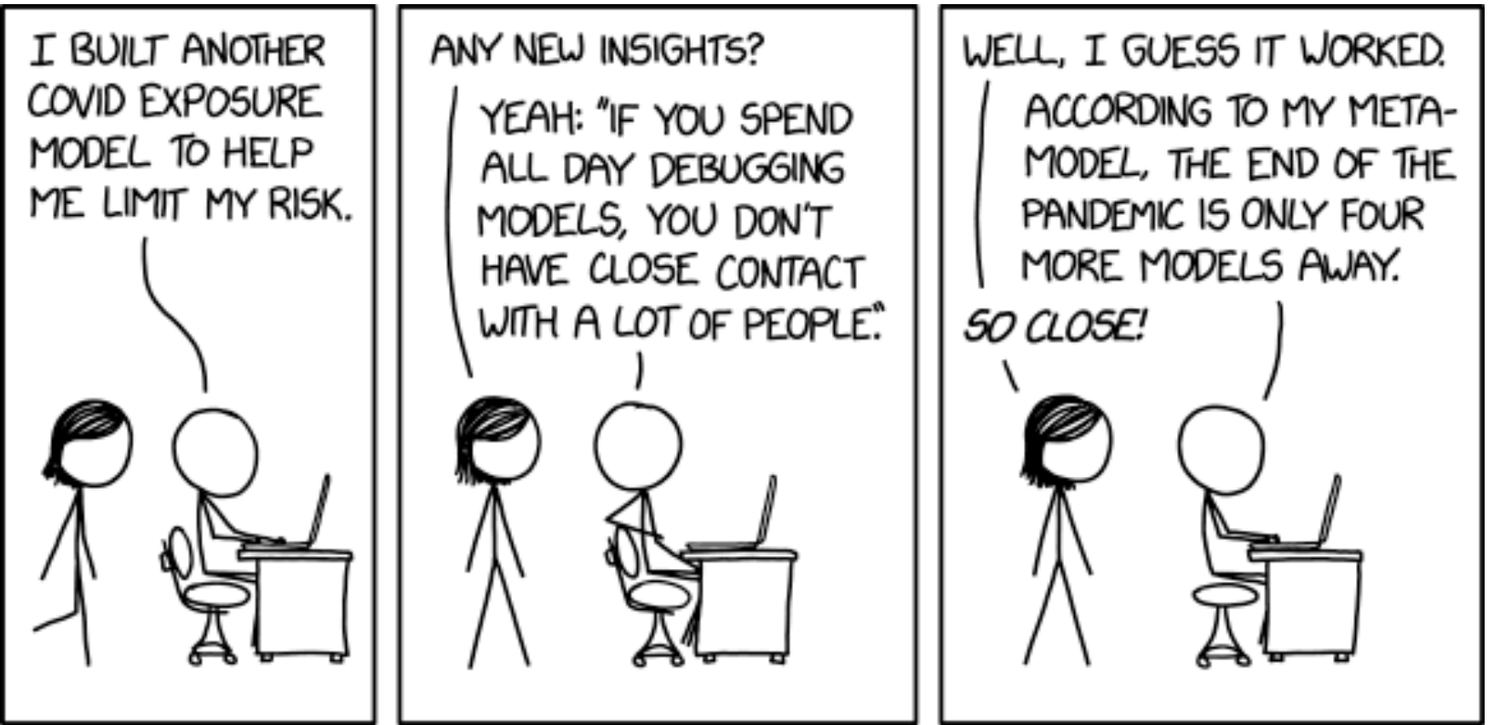
# Modeling
# And
# UI/UX



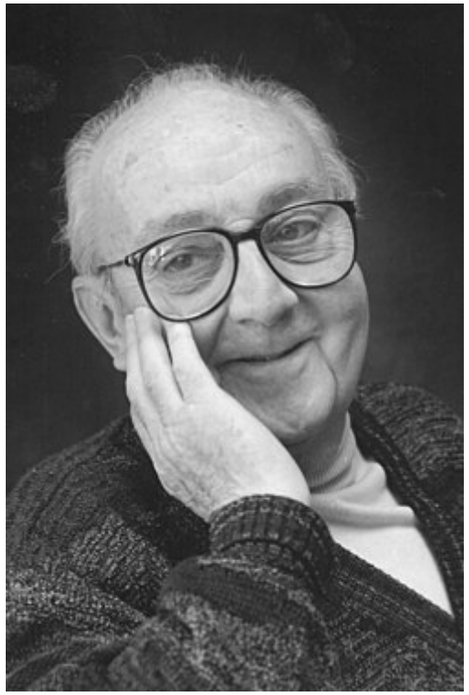


# The Story So Far …

- **QA** ensures our program works well and meets specs.
- **Requirements and Specifications** help us figure out what we are supposed to be building.
- How can we **plan** and **communicate** how we'll satisfy those requirements?
- How can we ensure the **user's experience** is (subjectively) high quality?

# One-Slide Summary

- A **Model** is a simplified representation of a system used to understand, design, communicate, or analyze that system.
  - Models are abstractions to help us manage complexity
- **User Experience (UX)** is how a user interacts with and experiences a product, system, or service. It includes a person's perceptions of utility, ease of use, and efficiency.
- A **User Interface (UI)** is an aspect of UX focused on the aesthetics of how a user interacts with the system.
  - Can be **graphical (GUI)** or textual.



**All models are wrong, but some are useful**

- George Box

A common aphorism in statistics

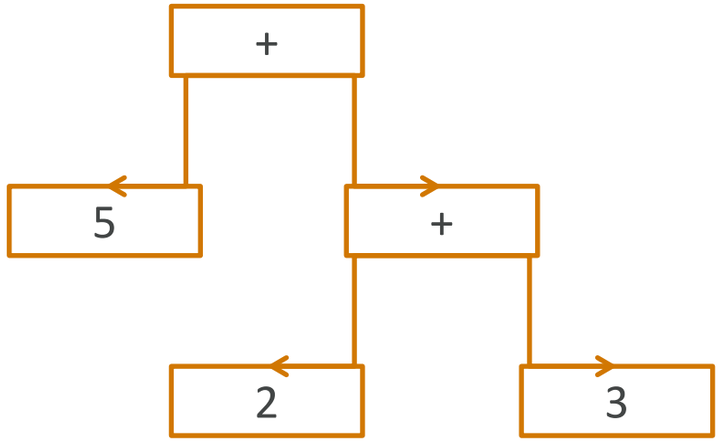
# Why Don’t We Just Write Code?

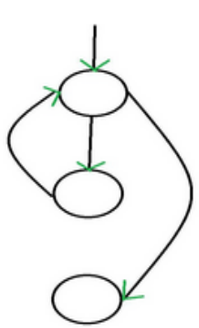
- **Systems** are too complex to understand directly from **code**
- Some **stakeholders** can’t read code
- We need to **discuss design** before building
- We need to **reason** about architecture
- We need to **communicate** across teams
- Models help us think before we build.

# Why Don’t We Just Write Code?

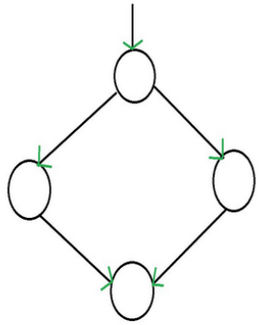
- Code can operate on models
  - Abstract Syntax Trees
  - Dataflow diagrams
  - Control Flow Graphs

Example: 5 + (2 + 3)

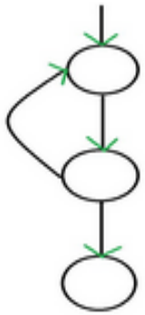


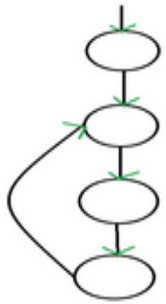
while



If-then-else



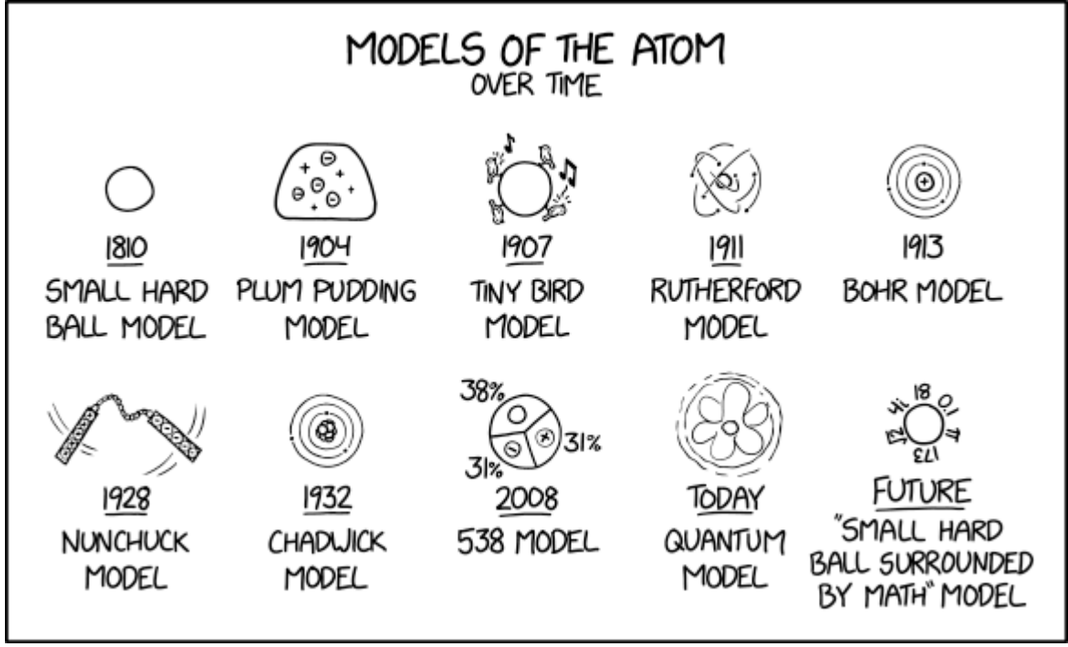
do-while



for

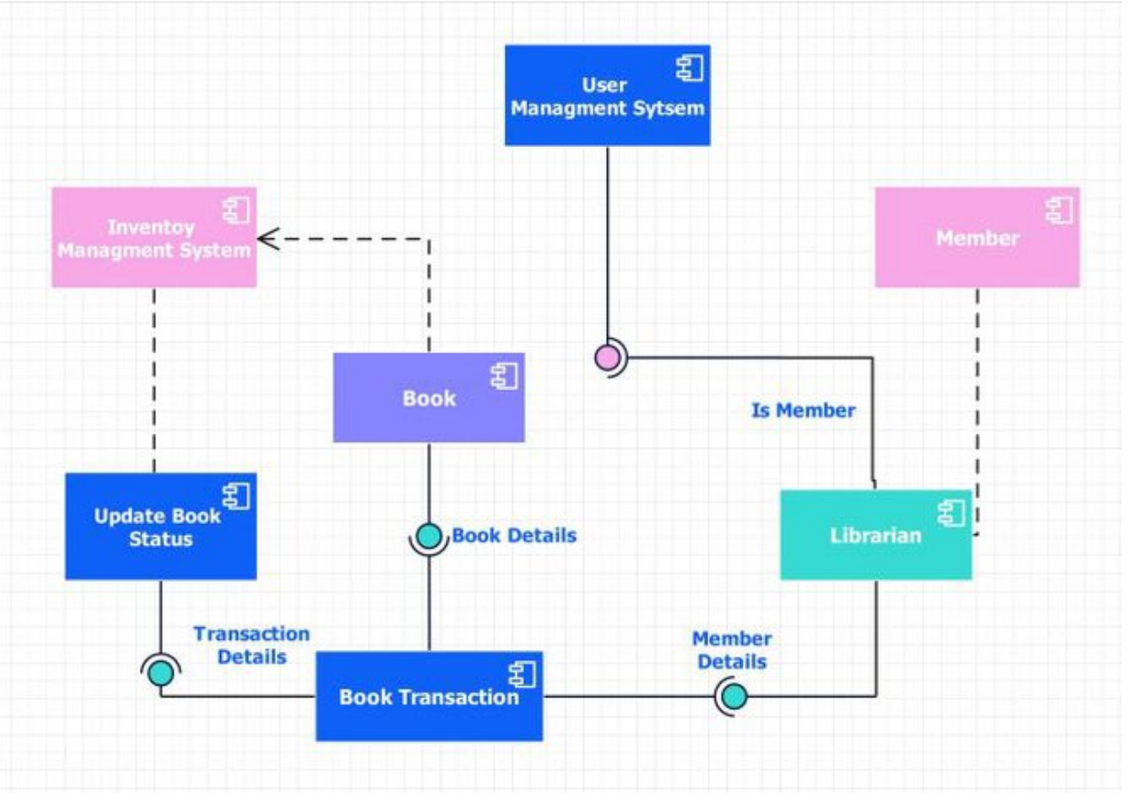
# Types of Models

- **Structural models**: what the system is
  - Class diagrams
  - Database schema
- **Behavioral models**: what the system does
  - Sequence diagrams
  - User workflows
- **Analytical/Formal models**: prove or predict things about the system
  - State machines
  - Formal specifications



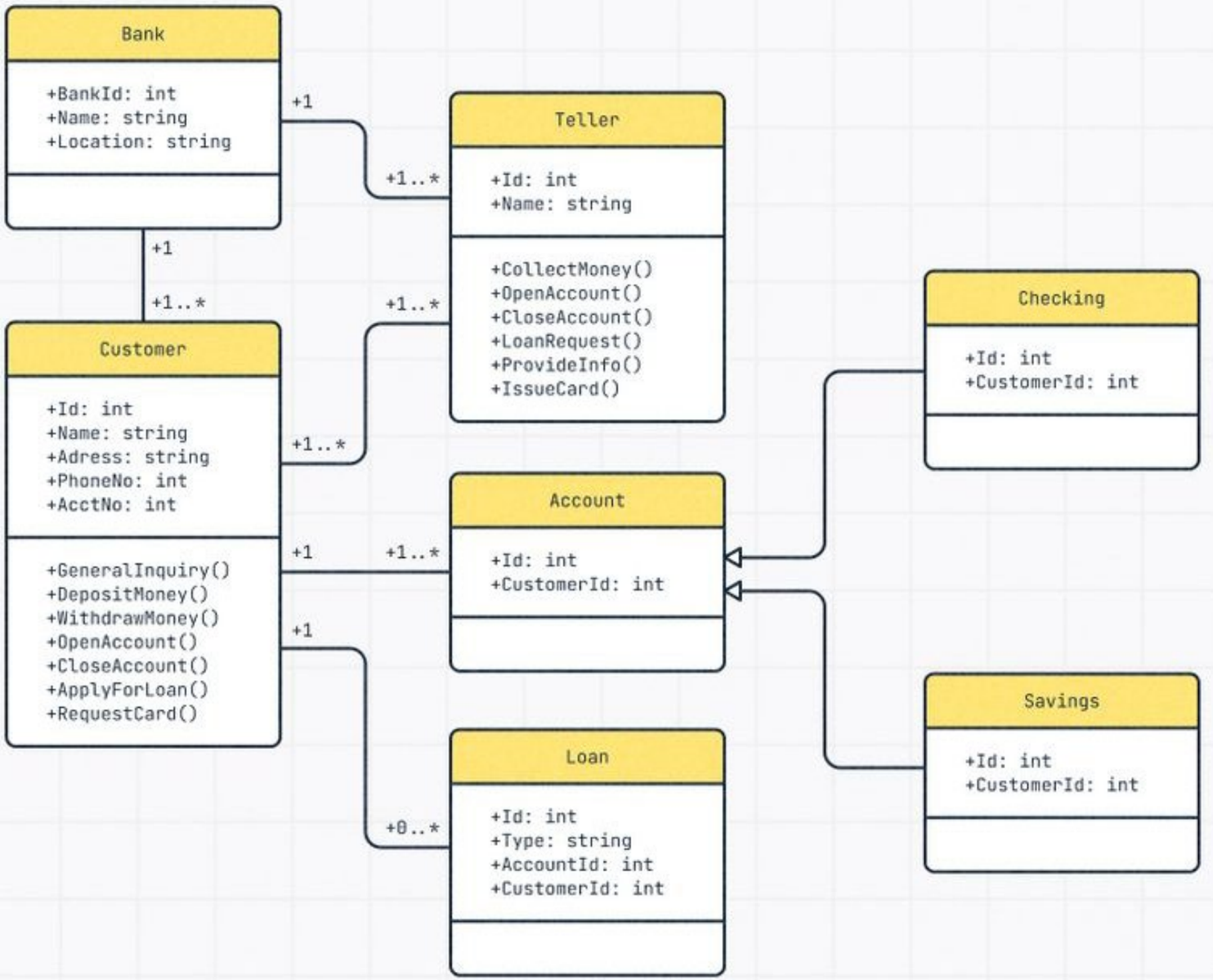
# Structural Models

- **Structural models**
  - UML Class Diagrams
  - ASTs
  - Database Schemata
  - Entity Relationship Diagram



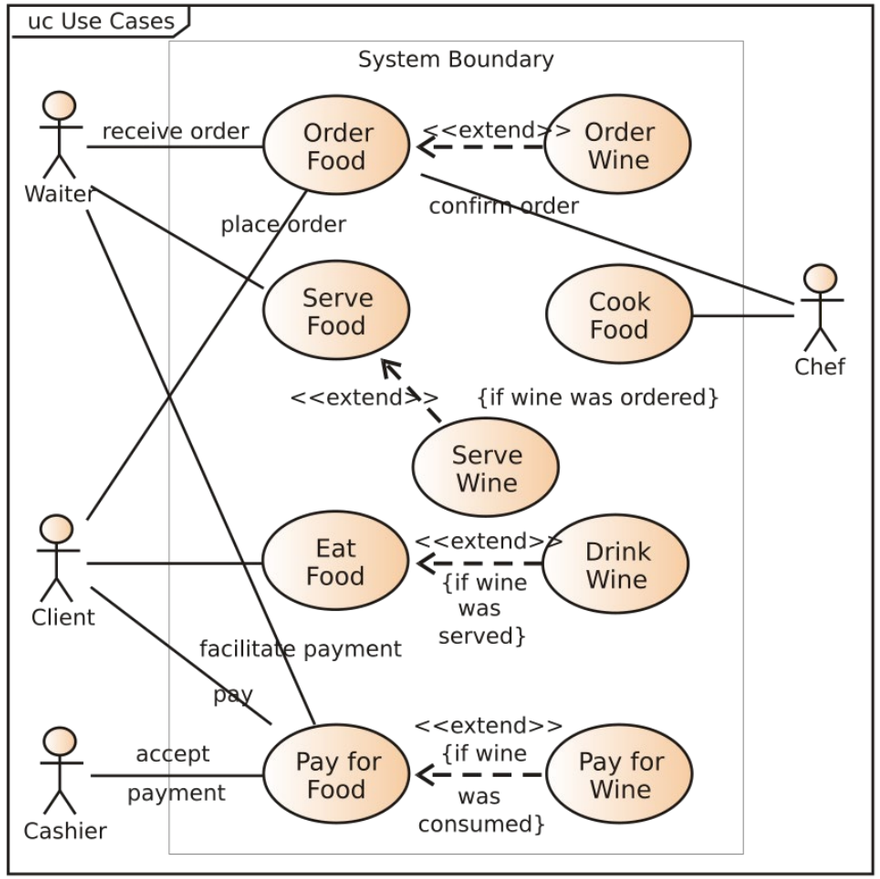

- Used for architecture, data design, interfaces, dependencies

# Class diagram for a banking system

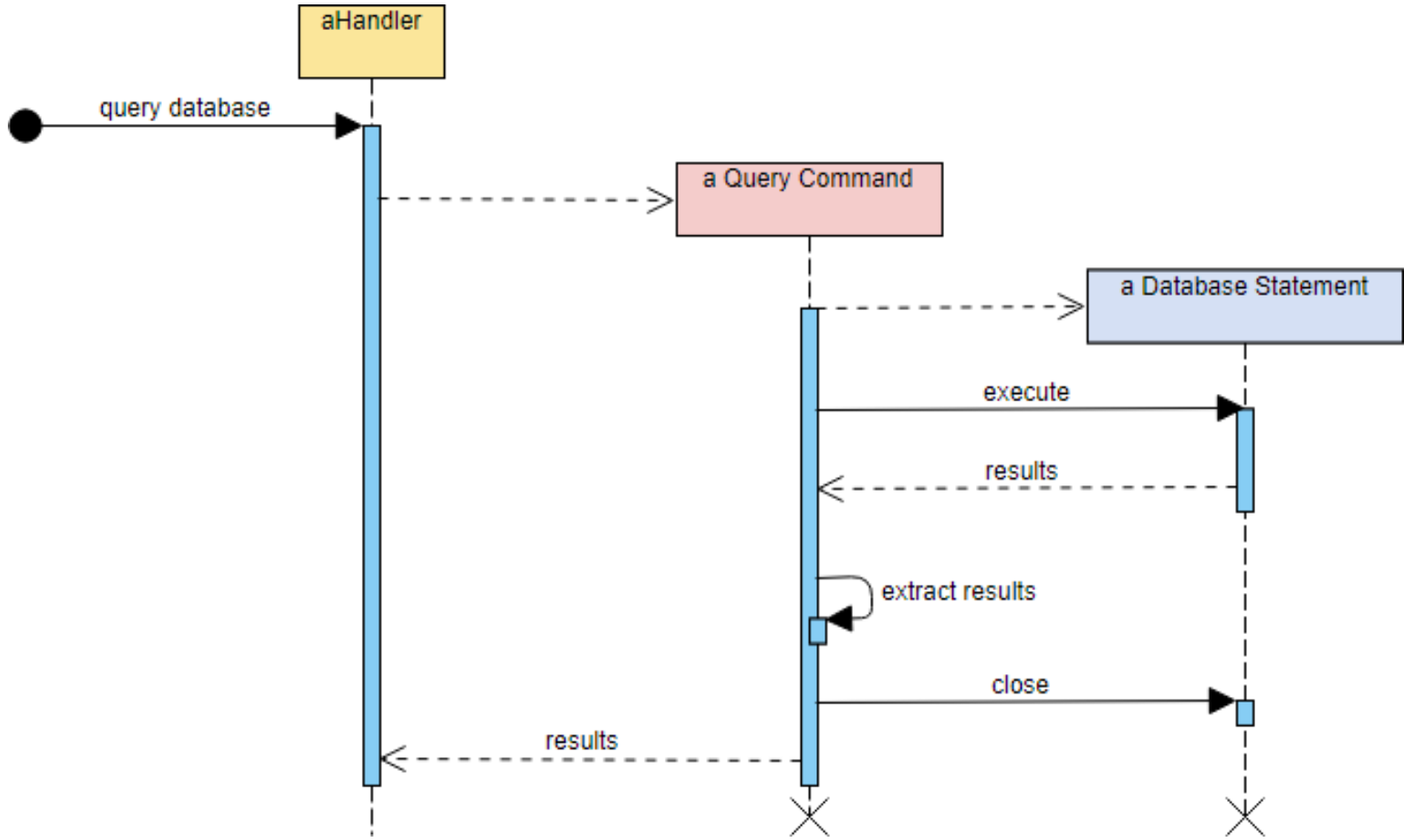


# Behavioral Models

- **Use Case Diagram**




**Sequence Diagram**

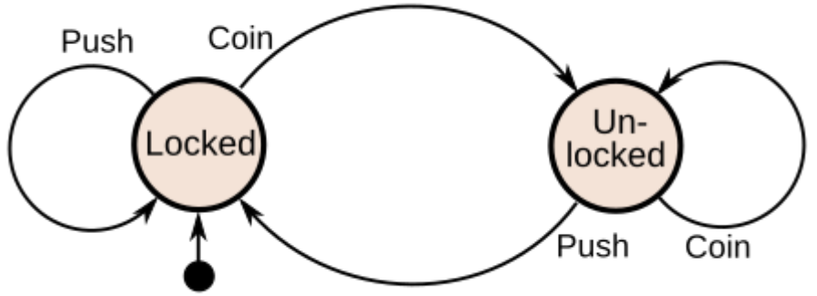



- Used for business logic, feature design, requirements
- Dataflow diagrams and CFGs are also behavioral

# Formal Models

- Finite State Machines
- Formal Specifications
  - Z3
  - TLA+
  - Alloy

FSM for a turnstile



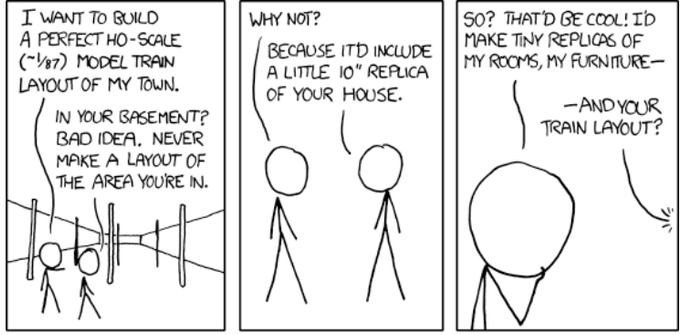

TLA+ 1-bit ticking clock

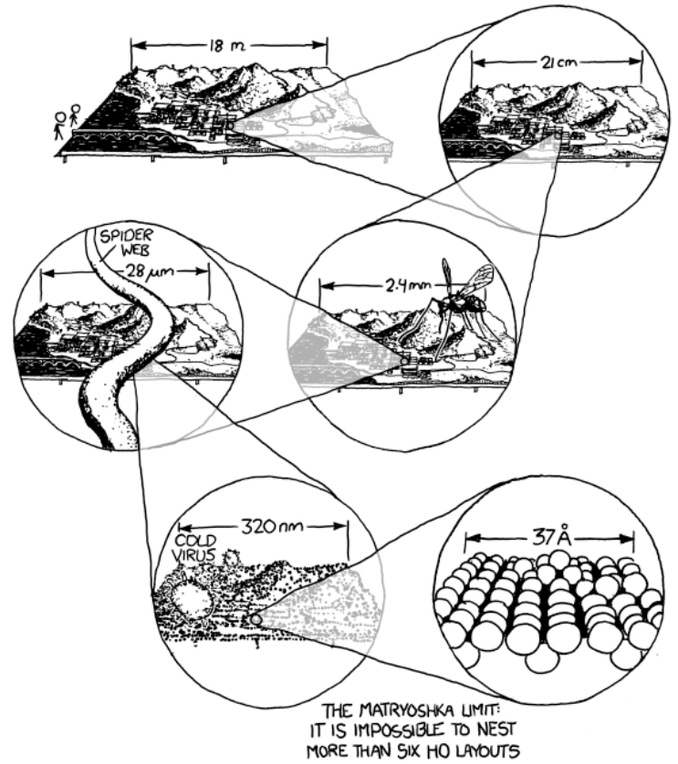
```
VARIABLE clock

Init == clock \in {0, 1}

Tick == IF clock = 0 THEN clock' = 1 ELSE clock' = 0

Spec == Init /\ [][Tick]_<<clock>>
```

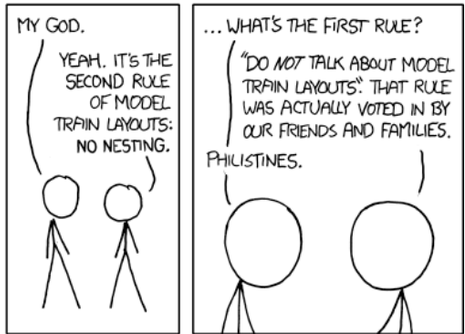
- Used for safety critical systems, proof of correctness/safety/soundness/completeness

# Models in Modern SWE

- Modern software models are often **executable, simulatable, or directly translatable** into code
  - Z3
  - TLA+
  - Alloy
  - Kubernetes YAML (IaC)
  - OpenAPI
  - SQL schema
- Often used for **Model Checking**

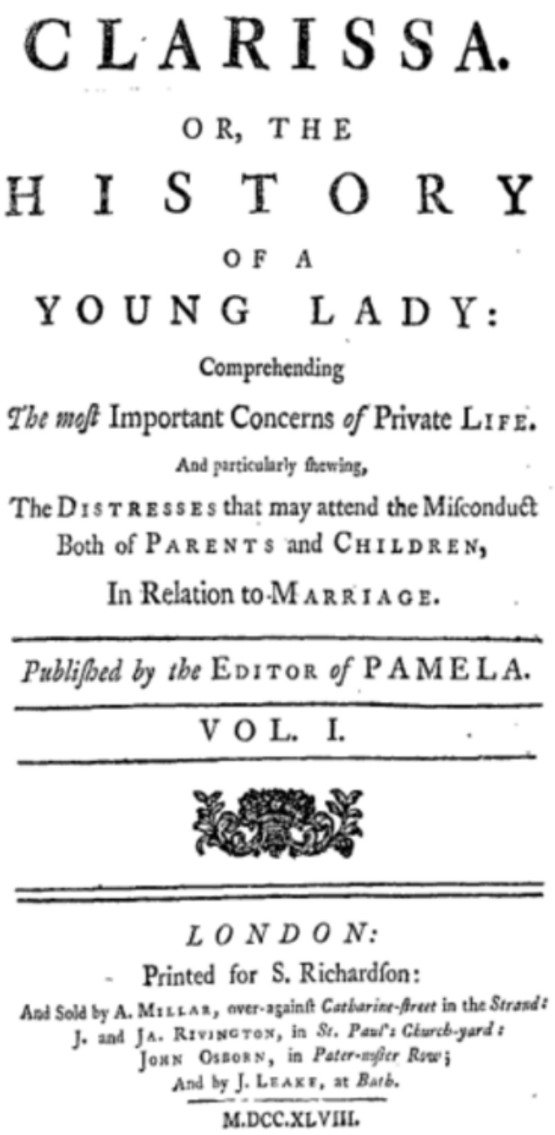


# Trivia: Literature

- The longest piece of English literature is widely considered to be this “non-traditionally published” work with **34,754,159 words** and 2318 chapters. The website hosting this work had its word counter’s 24-bit integer limit overflowed, requiring an external count to track its length.
- For traditionally published works, Samuel Richardson's "Clarissa" (~1,000,000 words) is generally considered the longest single continuous narrative in English.
- For reference, the Lord of the Rings trilogy is less that half a million words, while the entire Harry Potter series is 1,084,170 words


CLARISSA.

OR, THE

HISTORY

OF A

YOUNG LADY:

Comprehending

*The moſt* Important Concerns *of* Private LIFE.

And particularly ſhewing,

The DISTRESSES that may attend the Miſconduct Both of PARENTS and CHILDREN, In Relation to MARRIAGE.

*Publiſhed by the* EDITOR *of* PAMELA.

VOL. I.

*LONDON:*

Printed for S. Richardſon:

And Sold by A. MILLAR, over-againſt *Catharine-ſtreet* in the *Strand*: J. and JA. RIVINGTON, in *St. Paul's Church-yard*: JOHN OSBORN, in *Pater-noſter Row*; And by J. LEAKE, at *Bath*.

M.DCC.XLVIII.

# UI/UX

- Or, Designing software that is usable by humans (who may not have technical expertise).


Why are they called "semantic arguments" and not "ad homonyms"?

| UI | UX |
|---|---|
| How it looks | How it feels |
| Visual design | Overall experience |
| Buttons, colors, layout | Usability, flow, satisfaction |

# What do users care about?

- Users don’t care about your architecture
- Users don’t care about your algorithms
- Users care if the software is:
  - Easy to use
  - Fast
  - Clear
  - Doesn’t make mistakes
  - Doesn’t frustrate them
- From the user’s perspective, the UI *is* the system.

# Bad UI/UX


Available at: github.com/GoulartNogueira/BadUI

Made with ♥ by André G. N.

# Bad UI/UX

- Poorly designed UI/UX causes
  - User errors
  - Support costs
  - Training costs
  - Lost customers
  - Safety issues (medical, aviation, automotive)
  - Rewrites and redesigns

- All of these are expensive

# Core UI Design Principles

- **Consistency:** Similar things should look and behave the same
- **Feedback:** The system should tell the user what is happening
- **Reversibility**: Users should be able to interrupt or reverse actions
- **Affordance:** Controls should look like how they are used
  - buttons look clickable, scrollable areas have a scroll bar, etc
- **Error Handling:** Handle errors gracefully instead of showing error messages
- **Simplicity:** Don’t make users think too much

# Core UX Design Principles

- Know your users
- Know their goals
- Minimize steps
- Reduce cognitive load
- Design for errors
- Make the common case fast and easy
- Make the system predictable

# UI Design -- Constraints

Interfaces must take into account the physical constraints of computers and networks.

- How does a desktop differ from a smartphone? A remote?
- How do you make use of a touch screen?

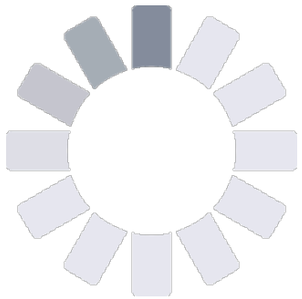
Constraints that the interface must allow for:

- performance of device (e.g., fast or slow graphics)
- limited form factor (e.g., small display, no keyboard)
- connectivity (e.g., intermittent)

# UI Design -- Unpredictability

Any operation that transfers data over a network has unpredictable response times and is subject to delay.

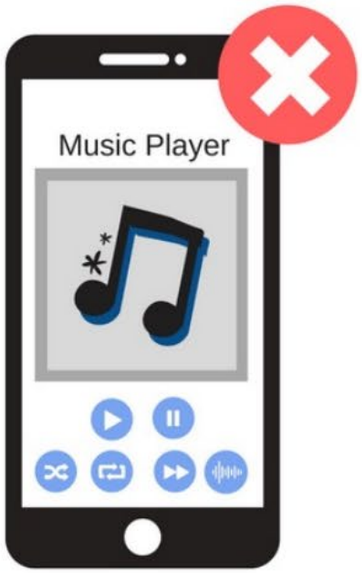
- Large data transfers should run asynchronously in a separate thread.
- Provide visual feedback to indicate that the operation is in progress.
- Provide a way for users to cancel long running data transfers

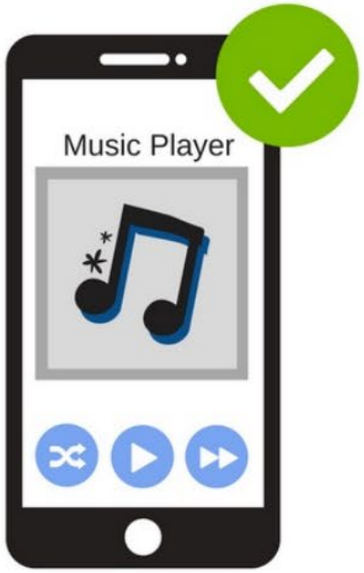


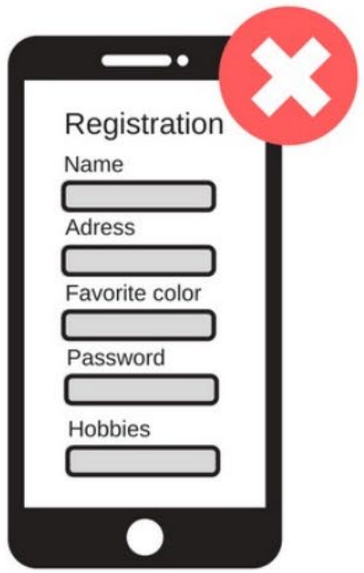
# UI Design -- Accessibility

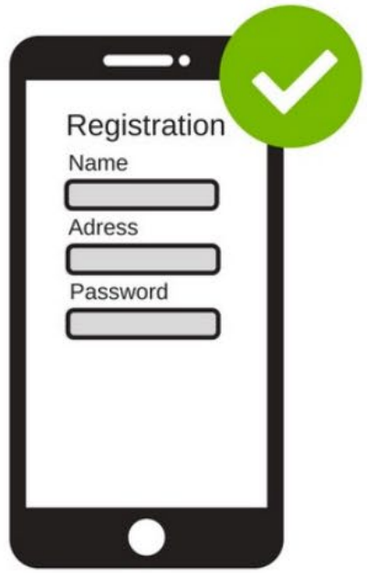
Smartphones have small displays and small virtual keyboards.

- People with poor eyesight, color blindness, hearing loss, or clumsy fingers may have difficulty using your applications.
- For your user testing try to find people who do not have perfect eyesight, hearing, etc. Have testers of various ages.
- Older people are often less able to use touch sensitive screens.



# UI Design -- Responsiveness

- Ideally, a single web site adapts to any device by using a mix of flexible grids and layouts, and careful use of CSS media queries.
- Many issues:
  - Less content can be displayed on a small screen
  - Smartphones are usually used in portrait.
  - A touch screen has different characteristics from a mouse.
  - A virtual keyboard needs screen space.

# UI Design -- Menus

Advantages

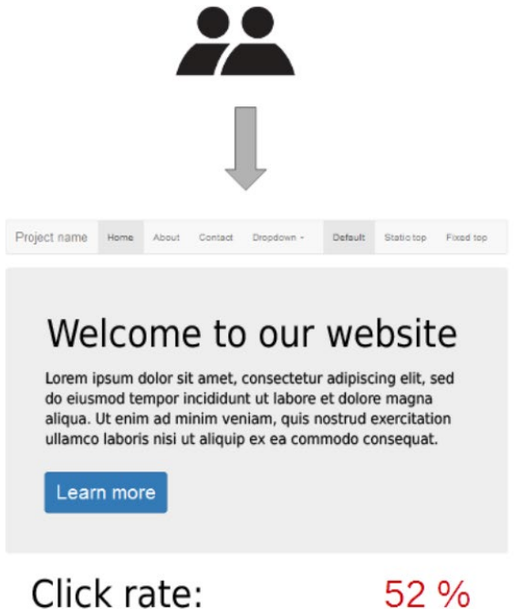
- Easy for users to learn and use.
- Certain categories of error are avoided.

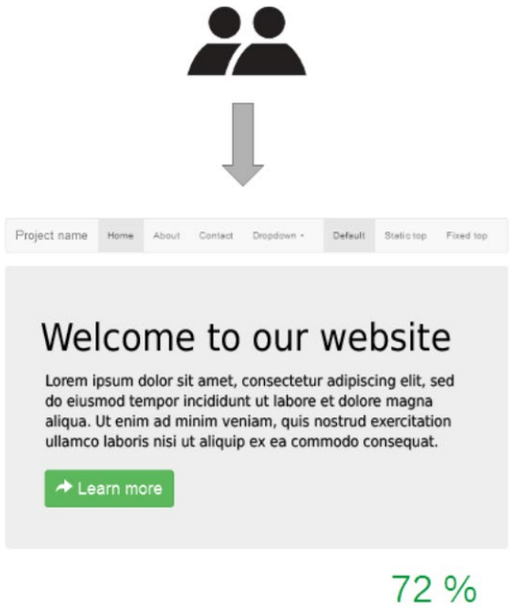
Major difficulty is structure of large number of choices.

- Scrolling menus (e.g., states of USA)
- Hierarchical
- Associated control panels
- Users prefer broad and shallow to deep menu systems.

# Usability Testing

- Give your UI to a user and watch them try to use it
- Don’t explain the interface
- Watch where they struggle
  - Gain insights into how to update the UI to make it easier
- Measure **interaction costs**, the number of interactions it takes to perform common tasks.
- A/B testing is a common way to measure preference and usability

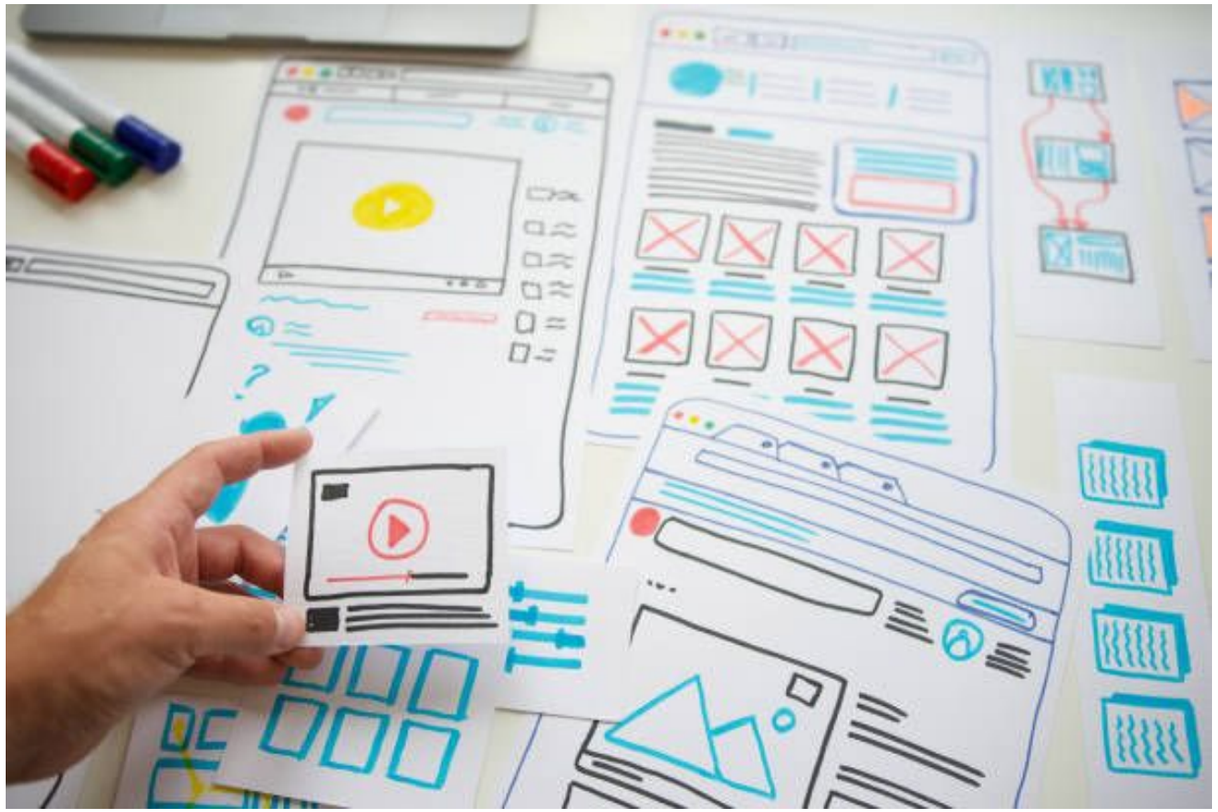


# Prototyping

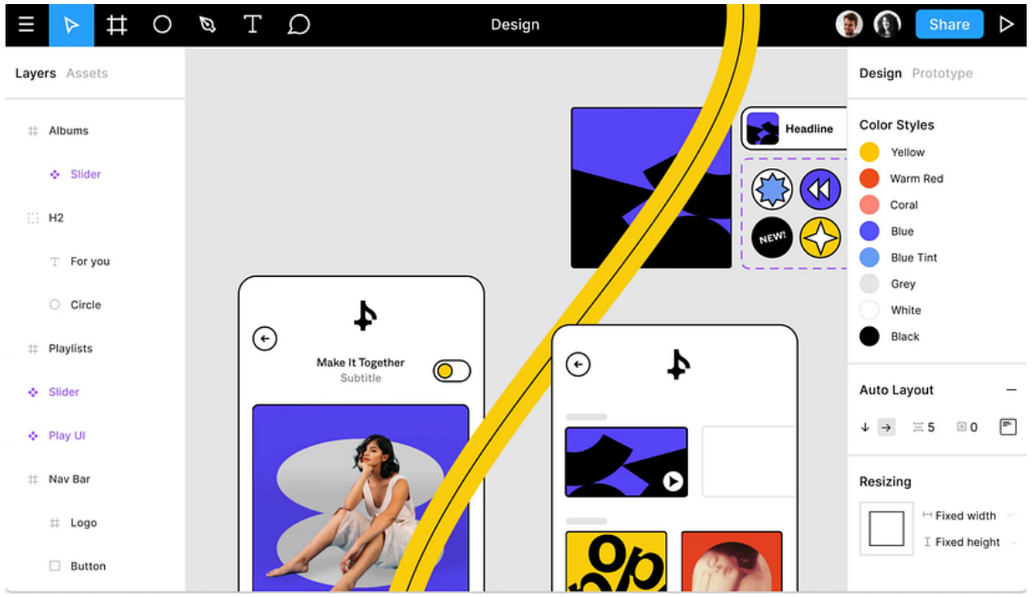
A **prototype** is a preliminary version that can be used to iterate rapidly between requirements and design.

- Test Variations
- Get feedback from users
- Communication between design team, other stakeholders
- Find failure modes, fail early



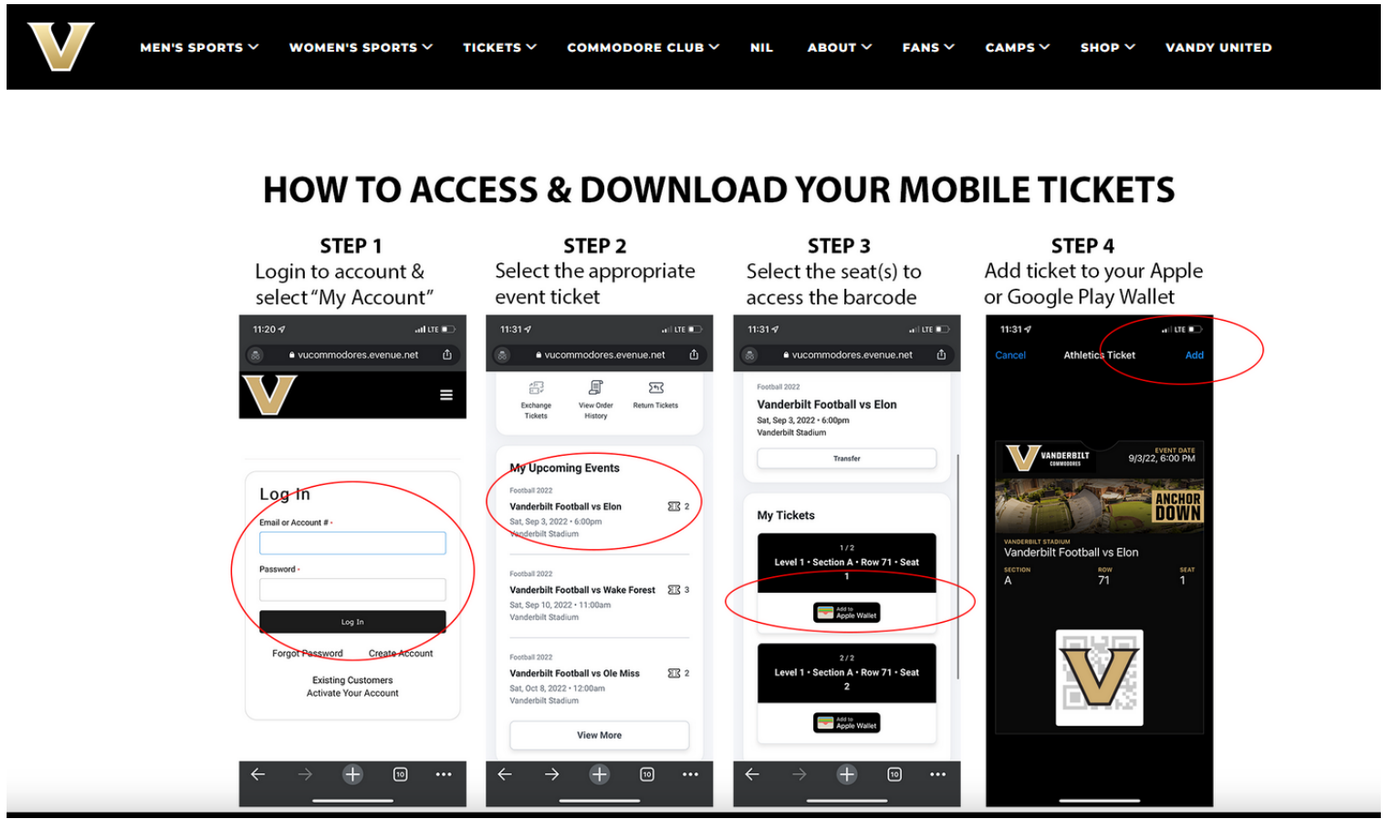
# Prototyping Methods

- **Low Fidelity** – fast and easy to create, shows basic functionality
  - Paper sketches
  - Storyboards
  - PowerPoint slides
- **High Fidelity** – looks and feels a lot like the final product
  - HTML website
  - Wireframing tools
    - Draw.io
    - **Figma**



# Prototyping A User Interface

- The best way to learn UI design is to do it
- So let's practice



# In Class Activity: Prototyping A User Interface

- Group up (ideally with your group from the last activity)
- Open your requirements from your GitHub repo
- Go to figma.com
- Make a prototype wireframe for your ticketing webapp based on your requirements
  - Must include at least 1 page for each teammate
  - Webpages should be interactable and connected in the prototype
- Submit the share link to Brightspace