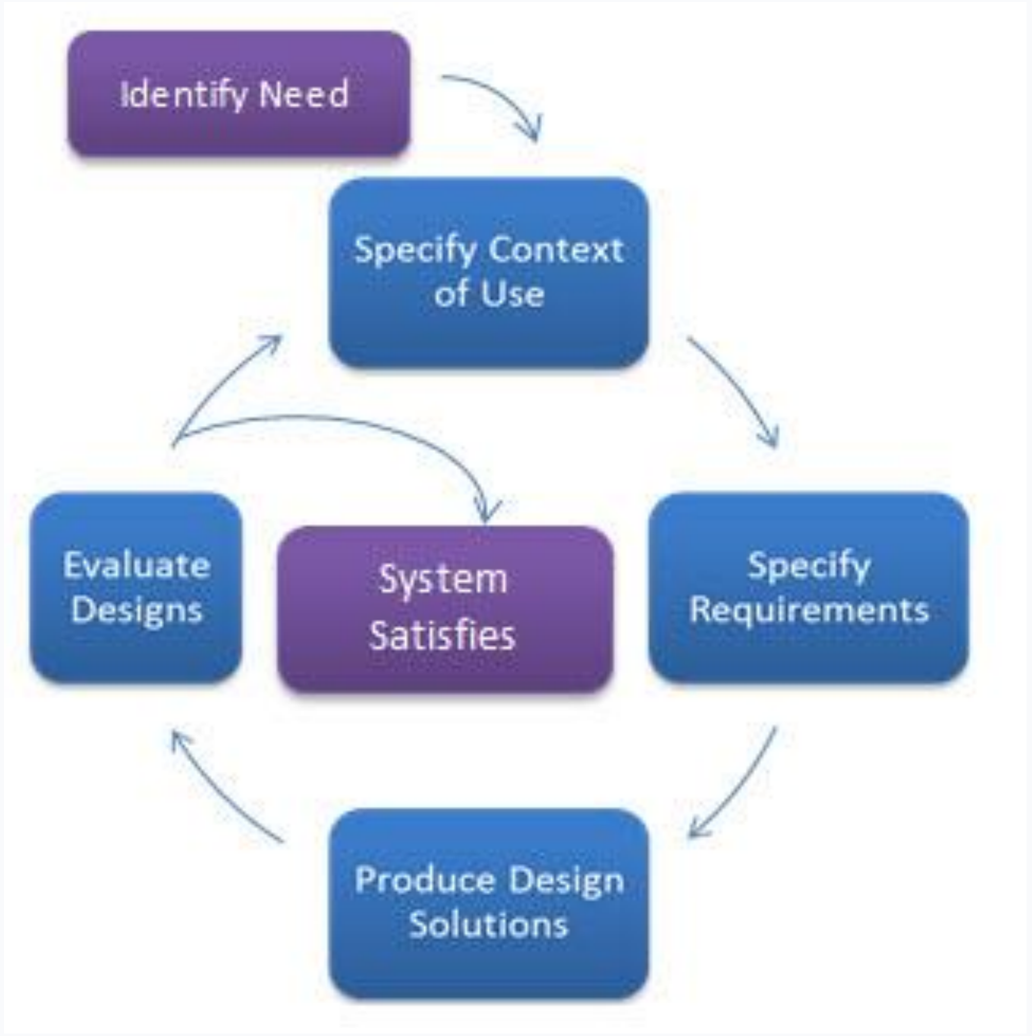
COMP 2511
Object Oriented Design & Programming

What is User Centred Design ?



Specify the context of use: Identify the people who will use the product (Persona), what they will use it for, and under what conditions they will use it. (Scenario)

Specify requirements : Identify any business requirements or user goals that must be met for the product to be successful. (Use-Cases)

Create design solutions: This part of the process may be done in stages, building from a rough concept to a complete design.

Evaluate designs: Evaluation - ideally through usability testing with actual users - is as integral as quality testing is to good software development.

User Centred Design

- Agile Development philosophy provides value to customers, but in order for our software to be truly successful in the eyes of its biggest critics, we must adopt a more user-centered approach.
- UCD can be applied to the design of anything that has a user—from mobile phones to kitchens.
- The era of **feature-centric development** is coming to an end. Consumers are beginning to realize that more features do not always mean a better product. Quality of experience is far more likely to be a product differentiator than product features
- UCD provides a way to engineer these quality experiences.

Nielsen Norman Group

Jakob's Ten Usability Heuristics

1 Visibility *of* System Status

Designs should *keep users informed* about what is going on, through appropriate, timely feedback.

Interactive mall maps have to show people where they currently are, to help them understand where to go next.

2 Match between System and the Real World

The design should speak the users' language. Use words, phrases, and concepts *familiar to the user,* rather than internal jargon.

Users can quickly understand which stovetop control maps to each heating element.

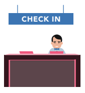
3 User Control *and* Freedom

Users often perform actions by mistake. They *need a clearly marked "emergency exit"* to leave the unwanted action.

Just like physical spaces, digital spaces need quick "emergency" exits too.

4 Consistency *and* Standards

Users should not have to wonder whether different words, situations, or actions mean the same thing. *Follow platform conventions.*



Check-in counters are usually located at the front of hotels, which meets expectations.

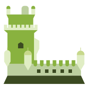
5 Error Prevention

Good error messages are important, but the best designs carefully *prevent problems* from occurring in the first place.

Guard rails on curvy mountain roads prevent drivers from falling off cliffs.

6 Recognition Rather Than Recall

***Minimize the user's memory load* by making elements, actions, and options visible. Avoid making users remember information.**



People are likely to correctly answer "Is Lisbon the capital of Portugal?".

7 Flexibility *and* Efficiency of Use

Shortcuts — hidden from novice users — may *speed up the interaction* for the expert user.

Regular routes are listed on maps, but locals with more knowledge of the area can take shortcuts.

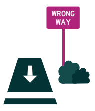
8 Aesthetic *and* Minimalist Design

Interfaces should not contain information which is irrelevant. Every extra unit of information in an interface *competes* with the relevant units of information.

A minimalist three-legged stool is still a place to sit.

9 Recognize, Diagnose, *and* Recover from Errors

Error messages should be expressed in plain language (no error codes), precisely indicate the problem, and constructively suggest a solution.



Wrong-way signs on the road remind drivers that they are heading in the wrong direction.

10 Help *and* Documentation

It's best if the design *doesn't need* any additional explanation. However, it may be necessary to provide documentation to help users complete their tasks.

Information kiosks at airports are easily recognizable and solve customers' problems in context and immediately.

JavaFX and MVC Architecture

Separation of Concerns Using MVC Architecture

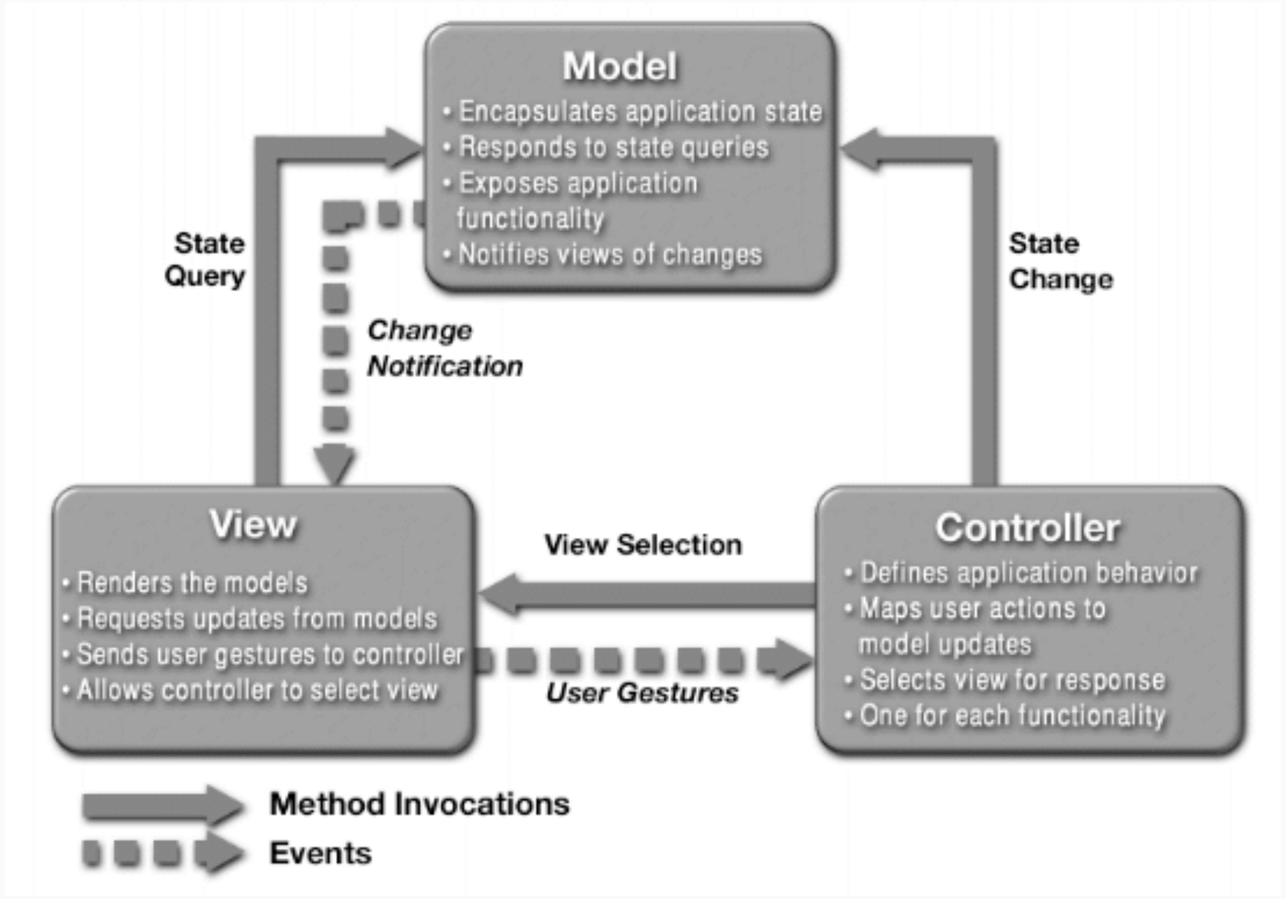

Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
Change Notification
State Change
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

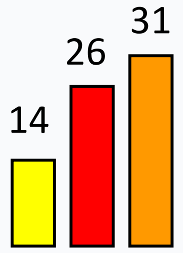
Image http://lkubaski.wordpress.com/2012/12/20/java-se-application-design-with-mvc-with-images/

View

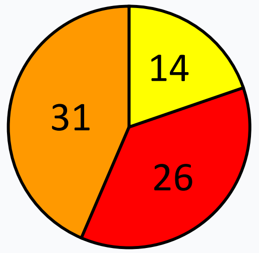
- The presentation layer that provides the interaction that the user sees (e.g. a web page).
- View component takes inputs from the user and sends actions to the controller for manipulating data.
- View is responsible for displaying the results obtained by the controller from the model component in a way that user wants them to see or a pre-determined format (e.g., HTML, XML)
- View can query the model for updates
- It is responsibility of the controller to choose a view to display data to the user.

Model: array of numbers [14, 26, 31]

➔ Different Views for the same Model:



versus



Model (Data)

- Holds all the data, state
- Responds to instructions to change of state (from the controller)
- Responds to requests for information about its state (usually from the view),
- Sends notifications of state changes to “observer” (view) (this “push” behaviour may not always happen)
- The model does NOT depend on the controller or the view

Controller

- Glue between user and processing (Model**)** and formatting (View) logic
- Accepts the user request or inputs to the application, parses them and decides which type of Model or View should be invoked
- Provides model data to the view

Benefits of MVC

- MVC inserts a controller class between the view and model and decouples the two tiers, thereby making the model and view components reusable without modification