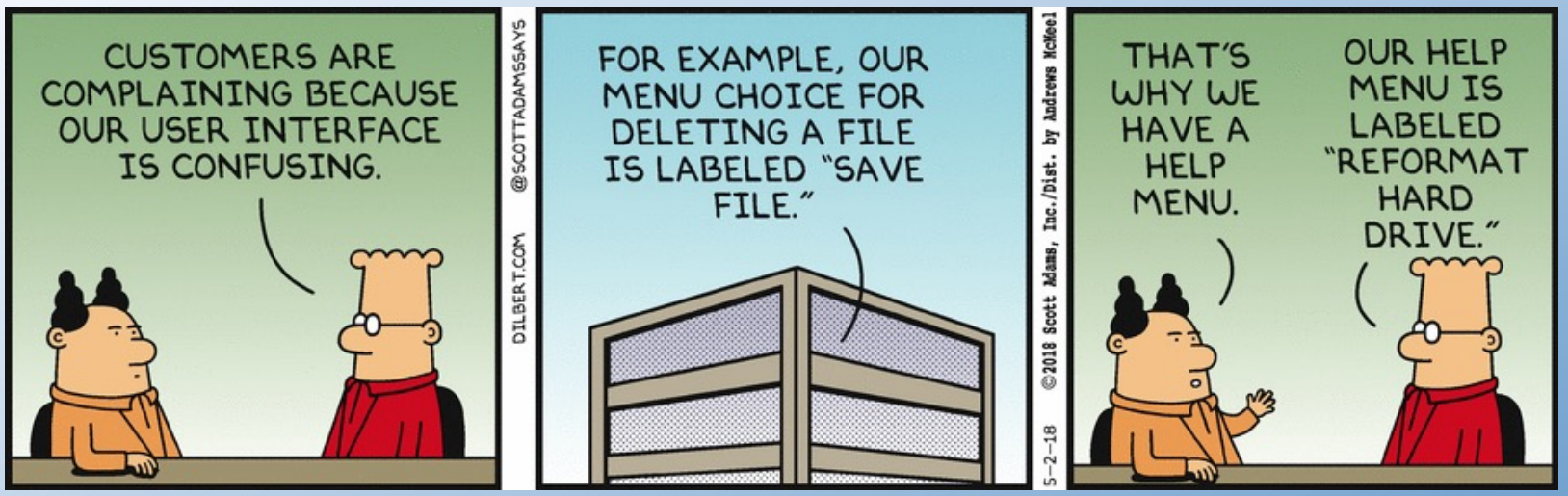


Design and the User Interface

CSCI2340: Software Engineering of Large Systems

Steven P. Reiss

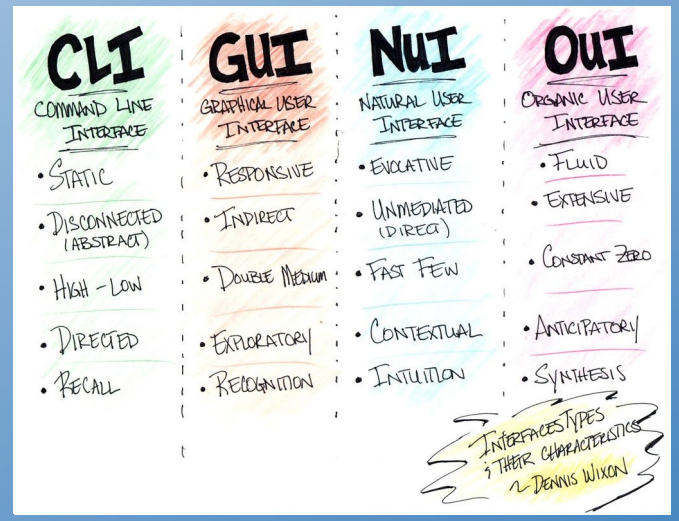
User Interfaces

- Are an essential part of modern applications
- Can make or break the application
- HCI is its own field of study
  - Understanding how people interact with computers
  - Understanding how to evaluate and assess interfaces
  - Understanding what works, what people like
  - Visual design
- Having a design background helps
  - **Computer scientists typically design poor user interfaces**

Homework Discussion

- I gave you 3 possible features to be added to bouncing balls
  - Were any of them easy with your current design?
  - Were any of them impossible with your current design?
  - What approach do you think you would take for each?
- Let's see what happened

User Interface Types



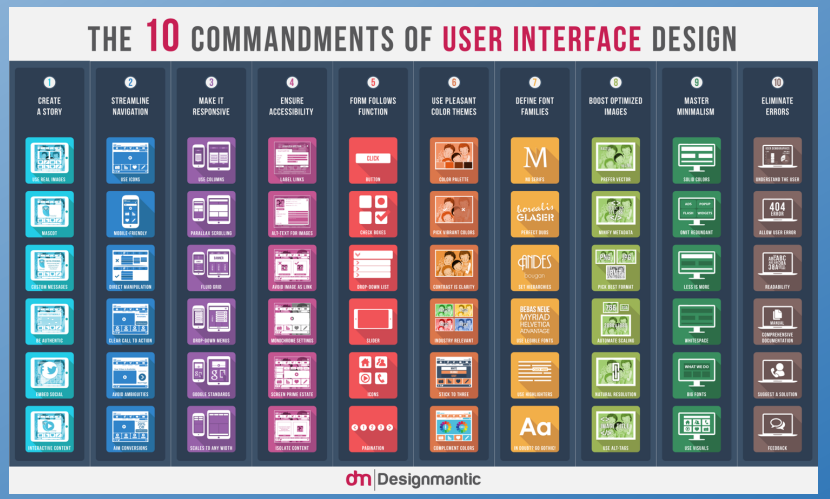
- Command-line interfaces
  - Relatively easy to deal with, but limited (git, …)
- GUIs: Workstation & phone interfaces
  - Widget-based, callback-based interaction, direct manipulation
  - AWT/Swing, JavaFX, Motif, Qt, WPF, Android, Swift, Dart, …
- Web interfaces
  - Original model: Reload a new page each time
  - Today: Update page in browser; RESTful connection to server
- Natural Language Interfaces
  - Search, ChatGPT, phone bots
- VR interfaces
  - OpenGL (WebGL), augmented reality, 3D headsets

User Interfaces

- The user interface is often critical to success
  - Difficult to get right a priori
  - Often needs a prototype
  - Often needs multiple prototypes
- The interface affects the rest of the application
  - How it works, what it needs to do,
  - What data it needs, what the user can do
  - Often at the design level
- The user interface is a risk to the system and to the design
  - We've discussed how to handle risk

Peter Morville's user experience honeycomb, which shows all the interrelated qualities of a great user experience.

User Interface and High-Level Design



- How these interact varies with the application
- Where does the UI fit into the design
  - Can be done later (treat as low-level design)
  - Can be the driving factor for the design
  - Can be in between
- It depends
  - How critical the user interface is to the application
    - How much the design of the UI affects the design of the application
    - With a completely different interface, how much of the application would persist
  - How well defined the user interface is from specifications
    - How sure are you of that interface
  - How separable is the user interface from the rest of the system
- Different approaches are used

Deferred User Interface Design

- The UI might be important, but not central to the design
  - Then the user interface is not critical at this stage
  - A variety of interfaces will probably work
  - You don't need to know the best from the start
    - Can prototype different interfaces later on
- Isolate the UI from the rest of the design
  - Determine what information might be needed by the UI
  - Determine what commands are needed by the system
  - Create a component with an interface or façade for the UI
  - Design of the UI is now a lower-level design problem
  - Don't even need to build a full UI initially
- Examples: SHORE, UPOD/Sherpa, S^6, FAIT

UI-Centric Design

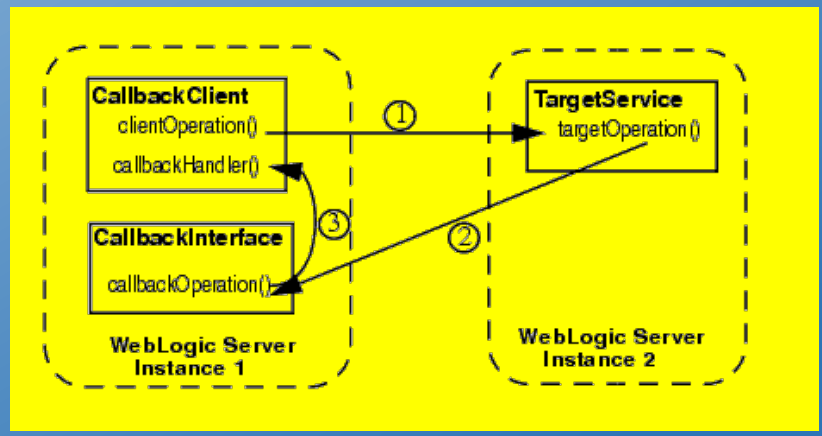
- Sometimes the UI drives the application
  - Then the user interface is more important
  - The system should exist to support the interface
- Specifications
  - Give a sketch of the user interface
  - Identify the commands and operations needed
- Build the user interface first
  - Rebuild it until you get it right
  - Fill in the rest of the system as needed
  - Even if the rest of the system is complex
- Often combined with UI-first implementation
- Examples: ROSE, DYMON

UI-Pervasive Design

- Sometimes the user interface is the application
  - Makes it more difficult to change
  - UI tightly coupled with the rest of the code
- Designing the user interface is designing the application
  - The rest of the application is secondary
  - Most system components interact directly with the UI
  - The rest of the application fits into the UI rather than vice versa
- Prototype ideas to get the overall UI structure correct
  - Then implement a real UI and fill in the details as needed
  - Still try to keep many changes local
- Examples: Code Bubbles, FREDIT

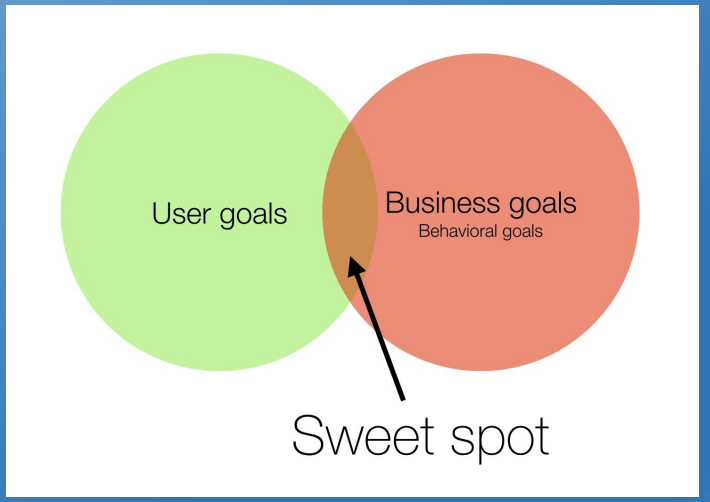
Interactive Interfaces Use Callbacks

- A callback is a routine/object registered with a component
  - Invoked when something changes
  - Example: UI buttons, drag and drop, file monitors
  - Example: Debugging events in Code Bubbles
- Useful for monitoring data structures that can change
  - UI registers callbacks with from the non-UI part of the application
  - SHORE: monitoring state of HO trains, sensors, etc.
    - In the UI to update the display when anything changes
    - Internally to continually checking safety
    - Updating the planning process when that happens
- Widely used in interactive UIs
  - Register callbacks with the view to handle user interactions
  - Done with widgets, HTML/JavaScript, …
  - Most modern applications are reactive: driven by these callbacks
- Need to add appropriate callback interfaces and methods to the design

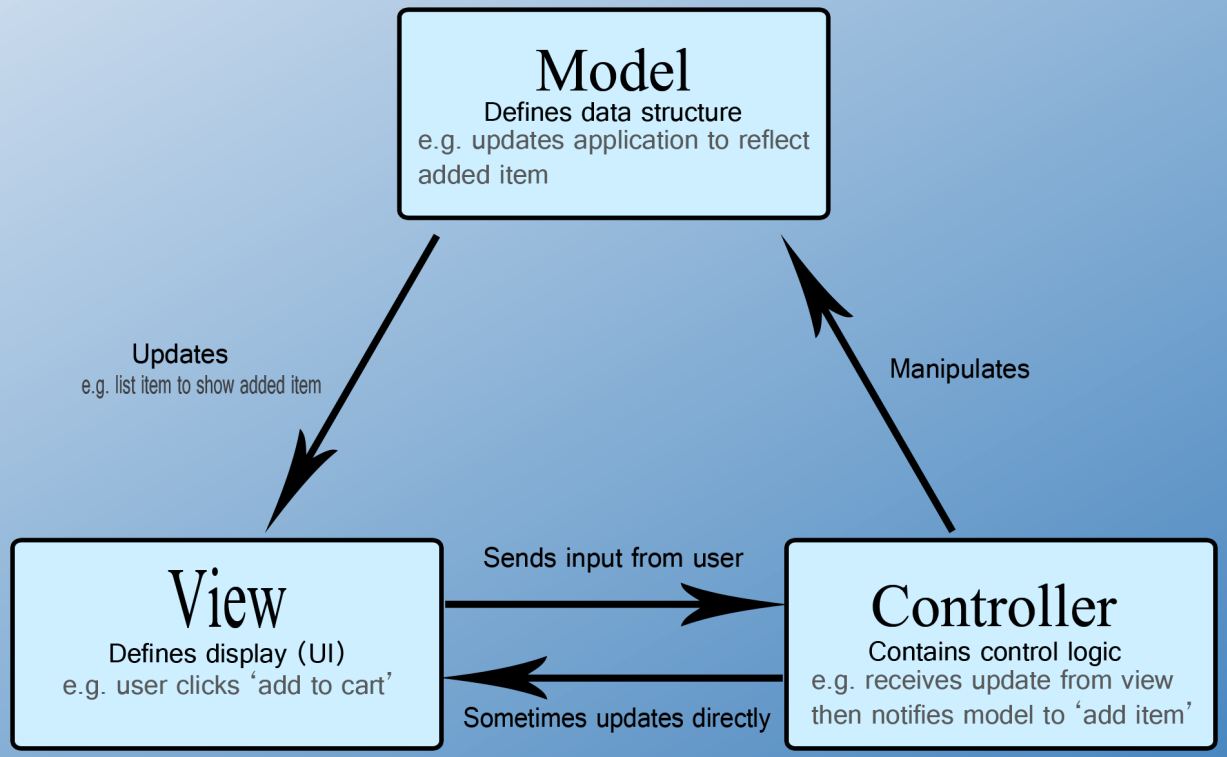


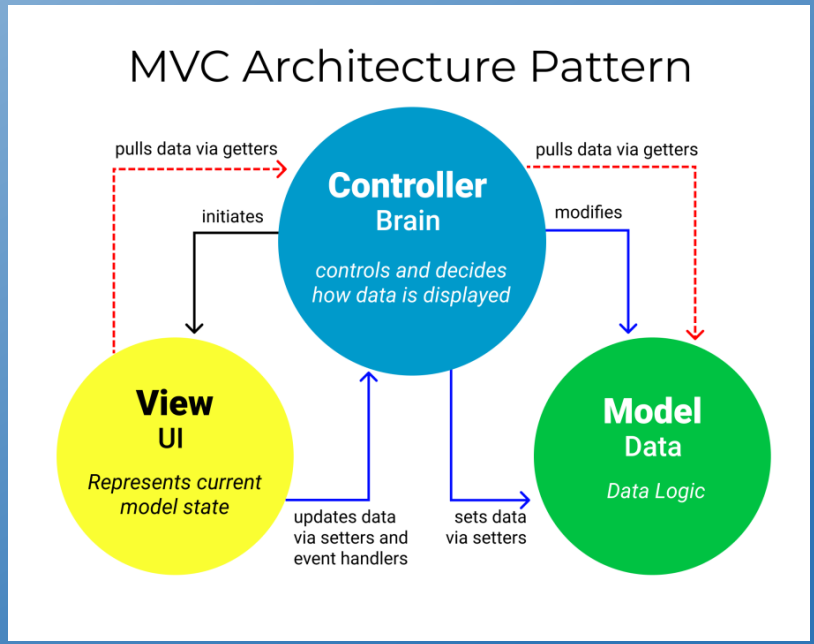
User Interface Design Goals

- Separate the user interface from the rest of the application
  - UI will change (a lot)
  - Want to minimize the changes on the rest of the system
  - At least isolate what might change
  - Even if the user interface is the application
    - Freezing a bubble to rearrange the screen
- Allow experimentation with the user interface
  - Prototyping in whole or in part
  - Before, during and after implementation
- Allow for future expansion and changes
  - In the system and the user interface
- Different strategies for UI integration
  - But all can be viewed as Model-View-Controller (MVC)

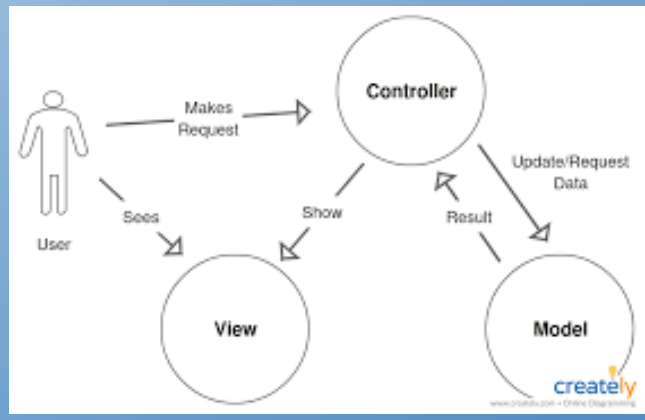


Model-View-Controller Architecture

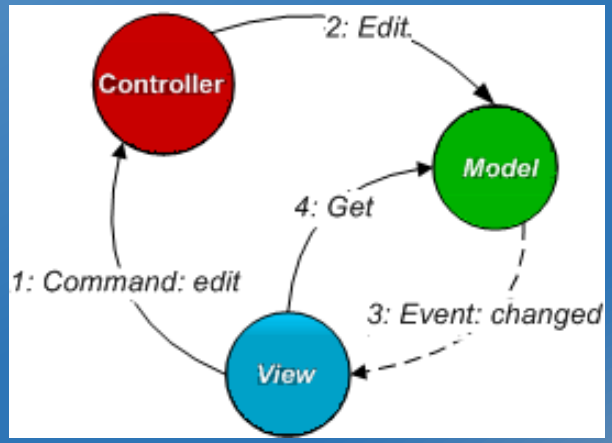




MVC Can Mean Almost Anything

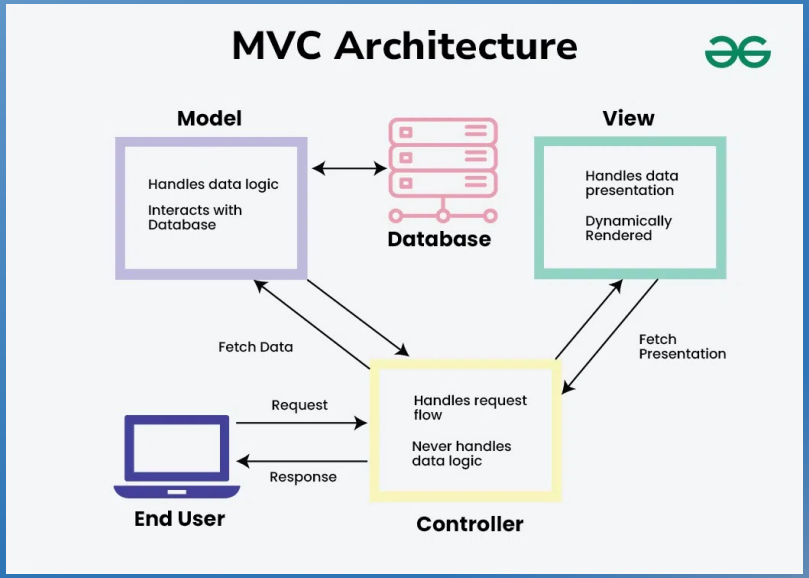


- Many ways to implement MVC
  - Many very different implementations call themselves MVC designs
  - Varying degrees of independence of model, view, and controller
  - Varying degrees of communication between model, view, and controller
  - Varying size and complexity of view, model, and controller
- MVC means different things to different people
- The components are often entangled
  - Model includes Controller
  - View includes Controller
  - Model includes View

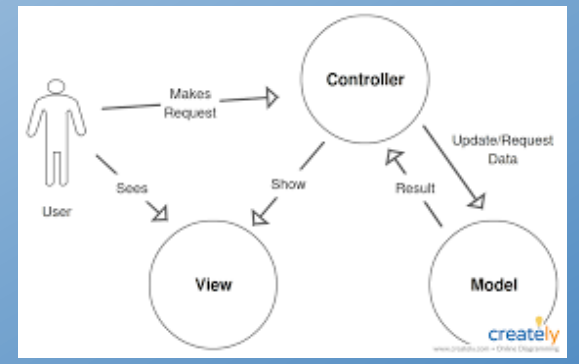


MVC Issues: Size of Components

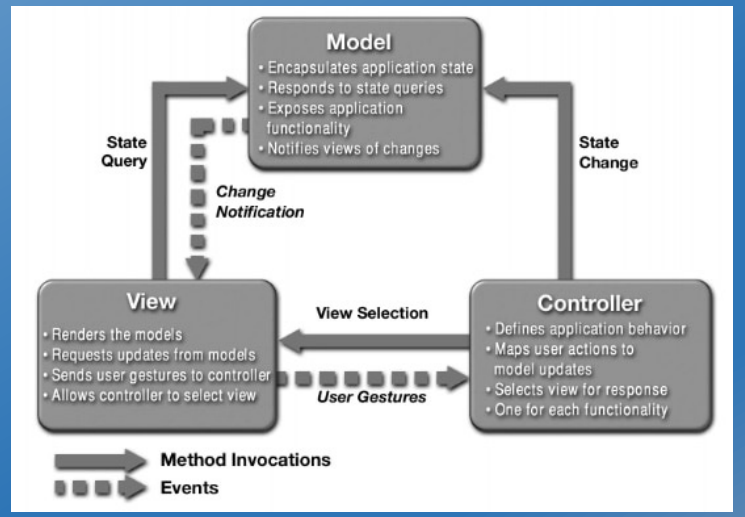
- **Model** can be a complex component or can be trivial
  - Complex: SEEDE, SHERPA, DYMON
  - Trivial: ROSE (all commands, very little retained)
  - In Between: SHORE, TWITTER DISPLAY, FREDIT, Code Bubbles Editor
- **Controller** can be a complex component or combined
  - Complex: TWITTER DISPLAY
  - Part of view: RESTful interfaces to microservices
  - Part of Model: Code Bubbles
- **View** depends on user interface
  - Simple set of commands (git)
  - Complex interactions (TWITTER DISPLAY)
  - Complex graphics (DYMON)
  - VR model of the world



MVC Issues : Who Talks to Whom

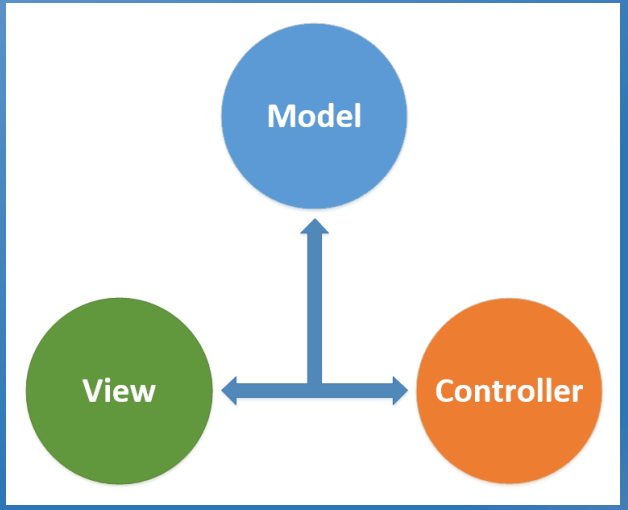


- Communication patterns in MVC also vary
- Can **model** communicate directly with **view** and vice versa
  - **View** can generate model update commands directly
    - RESTful applications changing the database directly
  - **Model** generates view updates directly
    - Callbacks, live data
- Or must all communication go thru the **controller**
  - **View** generates commands
    - Controller translates these into model updates
  - **Model** tells the controller what has changed
    - Controller translates these into view updates



MVC Issues: Representation

- The view needs to understand the model in the abstract
  - View shouldn't know the (implementation) details of the model
  - But needs to know what the contents are enough to display them
  - Also needs to know enough to query or update
  - Don't want to duplicate the model implementation in the view
    - Especially with a separate front end (web/mobile applications)
    - Don't want the view to depend on model implementation
    - But this is sometimes done
- Representing the view in the abstract can be tricky
  - What commands are allowed or implied?
  - What does direct manipulation mean?
  - How much does the controller need to know
  - Does the controller need a view of the model?

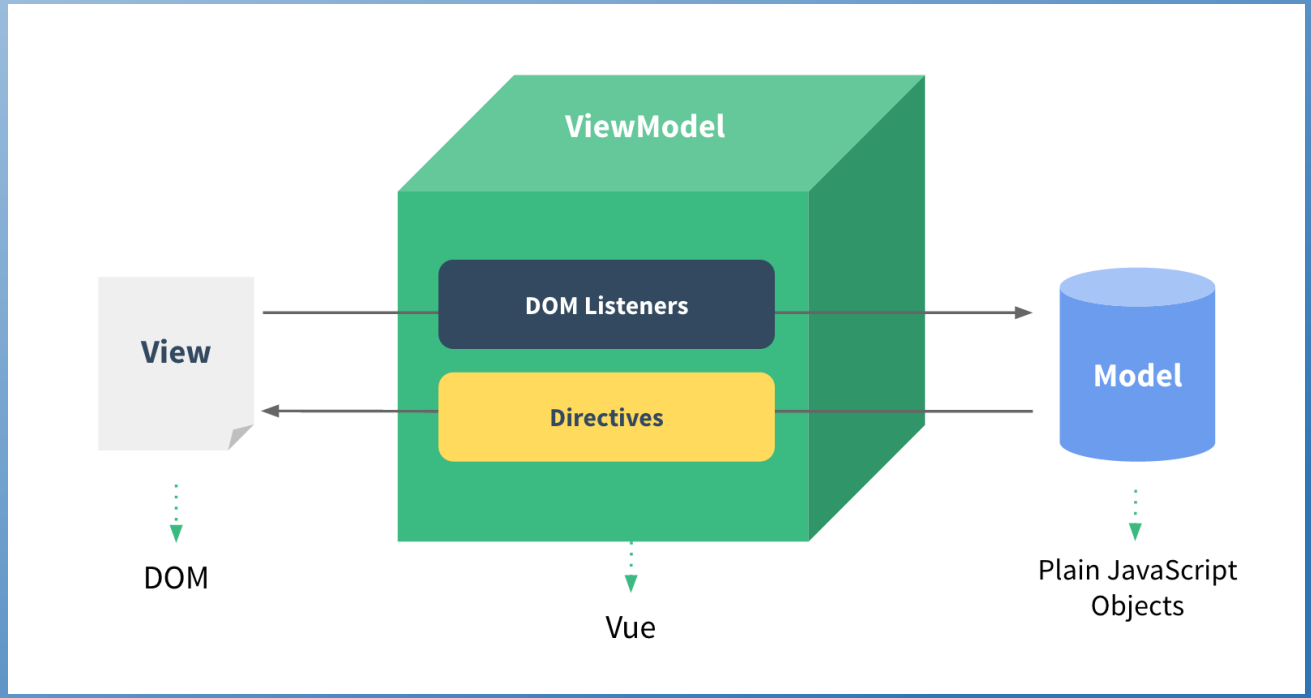


MVC Issues: Duplication

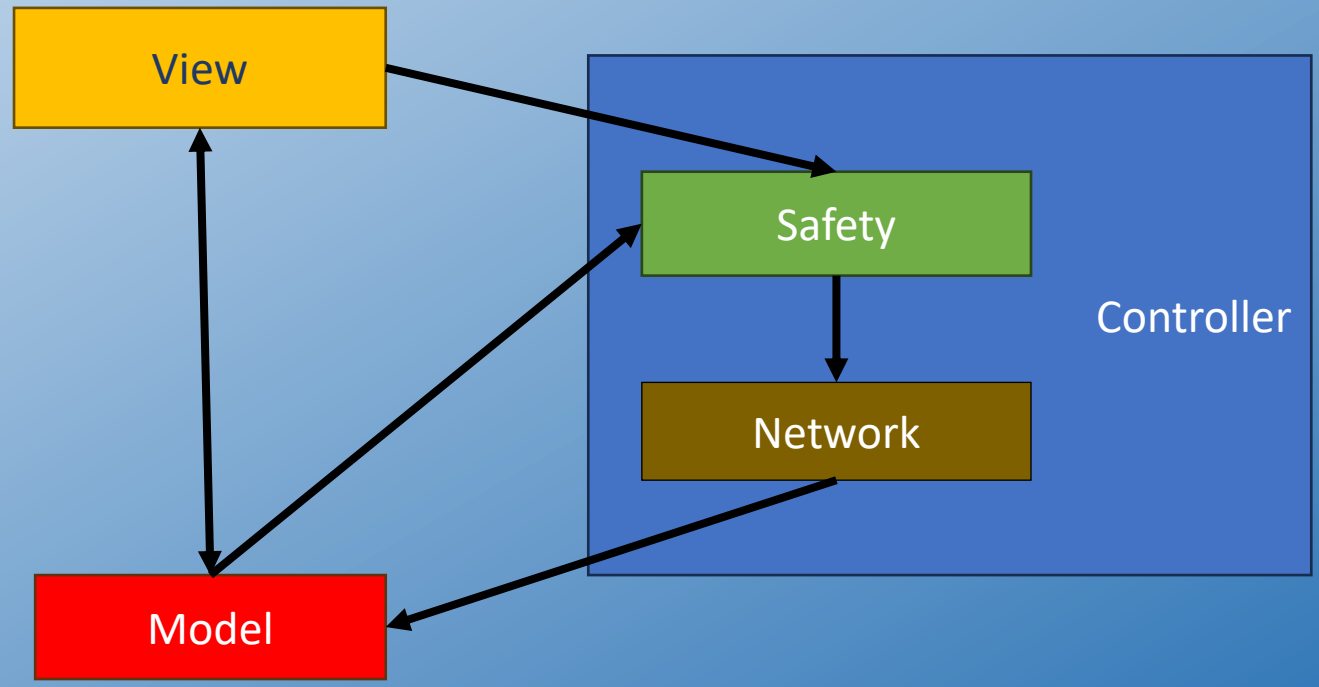
- The model might be duplicated
  - Actual model data structures in memory
  - View concept of the model (for display & editing)
  - Database concept of the model (for permanent storage)
  - Controller concept of the model (for update commands)
  - SHERPA: user's universe as JSON, data structures
- This means that evolving the model can be complex
  - Changes required in several places of the program
  - Leads to more code, errors, complexity, inefficiency
- This has led to various alternate solutions
  - Accessible (public) model, ORM, embed controller in model, …

MVVM Architectures

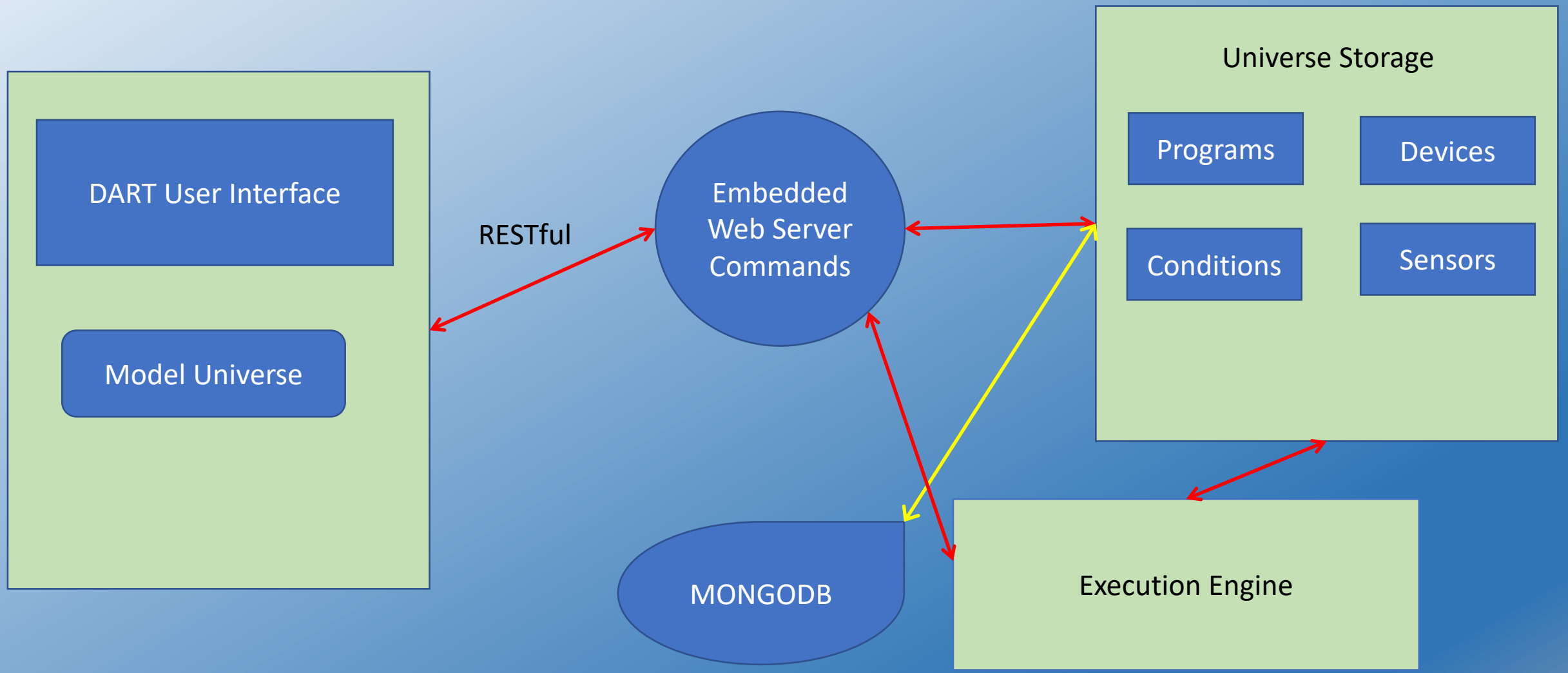
- ViewModel translates between view and model
  - Acts as simple controller
  - Makes model objects easy to manage and present
  - Handles most of views display logic
  - VUE



MVC Design Example: SHORE



MVC Architecture Example: SHERPA

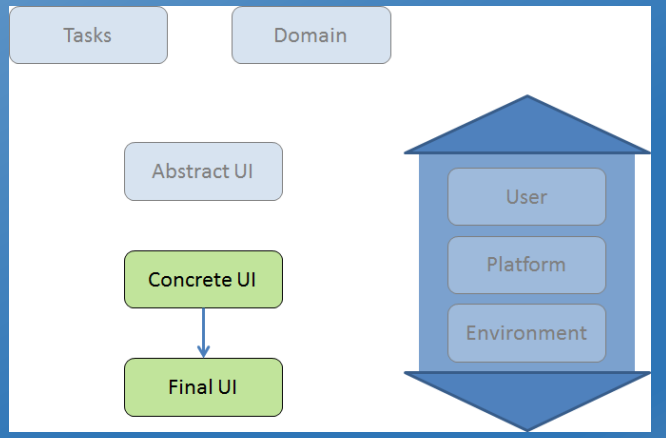


Exercise

- How does your programming assignment fit MVC
  - All these versions of MVC; which (or none) is yours?
  - Breakout into small groups
  - Say hello
  - Sketch out your MVC model quickly
  - Discuss how your programming assignment fits the MVC model
    - Do you have a model?
    - Do you have a controller?
    - Do you have a separate view?
    - How do these interact?
- 5-10 minutes

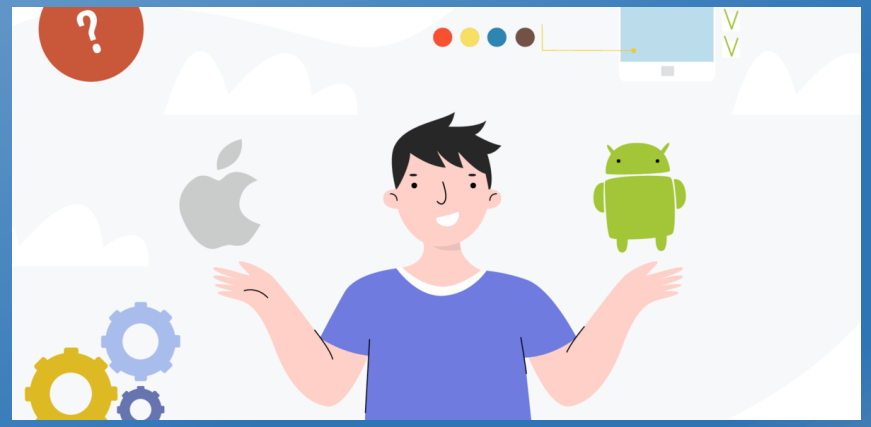
User Interface and High-Level Design

- Determine what MVC approach is appropriate
  - Based on complexity of model, control, user interface
  - Based on software architecture & specifications
  - Based on whether UI is separate process or not
  - Based on the needs of the system (importance of UI)
  - Determine responsible design components
- Separate the different MVC components
  - As much as possible and practical
- Avoid duplication
  - Avoid implementation dependencies
- Ensure high level design reflects your approach

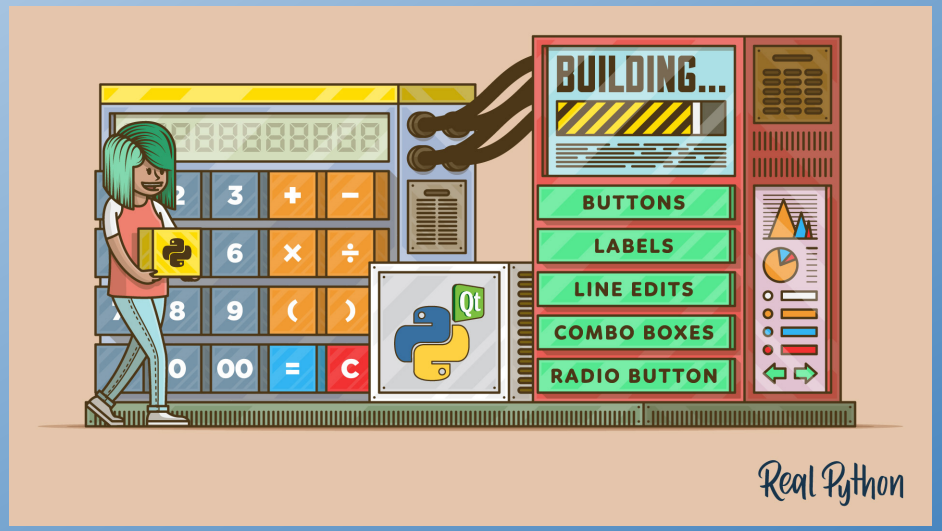


UI-Development Alternatives

- How the UI fits into the application affects development
  - It affects how one designs the application
  - It affects how one designs the user interface
  - It affects how one approaches the implementation as well
- Alternatives
  - UI-First development
  - UI-Centric development
  - UI-Last development



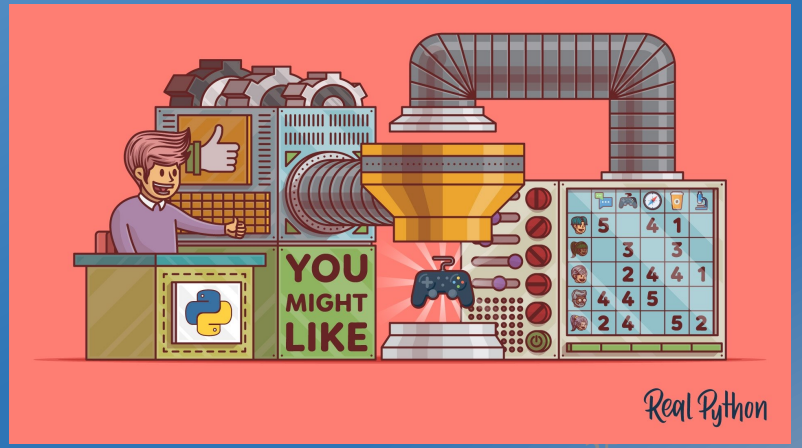
UI-First Development



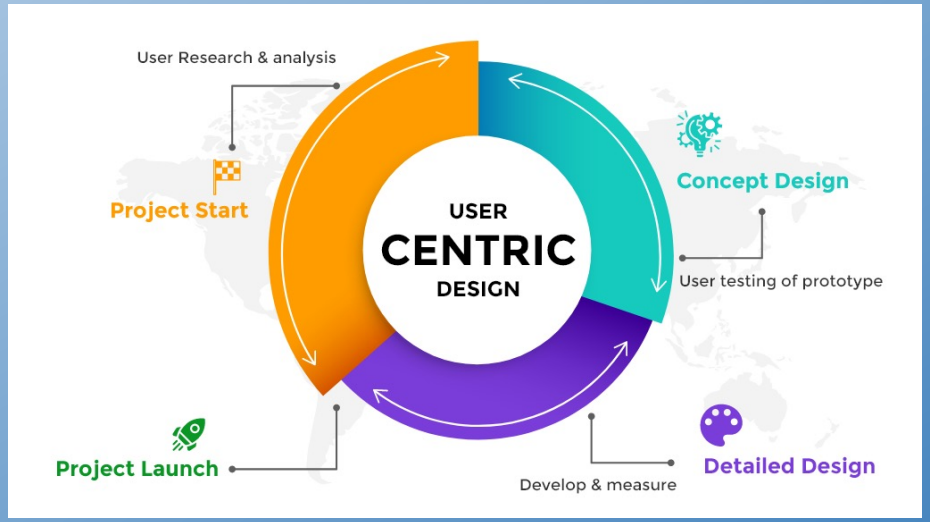
- Build and test a user interface first
  - Prototype
  - Multiple prototypes if needed
  - Relatively complete user interface
- Design the rest of the application around it
  - Implement UI commands one at a time
    - As needed for your scenarios
  - Continually test the interface
    - Executing commands
  - Can build the essential pieces in parallel
    - But might need to revise their interfaces to accommodate the UI
- Web apps (SphereE, Twitter Data), Code Bubbles

UI-First Development Pros and Cons

- Good chance of getting the interface right
  - Multiple prototypes
  - Lots of experience using the interface
- Provides a framework for the application
- Provides criteria for measuring performance, usability, etc.
- Slows down application development
- More difficult for multiple-person teams
- Akin to test cases first
  - User interface acts as the test framework



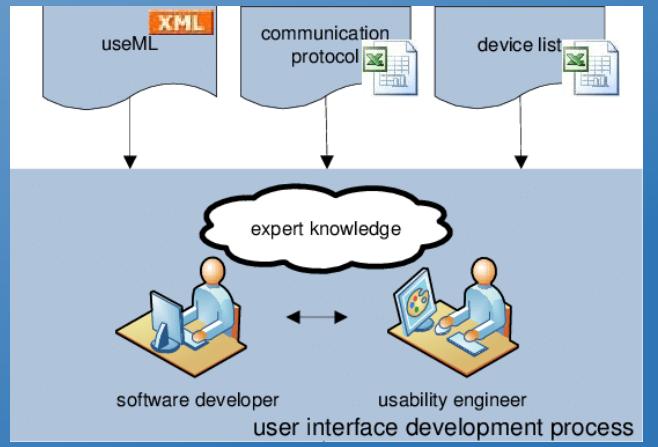
UI-Centric Development



- Many features are tied to the UI
- Build the application based on features
- Start with a very simple user interface
- Add the UI buttons and interactive capabilities one by one
  - Then add the implementations
  - Add the UI for the button, then add callback, then the implementation
  - Do this for each button/capability/dialog/panel/…
  - Can rearrange or modify the interface once buttons work
- Choose important parts of the applications based on UI & scenarios
- Can develop much of back end in parallel
- Fits in with agile development
- Examples: DYMON, SEEDE

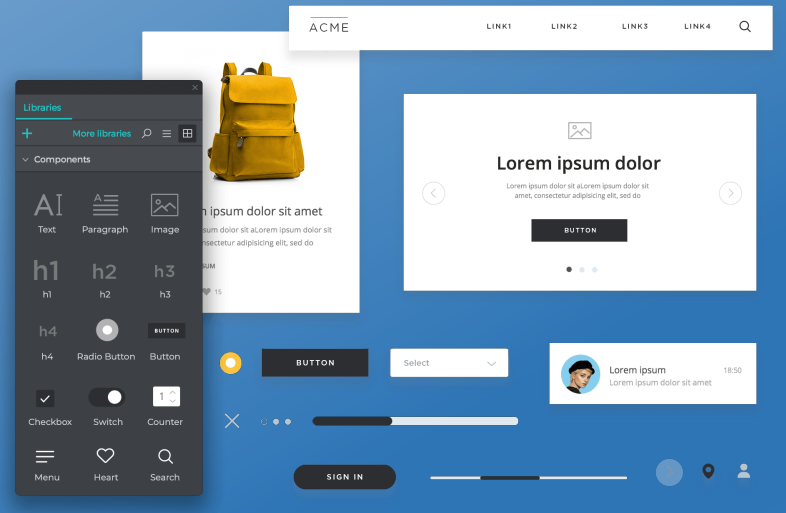
UI-Last Development

- Get an understanding of what will be needed for the UI
- Create a component (interface/façade) for the UI
  - As part of the high-level design
  - Can have a dummy implementation (or none)
- Get the main implementation working
  - Use the UI façade or just print statements to test it
- Then work on one or more user interfaces
  - Might require minor changes to the design, but not much
- Examples: SHORE (swing/javafx), SHERPA

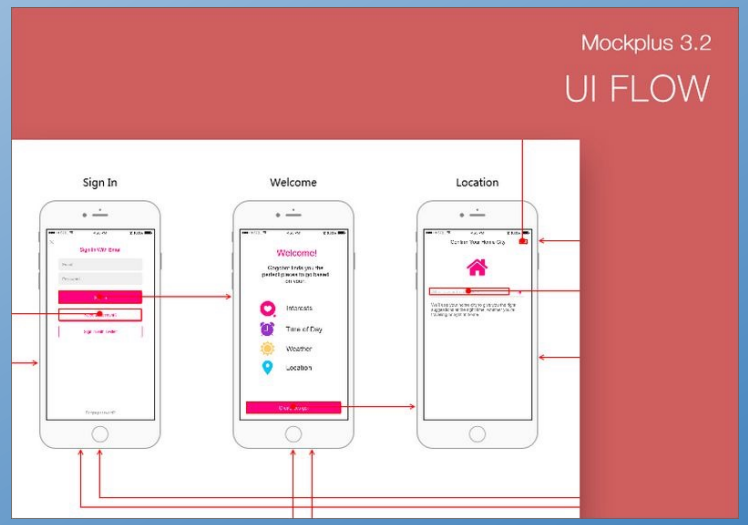


Prototyping the User Interface

- Design methods suggest designing 3+ different user interfaces
  - User can choose one or create one with ideas from all 3
- Development methods talk about building multiple UIs
- Need a concrete way of evaluating the UI
  - Having the wrong UI is a risk
  - Its difficult to know what is the correct UI
- Building multiple examples can be helpful
  - Can be shown to user to get feedback
  - Can play with it to get experience
    - By yourself (too tolerant)
    - By others

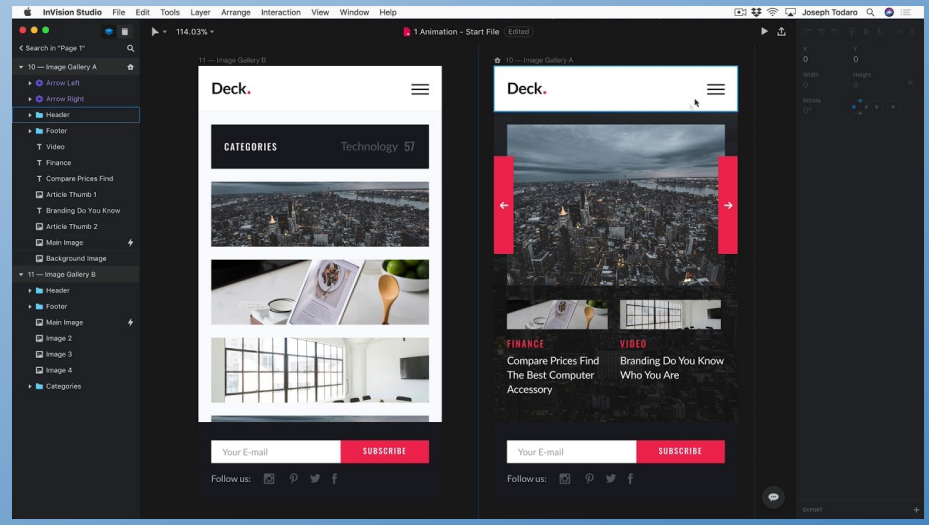


Types of Prototypes



- Low-fidelity: paper sketches
  - Akin to what is provided with specifications
  - Simple and cheap to create
- Medium-fidelity: clickable prototypes
  - Something the user can play with
  - Not necessarily detailed or pretty, just give a sense of design
  - Give a sense of flow with dummy results, etc.
  - Static web pages and links for a web application
  - Front-end only mobile applications
  - InVision is an example of a tool for this

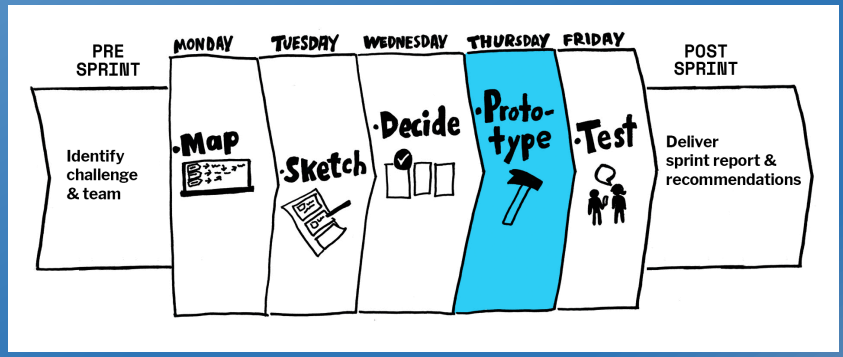
High-Fidelity Prototypes



- What looks like a complete application
  - Includes interactions and transitions
  - Coding the user interface portion of the system
- Tools let you do this without the code
  - InVision, Principle, Framer, UXPin, Figma, HTML w/o JavaScript
- Mocking code to replace a real back end
  - Implement the calls with static (or almost static) data
    - Easy to do with HTML, can be done with other models
  - Usable but non-functional implementation
  - User can evaluate the interface

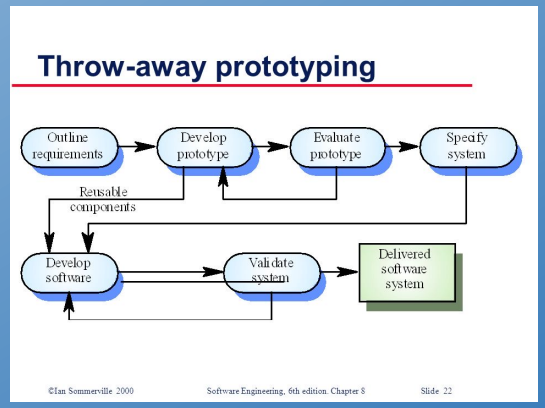
What is a Prototype

- Quick and dirty implementation to try something out
- Designed to be thrown-away, not kept
  - No commitment on the developer's part
  - Not perfect, just an approximation
  - Might not be performant, look the best, …
  - Code is not "production quality"
- Designed to be easy to change
- Designed to be reimplemented if used

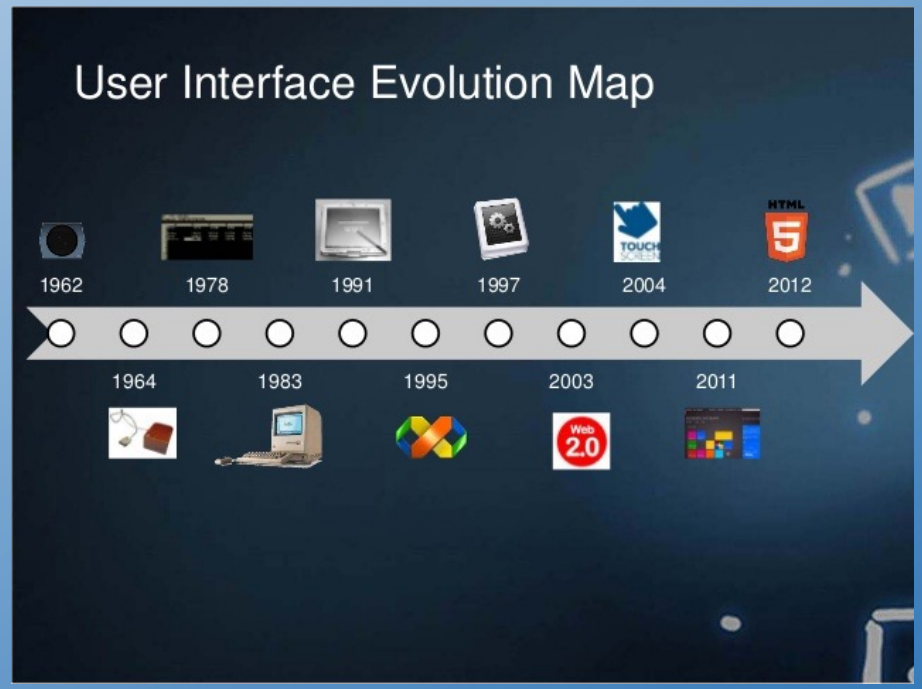


Do Developers Throw Away Prototypes

- Not really
  - They might refactor to clean them up
  - But they typically find their way into the final code
- What's wrong with this
  - Care that goes into design and code not there
  - Code is messy and difficult to maintain
- If you code a prototype, code it as if it were final
  - Or code in a different language (to force throwaway)



User Interface Evolution



- The user interface is going to change
  - As the application evolves
  - As users want more or different features
  - As the environment changes
  - Because users expect it to be “modern”
- Try to make this easy (or at least possible)
- A plug-in style design requires a plug-in style user interface
  - Core provides hooks for adding buttons, menus, etc.
  - The user interface isn’t designed as much as accumulated
  - Might need to be redesigned after a while

Homework Assignment

- We want to add a graphic notes to Code Bubbles
  - Bubble where the user could draw, image saved for the future
- Requirements
  - Should be small (sticky note sized) (minimal screen space)
  - Drawing interface shouldn't get in the way
    - Only mouse button 1, pull down menus (button 3) & keys are available
    - Interface not used frequently
      - Must be simple & obvious or self-documenting
- Develop a possible user interface
- Submit a sketch and discussion in Canvas by Thursday 10/10

PROJECT Homework

- **Tuesday 10/8: Initial Project Presentations**
- Tuesday 10/8: Initial High-Level Design (canvas)
- Following Tuesday (10/15):
  - User interface prototype for the project
    - Low-fidelity is fine for now
  - Statement of how the UI fits into the application
    - How it fits into the design
    - Approach to be used
  - Would be good to have it done earlier