# Software Projects

CS 346 Application Development

https://student.cs.uwaterloo.ca/~cs346

# Contents

# Introduction

# Project Styles

A **software project** is a specialized project which results in the delivery of a software product e.g., a new release of macOS.

Software production in the 1960s and 70s focused on bespoke projects: building custom software for a specific customer.

- Customers would define requirements and then engage an engineering firm to deliver their software. These types of projects still exist e.g., banking.

Software production since the 1980s has mostly shifted building generic software products that are useful to a range of customers (more profit).

- Application software fits into this category and can include large-scale business systems (e.g. MS Excel), personal products (e.g. Discord) or games (e.g. Flappy Bird).

# Project Styles

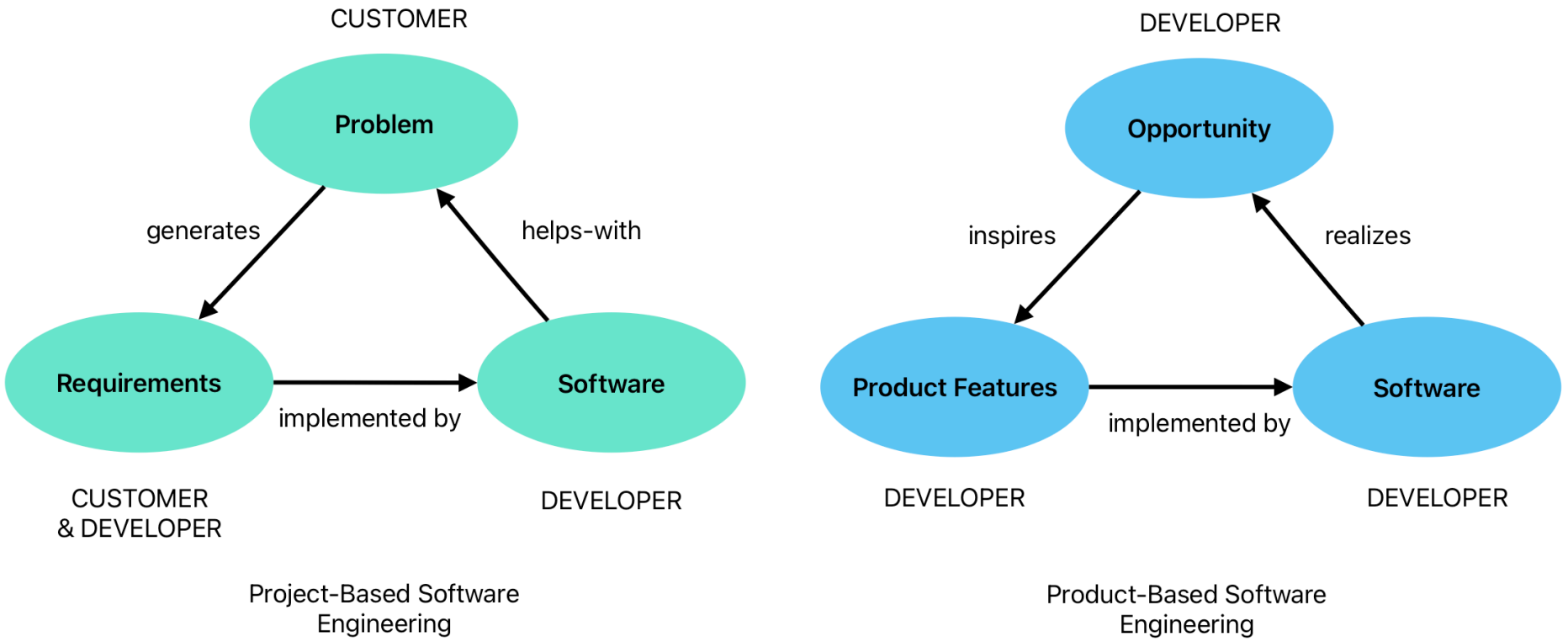



Figure 1: In modern product development, most decisions are made by the development organization on behalf of a customer. [1].

# Project Styles

How is a software project different from any other project?

In some ways, a software project is similar to any other project: it includes a set of steps that we will follow and track, and goals that we want to achieve.

However, software as a *product* has its own peculiarities:

- The cost of manufacturing software is largely the cost of hiring and empowering people to do the work.
- There are no “raw materials” like you have in manufacturing e.g., compare building a calculator application to building a car.
- Once produced, each additional “unit” of software that you sell is zero-cost. All of the cost is up-front, incurred in design and development.

> **Tip**
>
> We'll return to the topic of software projects very soon.

# Software Activities

The general project activities that we identified earlier are still applicable. In software projects, we usually order them in a circle, to demonstrate that we're in a continual cysle of planning, designing, implementation and delivery to customers. We often refer to this as the Software Development Lifecycle:

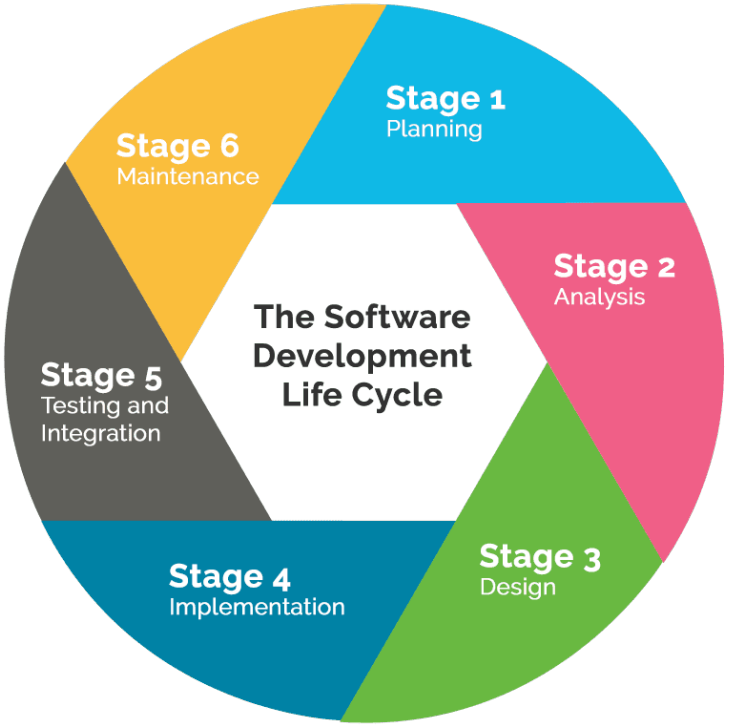
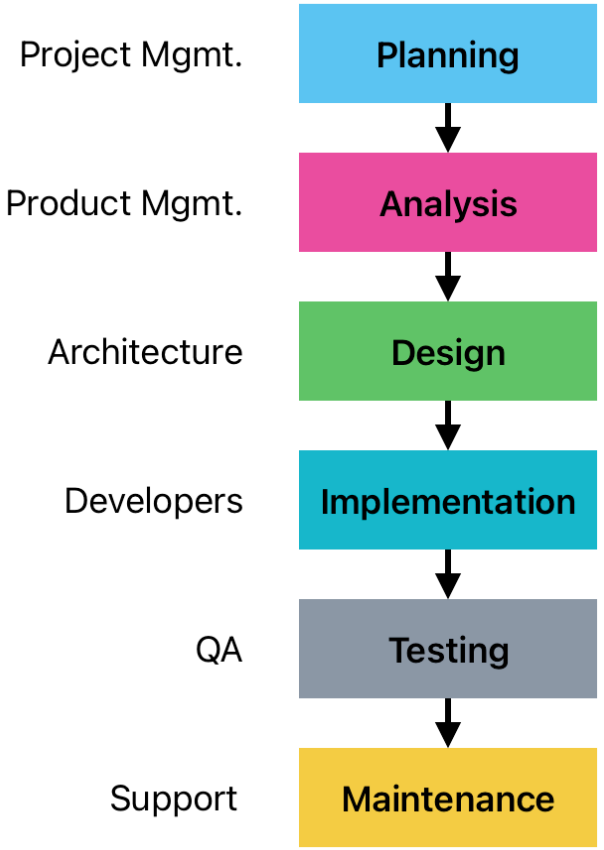



Figure 2: The SDLC. Image attributed to https://triniinxisle.com

# Software Activities

This approach of organising software products has been around for decades. It's also dubbed the `Waterfall Model`, due to the top-down flow of information in a project.

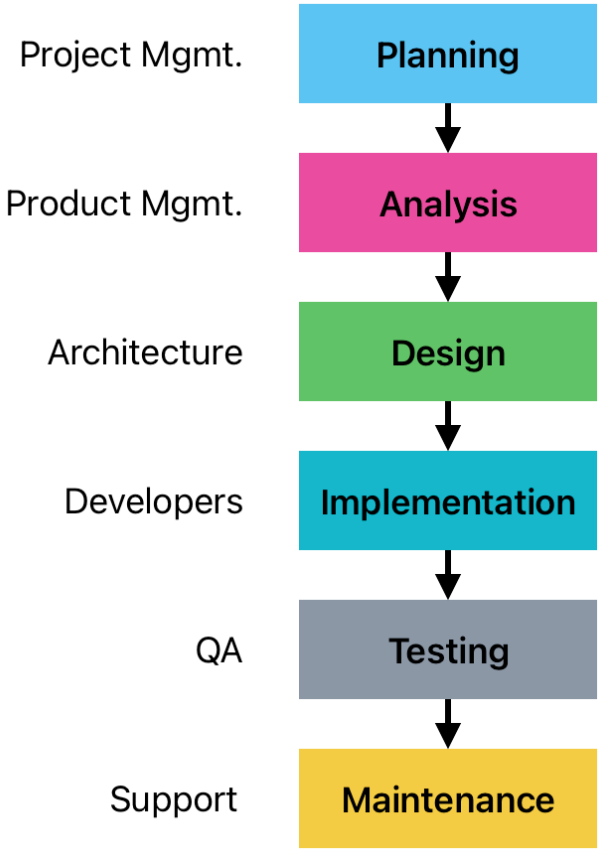
- Each step is "owned" and managed by a separate person or group.
- Steps need to be performed in order, and a step must be completed before the next step can be started.
- There is often gatekeeping enforced e.g., requirements approval before design.
- This project structure discourages collaboration across teams.



## Why Not Waterfall?

This model doesn't work very well. Why?

- Customers may ask for changed mid-design. You need to be able to address this.
- Your understanding of a problem will evolve! You may need to step back and revise decisions made earlier in the process e.g., design revisions.
- Compartmentalizing teams like this discourages collaboration. Diverse teams make better edcisions e.g., QA will have insights on what designs are testable.



# Agile Models

# The Agile Manifesto


Figure 3: Released in 2001, the Agile Manifesto attempted to reframe how we manage software projects [2]. It inspired a large-scale shift in software development practices.

# Agile Methods

Early, plan-driven development had evolved to support the engineering of large, long-lifetime systems (such as aircraft control systems).

- This approach is based on controlled and rigorous software development processes that include detailed project planning, requirements specification and analysis and system modelling.
- Plan-driven development involves significant overhead, and it's slow.

Agile methods were developed in the 1990s to become more efficient.

- Teams may be geographically dispersed and work on the system for years.
- Development focuses on applying our lifecycle to features over products.
  - Get feedback more frequently and make adjustments as-needed.
  - Focus on working software and iterating on implementation.

> **Tip**
>
> Agile methods focus on customer feedback and quick iteration cycles.

# Agile Methods

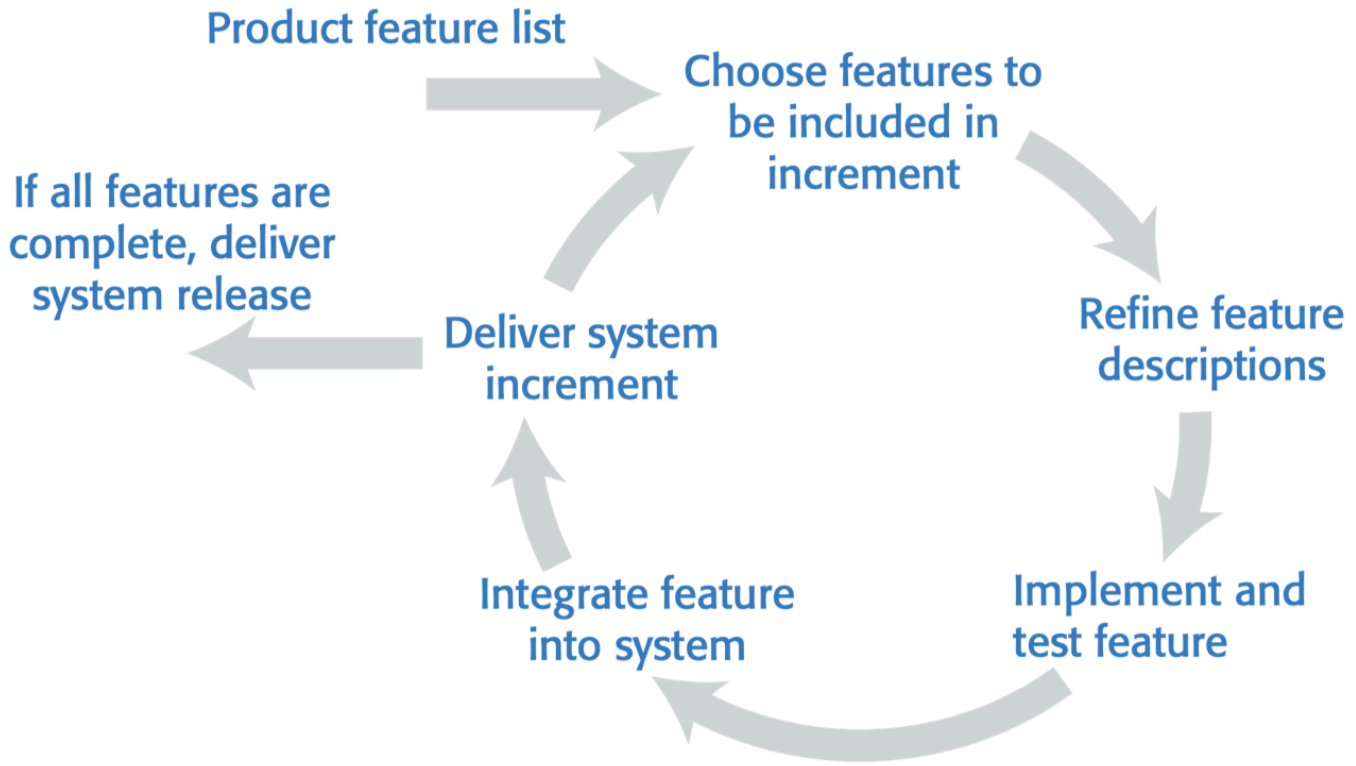



Figure 4: Incremental development doesn't attempt to design and build the entire system, but rather focuses on a cycle of continuous feedback and system increments. Diagram from Sommerville, Engineering Software Products, 2019.

# Extreme Programming (XP)

One of the most influential process models is Extreme Programming (XP), designed by Kent Beck in the late 1990s [3].

XP focuses on quick, responsive software development practices that reduce the distance between developer and user.

- Iterative planning continually reassesses candidate features. XP does this to account for customer and market changes.
- The team plans around a feature, but recognizes that the plan is fluid and may change.
- Goal of getting working software in front of a user ASAP.

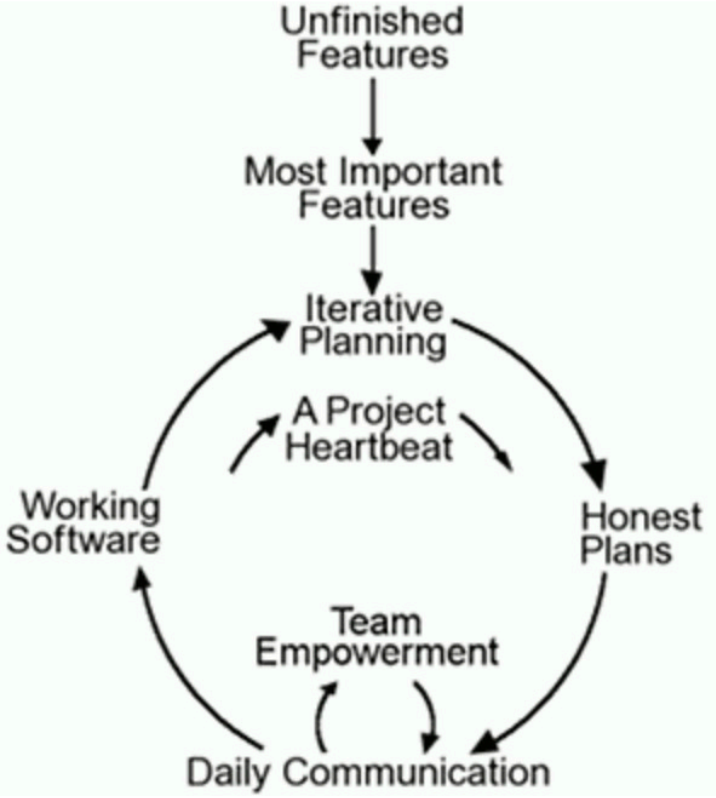



Figure 5: XP introduced the modern iterative development cycle.

# Extreme Programming (XP)

## Practices Commonly Adopted

1. **Incremental planning/user stories**. There is no ‘grand plan’ for the system. What needs to be implemented (the requirements) in each increment are established by the team and customer. Requirements are written as user stories, and are priority is determined by the time available and their relative importance.
2. **Test-driven development**. Instead of writing code then tests for that code, developers write the tests first. This helps clarify what the code should do and ensures that there is always a ‘tested’ version of the code available. An automated unit test framework is used to run the tests after every change.
3. **Continuous integration**. As soon as the work on a task is complete, it is integrated into the whole system. All unit tests from all developers are run automatically and must be successful before the new version of the system is accepted.

# Extreme Programming (XP)

4. **Refactoring**. Refactoring means improving the structure, readability, efficiency and security of a program. All developers are expected to refactor the code as soon as potential code improvements are found. This keeps the code simple and maintainable.
5. **Small releases**. The minimal useful set of functionality that provides value is developed first. Subsequent releases of the system add functionality incrementally. Small, well-tested releases provide more opportunities for feedback, and reduce the cost of acting on that feedback.

# Scrum

Software company managers need to understand how much it costs to develop a software product, how long it will take and when the product can be brought to market.

- Plan-driven development provides this information through long-term development plans that identify deliverables - items the team will deliver and when these will be delivered.
- Plans always change so anything apart from short-term plans are unreliable.

Scrum provides a framework for agile project organization [4].

- It is designed around short-term planning activities.
- The assumption is that requirements and plans will change during a project!
- Unlike XP, it does not mandate any specific technical practices.

# Scrum

## Key Concepts

Timeboxed iteration (aka “sprint”)

- Work is done in fixed-length iterations (usually 2-4 weeks), called `sprints`. Each sprint starts with planning activities, and ends with working software and a demo to a customer.
- A project will consist of a series of `sprints`, run end-to-end. The project ends when the stakeholders choose.

Self-organizing teams

- Teams make their own decisions and work by discussing issues and making decisions by consensus. No single person is “in charge”.
- Unfinished work is tracked in a `backlog`. The team decides what to work on at the start of each sprint. They are in control of what is implemented (based on feedback from stakeholders and users and other factors).

Customer feedback

- You engage the customer for informal feedback (when possible) and formal demonstrations of work completed during an iteration.

# Scrum

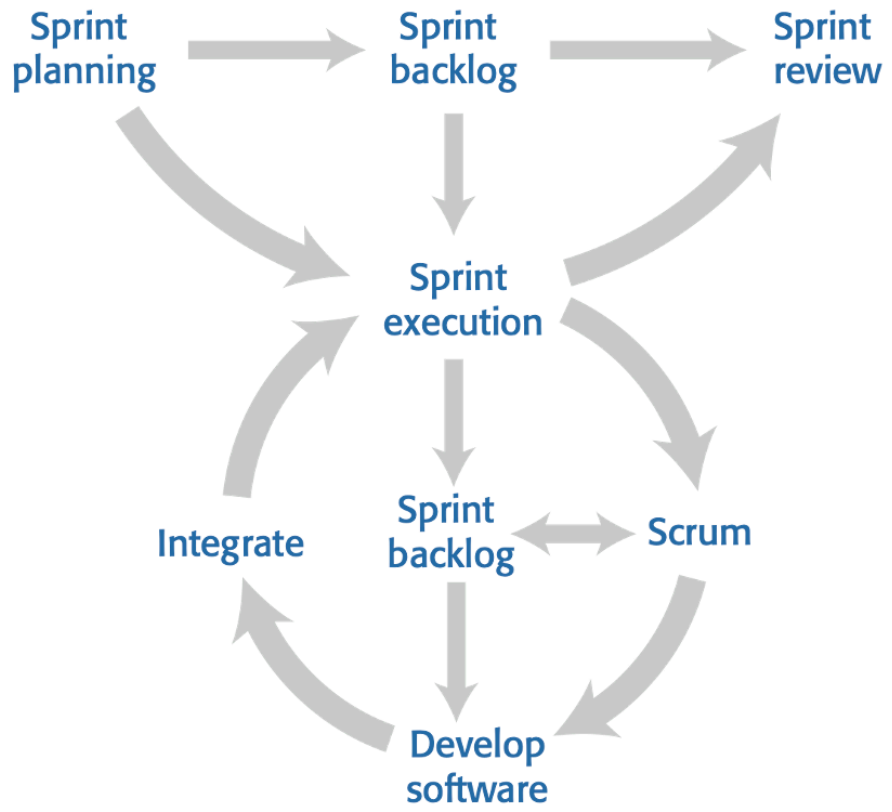



Figure 6: Sprints consist of planning activities, execution and delivery of software. Diagram from Sommerville, Engineering Software Products, 2019.

# Being Agile

# Principles

How to be successful as an Agile developer.

1. Be flexibile in your planning and expectations. Things *will* change as the project progresses.
2. Focus on quality, and don't be afraid to defer something if you don't finish it on-time. Never push through a poor design or untested feature.
3. Ask for feedback. Trust what users are telling, even over your own instincts.
3. Write automated tests for everything. You can't trust code that is untested, and you know you won't test unless it's automated.
4. Team communication is critical. Meet regularly and chat online frequently.
5. Be a great team member. Help one another out as much as you can. Tasks are less daunting when you can work through them with someone.

# Workflow

We will adopt workflow practices from Scrum and software practices from XP.

- The first couple of weeks will be spent brainstorming & choosing a project.
- You will also flesh out requirements, including a backlog of work items.
- You only start coding after your requirements and tasks are approved.

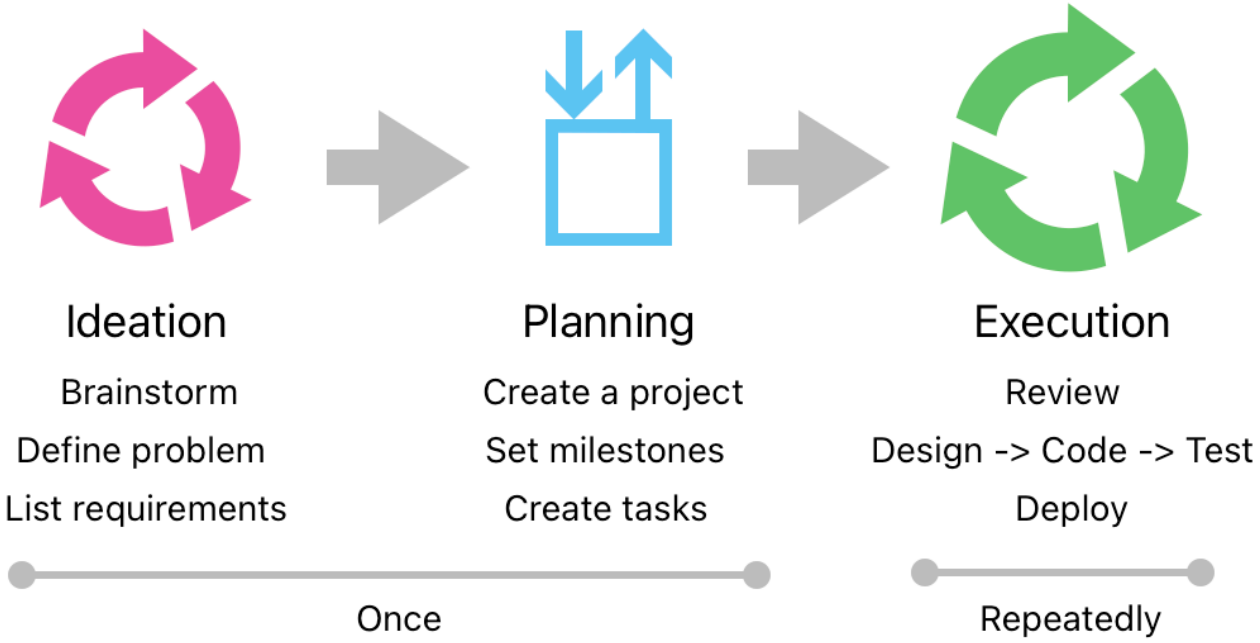



Figure 7: The first few steps set us up for iterative development starting in week 04.

## Phase 1: Ideation

This is the brainstorming phase, where you decide what to build.

- Brainstorm by talking with your teammates to see if you have any common interests that you want to explore. Identify personal goals if you have any e.g., front-end development, or database design.
- Once you've picked a project, then focus on a problem statement - what common problem you will solve for which users.
- Work through the process of determining your requirements, including a Problem Statement, Personas, Scenarios & User Stories.

## Phase 2: Planning

The planning stage where you identify milestones and sketch out a plan of when you will address significant features.

- Start by creating a project in GitLab, following instructions on the website.
- Determine your milestones and add them to your project.
- Add tasks for the features you plan to implement (minimal details at this time).
- Take a first-cut at assigning tasks to your milestones, and revise your goals as necessary.

## Phase 3: Execution

The execution phase comprises most of your project effort. Unlike the other phases, this one repeats on a three week cycle:

- Week 1: Starts with a planning meeting to review your progress and decide (as a team) what to include in scope. Work is assigned and team members start working on their features.
- Week 2: Work continues. The team meets periodically to ensure everyone is on-track. Adjustments can be made.

Your output should be:

- Tasks in GitLab updated or closed for each feature that you worked on.
- A feature branch that was created for this work; merged back to main.
- Revised source code, with passing unit tests.

# Final Thoughts

We'll revisit this when we talk about coding with AI. That will introduce a few more artifacts!

# Bibliography


[1] I. Sommerville, *Engineering Software Products - An Introduction to Modern Software Engineering*. London, UK: Pearson, 2021. [Online]. Available: https://iansommerville.com/engineering-software-products

[2] Various Authors, “Manifesto for Agile Software Development.” [Online]. Available: http://agilemanifesto.org/

[3] K. Beck and C. Andres, *Extreme Programming Explained*, 2nd ed. Boston, USA: Addison-Wesley Professional, 2004. [Online]. Available: https://www.amazon.ca/Extreme-Programming-Explained-Embrace-Change-ebook/dp/B00N1ZN6C0

[4] K. Schwaber and J. Sutherland, *The Scrum Guide*. 2020. [Online]. Available: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf