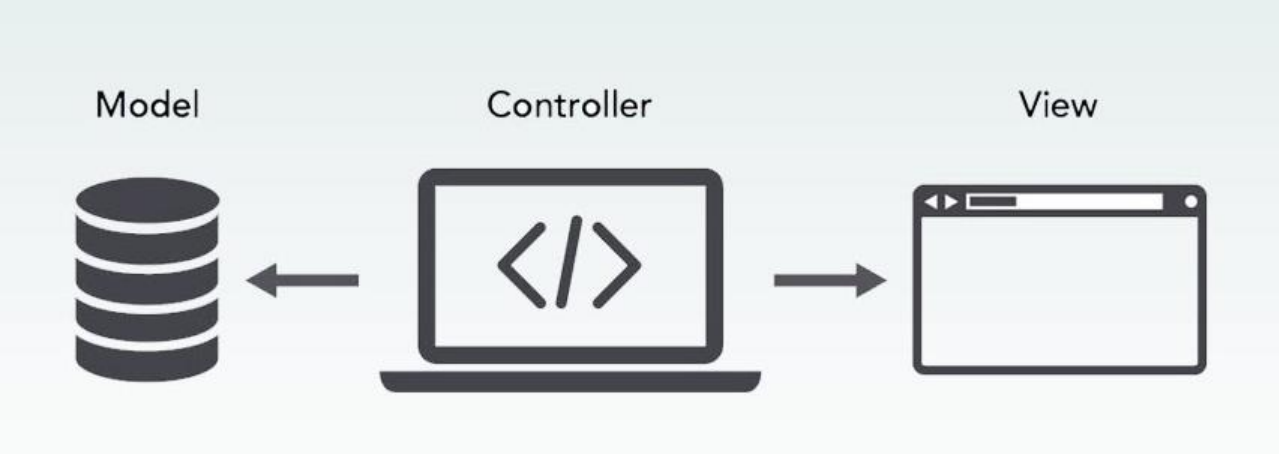
it also contains the **business logic** of the application, such as the rules for validating data and performing calculations. It does not have direct knowledge of either the view or the controller.



Components of the MVC pattern:

2. View:

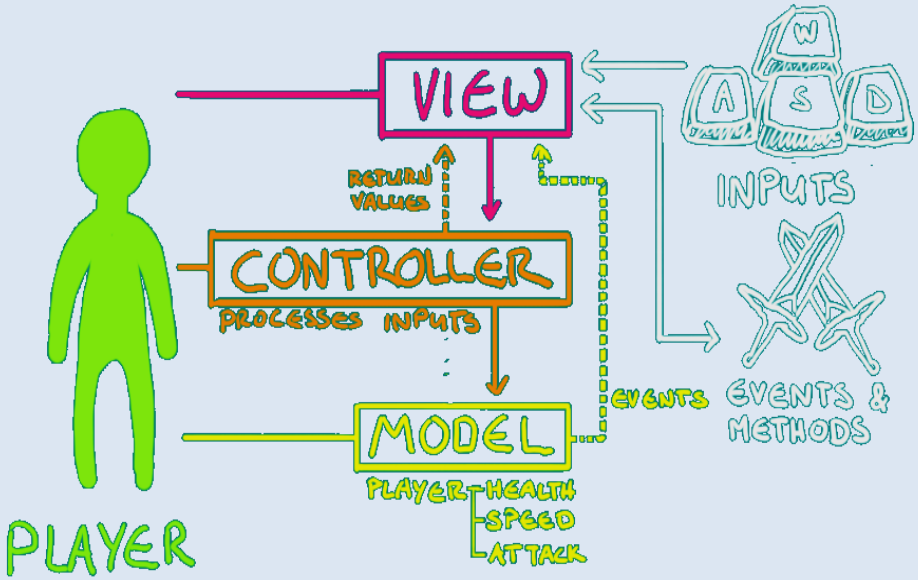
The view is the **user interface**. The view is responsible for **presenting** the data from the model to the user and for capturing user interactions. It displays the visual representation of the data and provides an interface for users to interact with the application. The view does not contain application logic and should remain decoupled from the model and controller.



Components of the MVC pattern:

3. Controller:

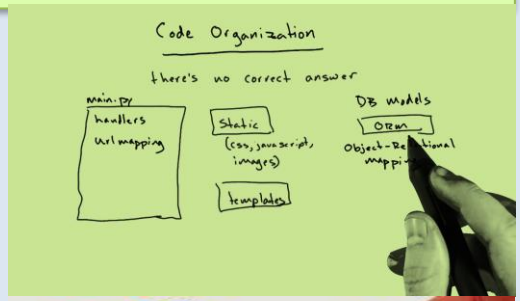
The controller acts as an intermediary between the model and the view. The controller handles user interactions and updates the model accordingly. It also updates the view to reflect the changes to the model. The controller is the glue that holds the MVC pattern together. It is responsible for coordinating the interactions between the model and the view.



Model – View – Controller (MVC)

Benefits of using the MVC pattern include:

- **Code organization:** The separation of concerns leads to better code organization and maintainability.



- **Code reusability:** The modular nature of MVC allows components to be reused in different contexts.

- **Testability:** The separation of concerns makes it easier to test individual components in isolation.

- **User interface flexibility:** The use of views allows for different representations of the same data, enabling flexibility in user interface design.

Model – View – Controller (MVC)

Popular MVC frameworks:

- Django (Python)
- Rails (Ruby)
- Laravel (PHP)
- ASP.NET MVC (C#)

Examples of popular websites and applications that use the MVC pattern:

- Web applications: Twitter, Facebook, Google, Amazon, Wikipedia
- Desktop applications: Microsoft Office, Adobe Photoshop, IntelliJ IDEA
- Mobile applications: iOS apps, Android apps