LECTURE IX

Software Engineering and Version Control

SECTION I

Why Discuss Software Engineering?

Project Failure

I/A

Have you worked on a project that failed? (school project, CS project, etc.)

Why did the project fail?

- Missed deadline
- Poor communication
- Budgetary issues

Reasons for Project Failure

- **70% of organizations experienced project failure** (KPMG, 2023)
- The most common factors for project failure were…
  1. **Lack of clear goals** – unclear project objectives
  2. **Poor risk management**
     - Common risks are…
       - **Scope creep** – initial project objectives aren't well-defined and deliverables are slowly added
       - Poor product performance, cost, time, health and safety
  3. **Poor communication**

Reasons for Project Failure (Cont'd)

I/A

Which **factors** contributed to each of the following projects' failure?

- **Boeing 737 MAX**
  - Poor communication – pilots were not informed of the existence of the flight stabilization software, its potential to fail, and how to handle failure
  - Poor risk management (health and safety) – the lack of pitch sensor redundancy made flight stabilization more likely to fail
- **Baltimore Bridge Collapse**
  - Poor risk management (health and safety) – the bridge was not designed to withstand the impact of the vessel which struck it

Reasons for Project Failure (Cont'd)

I/A

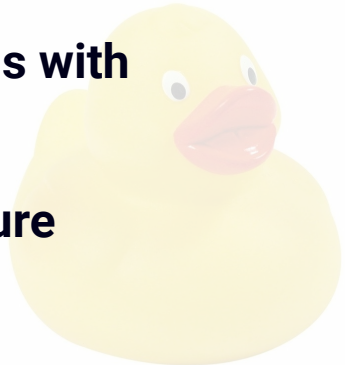
Which **factors** contributed to each of the following projects' failure?

- **California High Speed Rail**
  - Lack of clear goals – the original goal was to connect Bay Area and LA with a nonstop route
    - Detours added as political compromise; construction began in the Central Valley
  - Cost overruns – $33 bil estimate rose to $113 bil
  - Poor time estimates – not projected to be completed in this century

Source: California High Speed Rail Authority • By The New York Times

Software Engineering

- **Software engineering** is the design, development, testing, and maintenance of software applications
  - It's not about simply writing code; **it's about *how* we write code**
- We will connect the processes behind software engineering to other engineering disciplines… mechanical engineering, electrical engineering, civil, etc.
  - This will give insight into **best practices for building systems with complexity and scale**
  - There are processes to **mitigate the chances of project failure**



Software is Everywhere!

- Many interdisciplinary engineering projects involve writing programs…
  - Control algorithms, device drivers, software for managing manufacturing systems, CAD, CAM, etc.
  - Engineers often **write custom scripts** (small programs) to automate their work in software like MatLab, LabVIEW, Ansys, Solidworks, NX, and Altium
- Again, even if you don't write code, **software engineering *processes* can improve your project**

SECTION II

Defining the Problem

Problem Definition

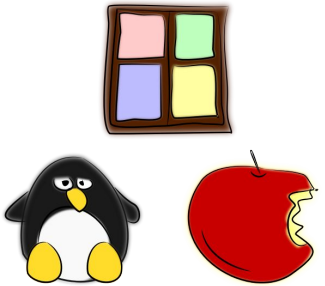
- Most projects that fail do so because of unclear goals or **poor problem definition**
- Your product, whether it's a device, a program, or a structure, **must satisfy the problem provided by a client**
- A **problem** is composed of **requirements** and **design constraints**
    - **Requirements** are statements that define what the product is supposed to do
    - **Design constraints** constrain the ways the product can be designed and implemented

Functional vs Nonfunctional Requirements

- **Functional requirements** – what the product must do
  - Ex) “The coffee maker shall make hot coffee for the user”
- **Nonfunctional requirements** – how the product does the task (how it behaves)
  - These are **often words that end with -ility**
    - Ex) usability, reliability, security, scalability
  - Defines performance requirements, security, usability, etc.
  - Ex) Usability – “The CAD software target audience is engineers.”
  - Ex) Performance – “The cleaning robot must sweep 100ft^2 in 5 minutes”

Design Constraints

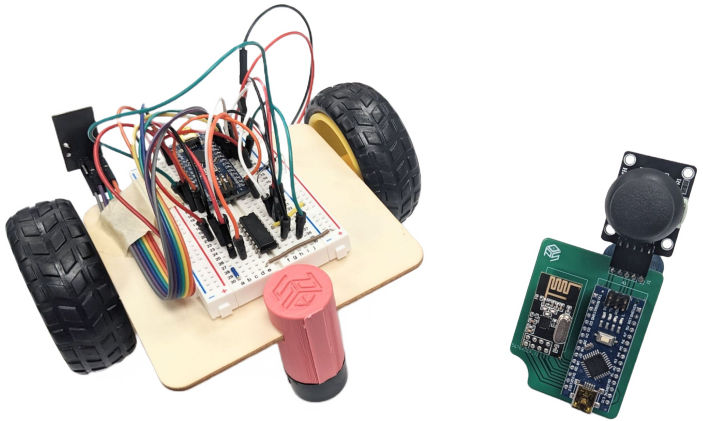
- Design constraints limit how the product can be made
  - Platform requirements – Is the product meant for phone or computer? Mac or Windows?
  - Cost requirements – How much money can be spent on the project's development?
  - Time requirements – When does the client need the project completed by?



Example Requirements

Determine whether each of the requirements from the 2024 OPS Capstone Project are **functional or nonfunctional requirements or both**. Identify the **design constraints** as well.

2024 Capstone Project Link



SECTION II

Why Methodologies Matter

House Building Analogy

Let’s say a client hires you to build a house.
What are the **steps to build** the house?

- **Ask the client for their requirements** (Requirements)
  - How many stories, how many bedrooms/bathrooms?
- **Draw up a blueprint**; **create a bill of materials** (Design)
- **Build the house** (Implementation)
- **Test the house** for functional plumbing, working electricity etc. (Verification)
- **Give the house to the client** (Deployment)

House Building Analogy (Cont'd)

- In this house-building analogy, you need to do a lot of things for this project to go right…
  - **Planning** - set requirements so the client is happy, make blueprints, source materials
  - **Documentation** – write blueprints, get permits, track expense sheets, make construction schedule
  - **Versioning** - track iterations of the blueprints or bill of materials
- **The same is needed for any other project**… software, mechanical, whatever!

Software Development Methodologies

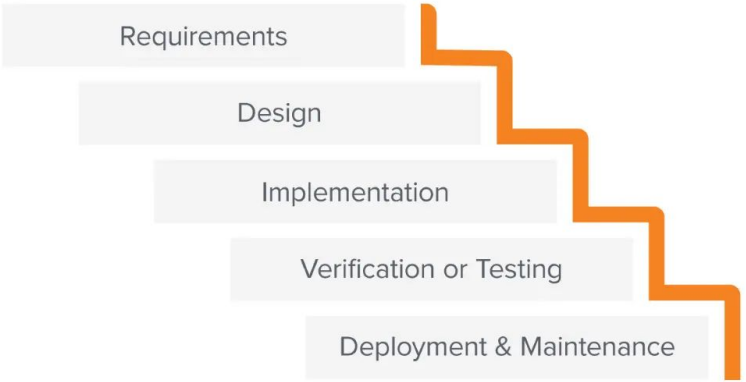
- A **software development methodology** is a **series of processes** used in software development…
  - Ex) Requirements specification, design, implementation, verification, deployment
  - These processes are facilitated by **planning**, **documentation**, and **version control**
- Applying a method to your project…
  - Improves team communication and collaboration
  - Reduces errors and rework

SECTION III

Software Development Methodologies

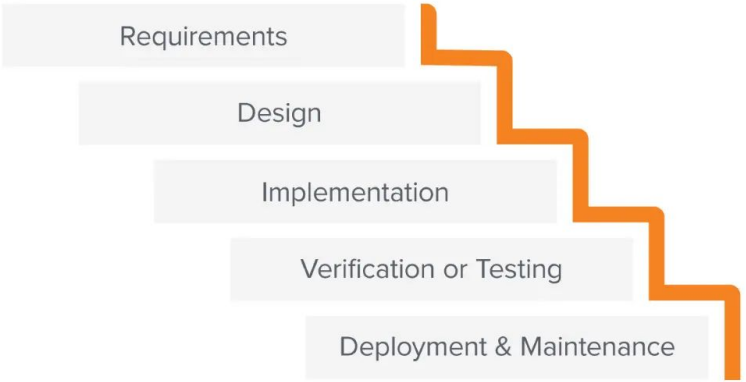
Waterfall Methodology

- The **Waterfall** model is a sequential development process that flows through project phases:
  - Analysis, design, development, and testing
  - Each phase is completed before the next phase
- Notice how **the phases are universal to any engineering project**, not just software



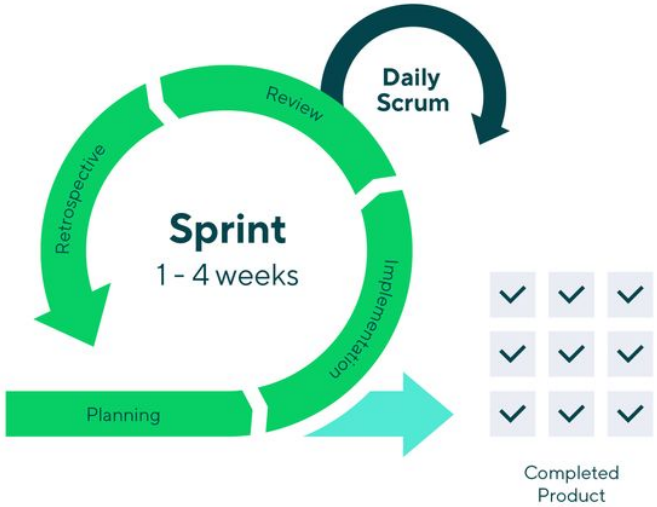
Waterfall Methodology (Cont'd)

- The documentation, requirements, and constraints are fleshed out very early on
    - Since research is done at the beginning, **time and cost estimates are more accurate**
- There is no backtracking - **parameters are harder to change** after they're set, unlike Agile



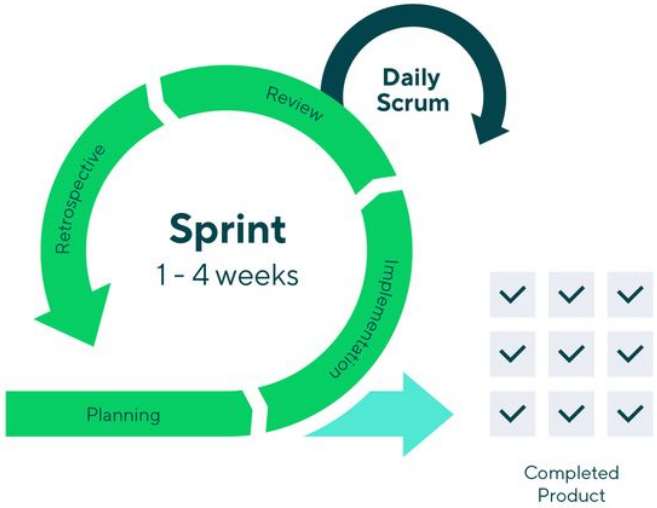
Agile Methodology

- **Agile** is an approach that reduces large projects into smaller tasks
    - The tasks are completed in **iterations**
- Each requirement is represented as a **user story** which is written from the user's perspective
- During a **sprint** – a 1–4 week iteration – developers work on tasks determined in an initial **sprint planning meeting**



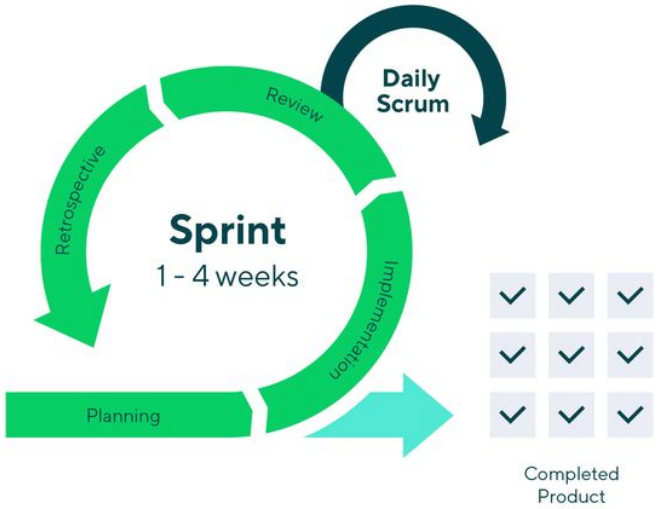
Agile Methodology (Cont'd)

- Developers hold a short, 10-minute **stand-up** meeting or **daily scrum** to discuss task progress
- Each sprint ends with…
  - **New features added** to the product
    - In the Waterfall model, the entire implementation is released at once
  - A **retrospective** meeting to evaluate areas of team and product improvement



Agile Methodology (Cont'd)

- Agile encourages **responsiveness** and **frequent collaboration** through repeated planning and regular meetings
    - If team members are not strong communicators, they may struggle to use Agile
- It is **easy for a team to veer off-course** when requirements (user stories) can be reassessed between sprints



SECTION IV

Version Control Systems

Version Control

- Version control tools save lives!
  - They **record changes to files** over time
    - Keeping track of different revisions of code, circuit designs, CAD model, etc.
  - You can **revert back to previous versions** if necessary

- We will look at two popular but different version control systems…
  - **Git** and **Perforce**

git

PERFORCE

Git

- **Git** is a free, open source, and *distributed* version control system 
  - A project is downloaded onto your local computer as a **repository**
  - You, the developer, makes changes to the repository, which they record as **commits** on your local repository
  - New versions of the repository are created as **branches**
- What does it mean to be distributed?
  - In a **distributed** version control system, multiple copies of the repository exist on different computers
  - A developer team must eventually merge all their code to one **main branch**

Perforce

- **Perforce** is a licensed, enterprise-oriented, and *centralized* version control system
  - Developers commit changes to a **central server** with one main copy of the project
    - This one project copy is the **single source of truth**
    - This creates less of a hassle when looking for the true project version
  - Developers **checkout** individual files and lock them to prevent others from making changes and creating merge conflicts
  - Nothing is local – all changes are submitted to the server, which can be **slower than a distributed version control system when changes are small**

Choosing a Version Control System

- **Git** is by far the **most widely used version control system** (VCS)
  - Why? It's free, versatile, reliable, and also… very, very free
  - Developers can use it for free and remotely with services like GitHub
- Large files and code bases
  - **Git is not designed to handle large file storage** without additional plugins
    - There is natively a 2 gb limit on commits
  - **Git merge conflicts are really nasty in large, active repositories**
    - Enterprise companies with many files may prefer to use Perforce for this reason

SECTION V

Final Thoughts

Being a Better Engineer

- Bring intent to your next project by adopting a methodology
  - Why do projects fail? Unclear objectives, poor risk management, and bad communication
  - Successful projects are **well-planned** and **well-documented**
- Try using Git with your future projects
  - View the Version Control with Git Workshop material for more details
  - **Git can be used for more than just code!** Images, documents, and some CAD files can be archived with Git

FAIR USE DISCLAIMER

Copyright Disclaimer under section 107 of the Copyright Act 1976, allowance is made for “fair use” for purposes such as criticism, comment, news reporting, teaching, scholarship, education and research.

Fair use is a use permitted by copyright statute that might otherwise be infringing.

Non-profit, educational or personal use tips the balance in favor of fair use.