UNIVERSITY *of* WASHINGTON

Software Design for Data Science

Software & Use Case Design

Melissa Winstanley
University of Washington
February 1, 2024

Today’s agenda

- Lecture on design
- Design exercise
  - Create a `docs` directory within the root of your project repository
  - Create Markdown files in the `docs` directory
  - Add/commit/push at the end of class
  - More exercises next week
  - Finish up the design exercise for review by 2/15

Software Design

Software Design

“…a specification of a software artifact intended to accomplish goals, using a set of primitive components and subject to constraints”

- Wikipedia

Why software design?

"The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is so difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.

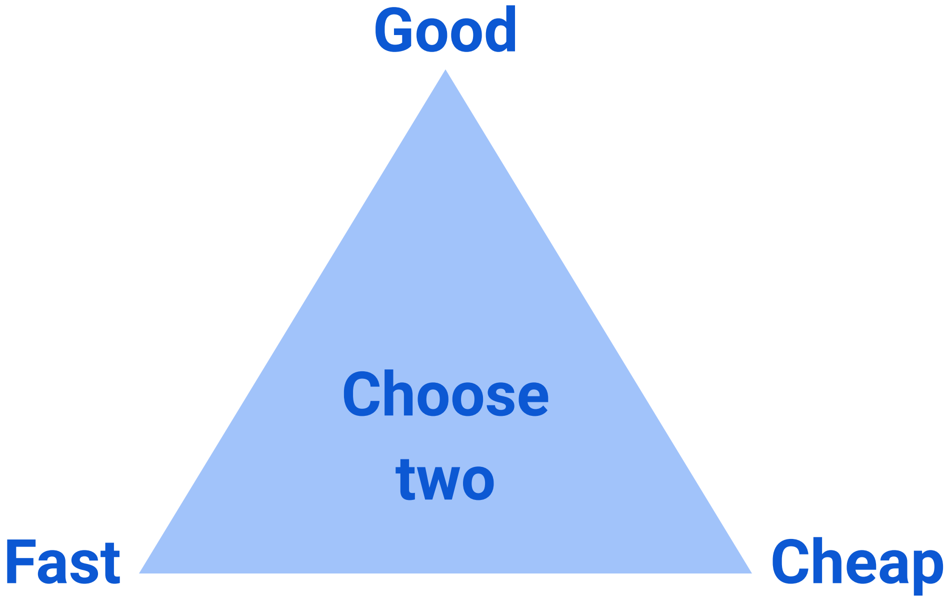
Therefore the most important function that software builders do for their clients is the iterative extraction and refinement of the product requirements."

- Fred Brooks, *The Mythical Man-Month*

Fred Brooks' suggested planning schedule

- 1/3 for design
- 1/6 for coding
- 1/4 for unit/component testing
- 1/4 for system testing

The Classic Dilemma



Good

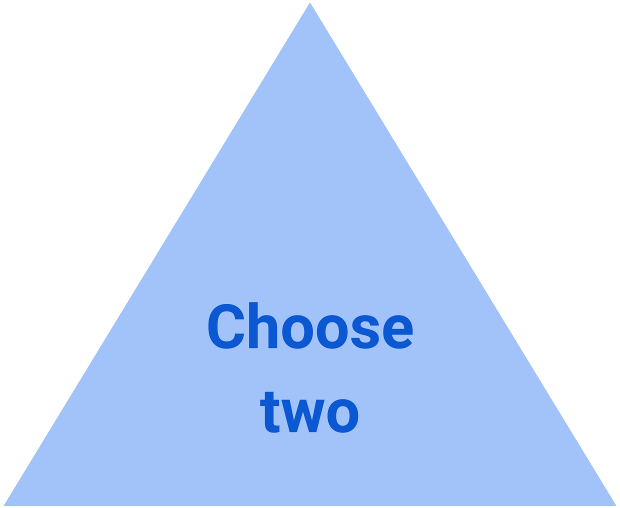
Choose two

Fast

Cheap

The Classic Dilemma: Software Edition

Features



Choose two

Time

Resources

Prevent Feature Creep

- **Feature creep**: gradual accumulation of features over time
  - Often has a negative overall effect on a project
- Why does it happen?
  - Features are “fun”!
    - Developers like to code them
    - Marketers like to brag about them
    - Users want them
  - …but…
    - More bugs
    - More testing
    - More time

Improve Your Time Estimation

- **Time** is the most valuable resource for your project
- To spend your team’s time efficiently…
  - …you need to know **what to build**
  - …you need to know what ***not*** to build
  - …you need to know **what order** to build things
- Design can help with this!
- Estimating how long something is going to take is HARD!
  - Almost everything takes longer than you think it will
  - Design will help you order and parallelize your work so you aren’t surprised

Design Benefits

- Systematic approach to a complex problem
- Finds bugs before you code
- Enables parallel work
- Promotes testability
- Build an understandable, extendable, maintainable system

Software Design Process (for this class)

1. Identify the users
    - Who they are, what they want
    - High level
2. Functional design
    - What the system does
    - Specification
3. Component design
    - How the software will work
    - Implementation

Running Example: Inventory Management

Data: historical
product
inventory

User Stories

Start with your users

- Who are your users?
    - Researchers?
    - Policy makers?
    - Developers?
    - Children?
    - Trained monkeys?
- What do they want to do with your software?
- How are they interacting with it?
- What needs do they have?
- What skill levels do they have?

Example: Ram

Ram is a buyer for a store (he puts in orders to suppliers).

Ram wants to see current inventory of items.

Ram wants to see predictions of future purchase trends.

Ram needs to order items from suppliers.

Ram's job does not involve technical skills and he values a simple user interface.

Example: ???

What might be another user story?

Example: Valentina

Valentina is a data scientist.

She keeps the software running and responds to bug reports.

She needs to update the inventory periodically.

She needs to iterate on the prediction model to improve accuracy.

Valentina is highly technical and knows basic programming skills.

Example: Taylor

Taylor is the store manager.

She needs to see current inventory and predictions.

She wants to see orders that have been placed.

She wants to see current purchase trends.

Taylor’s job does not involve technical skills and she values a simple user interface.

Exercise: Your User Stories

- You will probably have several different kinds of “users” and a “technician” or two.
    - If you have a machine learning model backing your tool, don’t forget the “technician” who will train the machine learning model or update it.
- Take 10 minutes to write 2-3 user stories with your team.
- Use a Markdown file in the `docs` directory
- Remember
    - Who
    - Wants
    - Interaction methods
    - Needs
    - Skills

Functional Design
(Use Cases)

Example: Ram

Ram is a buyer for a store. Ram wants to see current inventory of items. Ram wants to see predictions of future purchase trends. Ram needs to order items from suppliers. Ram’s job does not involve technical skills and he values a simple user interface.

- View inventory
- View predictions for an item
- Make order

What do we do with inventory management systems?

- View inventory
- View predictions for an item
- Make order
- View purchase history graph

These are use cases - from the point of view of a user, how will they use your system?

Implicit use cases

There's an additional implicit use case in Ram's example (not explicitly stated but understood to be the case).

- User authentication!

Describing a use case

1. Objective of the user interaction
2. What information does the user provide?
3. What response does the system provide?

Usually about 3-9 clearly written steps.

This is not code!

Example: System user authentication use case

Objective: System validates that the user is allowed to access an account

User: access login page

System: display “enter username and password”

User: enter username and password on keypad

System: [if correct] show inventory screen

[if incorrect] display “incorrect username or password, please try again”

Think about edge cases!

Example: View inventory

Objective: System displays accurate inventory information that the user needs

User: access inventory page

System: display first 100 items with inventory quantity

User: selects a filter on product category, quantity, or name

System: shows subset of items

User: selects “next page”

System: shows next 100 filtered items

Exercise: Your Use Cases

- Translate your user stories into use cases (functional designs)
- Use a Markdown file in the `docs` directory
- Remember
  - Explicit use cases
  - Implicit use cases

Component Design

What is a component?

“An individual software component is a software package, a web service, a web resource, or a module that encapsulates a set of related functions (or data).”

- Wikipedia

Specifying components

Describe components with sufficient detail so that someone with modest knowledge of the project can implement the code for the component.

- Name
- What it does
- Inputs (with type information)
- Outputs (with type information)
- Assumptions
- How it uses other components

Sound familiar? Check out Lecture 3 on Interface Specifications!

Developing Component Specifications

1. What are the components in the use cases?
    a. Packages
    b. Modules
    c. Resources
    d. Data
    e. Functions
2. What components are already available?
3. What are the sub-components needed to implement those components that aren't already available?

    Do 1-2 for each such component

Example: System authentication use case

Components:

- Database with username => password data
- User interface to prompt for a username/password
- Control logic

Example: System authentication control logic

- Name
  - `authenticate`
- What it does:
  - Verifies a user is known & the password supplied by the user is correct
- Inputs (with type information)
  - *Username*, a string that is the user’s account identifier
  - *Password*, a string
- Outputs (with type information)
  - Boolean: True if success, False if failure
- Assumptions: none

Example: View inventory use case

Components:

- Data manager with inventory
- User interface with filters and inventory display
- Control logic

Example: View inventory control logic

- Name
  - `viewInventory`
- What it does:
  - Verifies that the withdrawn amount is available in the user’s account and debits the account if so
- Inputs (with type information)
  - *Page*, which set of 100 items the user is viewing
  - *Filters*, a dictionary of `{ filterName: filterValue }`
- Outputs (with type information)
  - Boolean: True if success, False if failure
- Assumptions
  - User is already authenticated

Pseudocode

- Helpful to gain insight into how components interact
- Not really code - mostly readable English with some flow control & variables

```
viewInventory(page=0, filters=None):
    translate filters into data model equivalents

    select item and quantity from data manager
    where filters match
    skip the first 100*page
    maximum 100
```

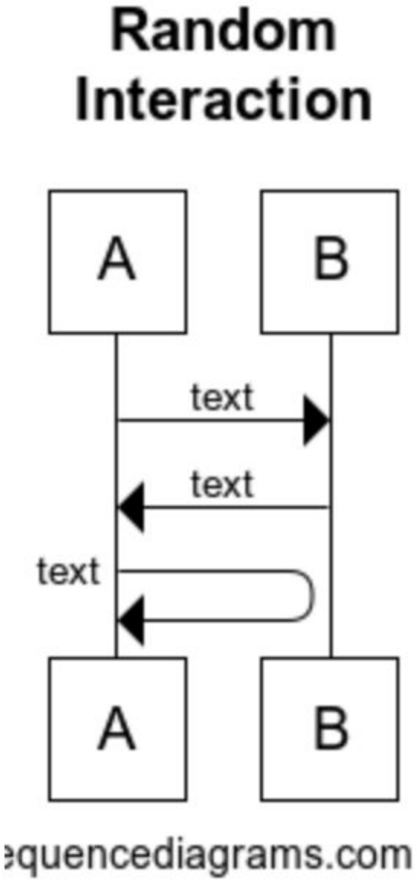
Another component!

Specifying component interactions

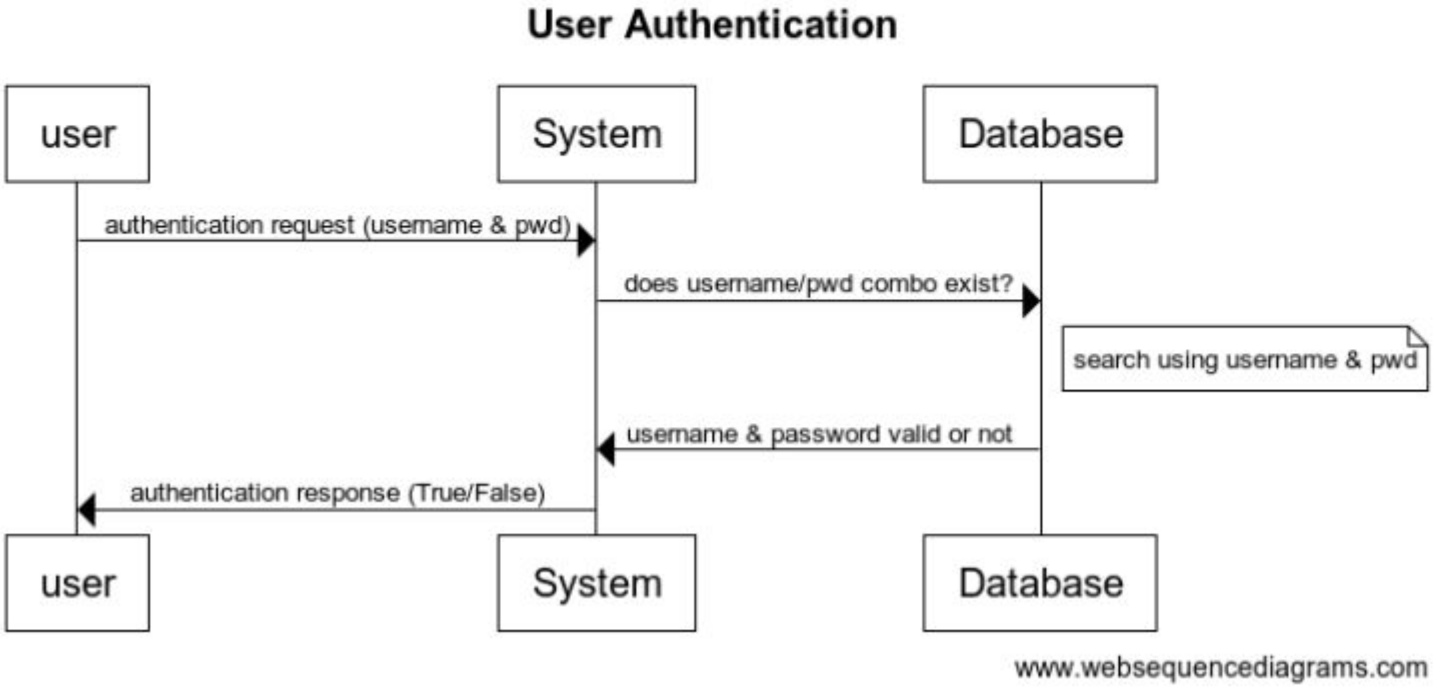
Diagrams are very helpful!

Web tool for making these:

https://www.websequencediagrams.com/



Example: user authentication component interactions



User Authentication

user

System

Database

authentication request (username & pwd)

does username/pwd combo exist?

search using username & pwd

username & password valid or not

authentication response (True/False)

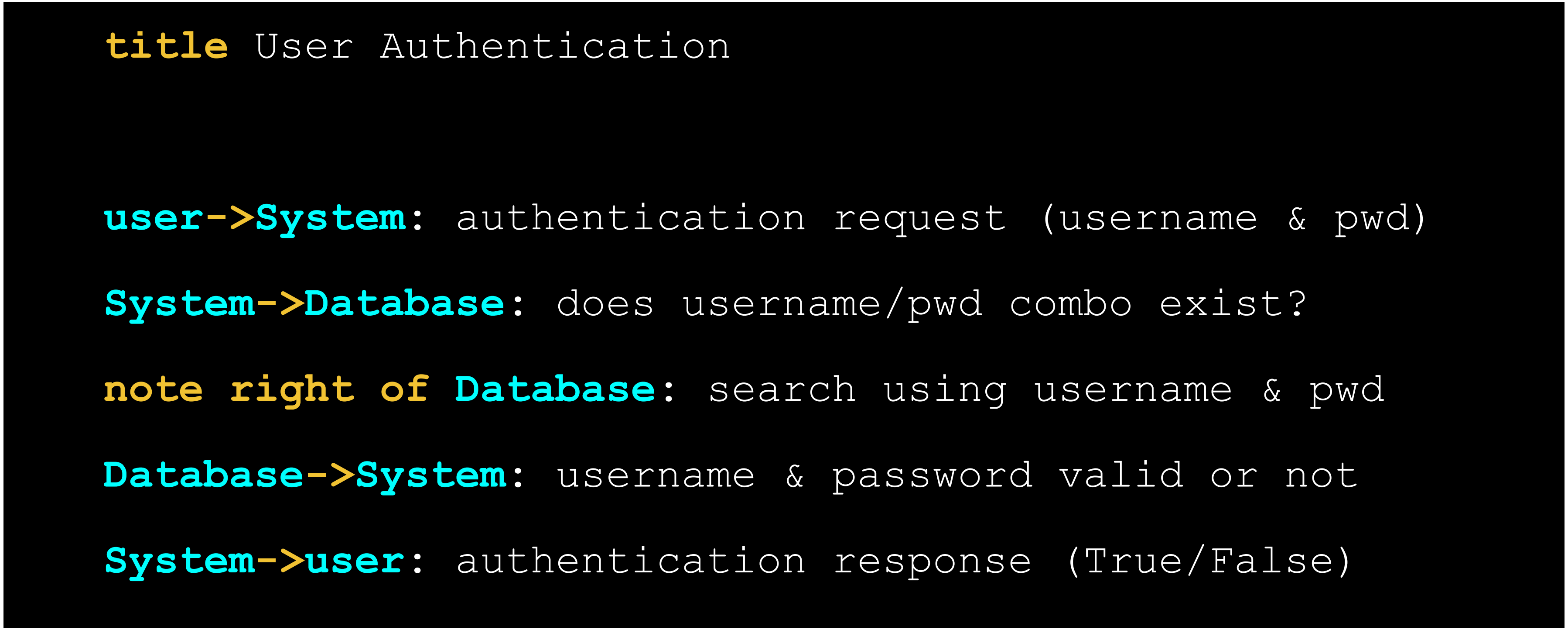
user

System

Database

www.websequencediagrams.com

Example: user authentication component interactions



```
title User Authentication


user->System: authentication request (username & pwd)

System->Database: does username/pwd combo exist?

note right of Database: search using username & pwd

Database->System: username & password valid or not

System->user: authentication response (True/False)
```

Another diagram option: draw.io

https://app.diagrams.net/

We’re not picky on the format you use!

Component Design: Summary

1. Identify components
2. Specify each component
3. Specify component interactions

Next week: common architecture patterns

From Design Onwards

Milestones

- What are you actually going to *do* and in what *order*?
- Are there any *dependencies* between components such that you must build Component A before Component B?
- What will success look like?
- This is not specific tasks - these are broad strokes

Example: Inventory Management System milestones

1. **Build account infrastructure**
   Success: account database/middle control layer exist that can access an account
2. **Build user authentication flow**
   Success: authenticate user/password by calling the Python functions directly
3. **Build user interface**
   Success: user can log onto their account with their username and password
4. **Build inventory view feature**
   Success: user can view current inventory (no filters or pages)
5. **Build inventory view extra features**
   Success: user can filter and paginate through the current inventory

…

Breaking down a milestone

Once you have milestones, you need to break them down into tasks:

- Which components do you need to implement?
- What packages do you need to incorporate?
- What tests or validation do you need to do?

Suggestion: use GitHub Issues!

Implementing a component

Recommended approach:

1. Write the *interface* for the component.
2. Write some *tests* for the component (coming up in lecture!).
3. Write the *implementation* of the component.
4. Iterate on the tests as you identify more.

Dividing up work

- Components can be a good natural division of work
- Example:
  - Melissa builds the user interface components
  - Mithali builds the machine learning model
  - Yash builds the control logic layer

This is not the only way to divide up work!

Dividing up responsibilities

Besides actual coding work, there are other responsibilities to divvy up. Everyone is encouraged to contribute to everything as they like, but dividing up who is ultimately accountable for making sure everyone does their part of the project is helpful. Common roles:

- Designers (system design, documents, communication)
  - Keep everyone accountable for documentation, design, & communication with instructors
- Developers ("devs")
  - Focus on making sure everyone makes good implementation decisions
- Testers
  - Ensure everyone keeps up with good code testing and style practices
- A project manager a.k.a. "PM"
  - Run standups & keep track of milestone progress

Project Software Design Requirements

- Functional Specification (user stories & use cases)
- Component Specification
- Milestones

More details on expected output:

https://uwdata515.github.io/projects.html

Exercise: Your Components

1. Translate your use cases into components
2. Create a specification for each component

   Potentially including pseudocode

3. Create interaction diagrams for how components interact

Use a Markdown file in the `docs` directory

Go as deep as you can!