Software Projects

CS 430 Applications Software Engineering (Development)

https://student.cs.uwaterloo.ca/~cs430

Contents

Introduction

Software Projects

A **software project** is a specialized type of project, focused on the delivery of software 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 to building generic software products that are useful to a range of customers.

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

Software Projects

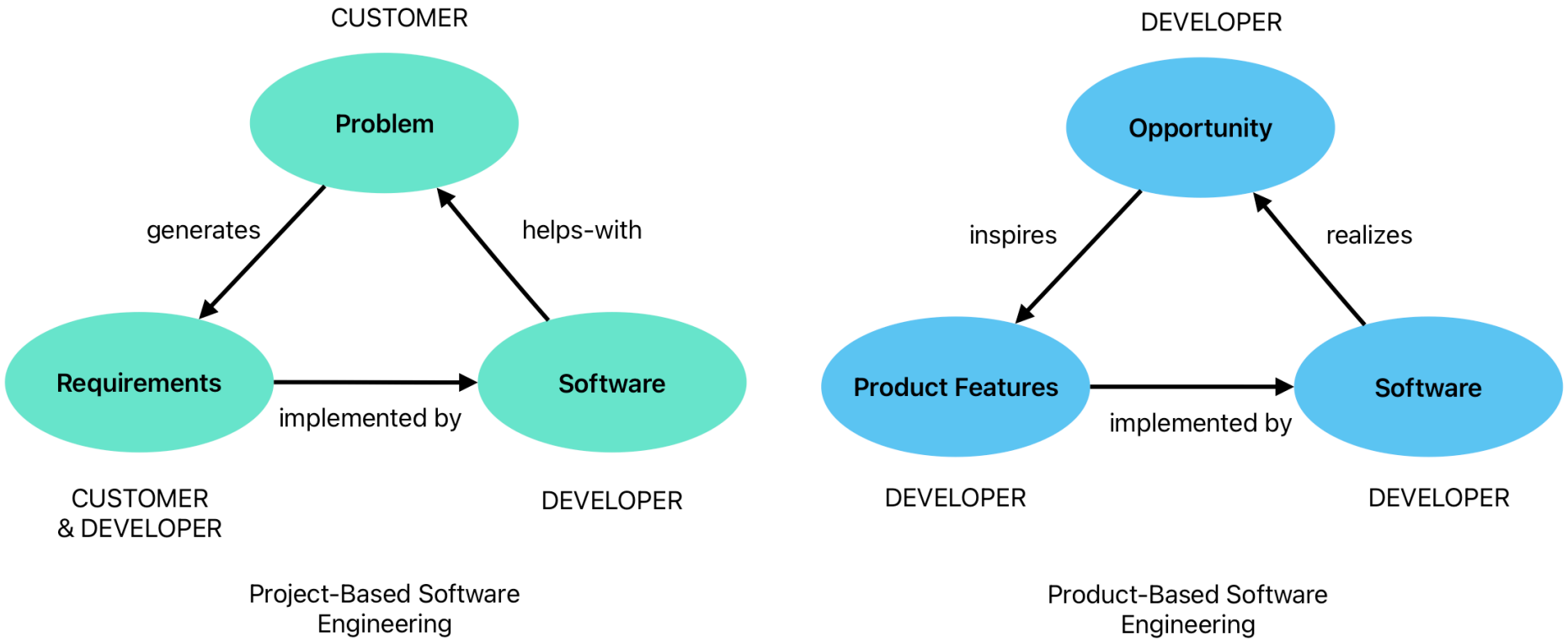

Figure 1: In modern product development, most decisions are made by the development organization on behalf of a customer. [1].

Software Projects

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.
  - Once developed, software costs practically nothing to make/distribute.
- Software projects are never naturally “finished”.
  - Maintenance can continue for as long as you have customers.
  - Software production ends when it stops being useful/profitable.

Software Activities

For these reasons, software activities are generally drawn in a circle (since you end a project, and then start planning the next iteration of your software).

We often refer to this as the Software Development Lifecycle.

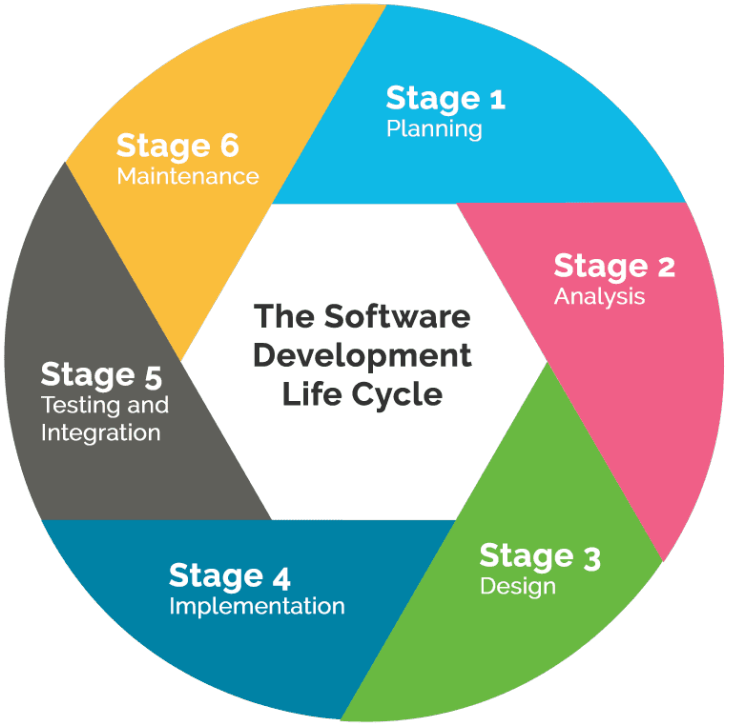

Figure 2: The SDLC. Image attributed to https://triniinxisle.com

Software Activities

This approach of organising software products has also been dubbed the `Waterfall Model`, since it assumes that each step flows into the next.

- Steps need to be performed and completed in order. A project can only progress forward.
- Each step has a distinct “owner” that manages that step.
- There is often gatekeeping enforced as you proceed e.g., `analysis` need to be approved before `design` starts.
- This structure discourages collaboration between levels.

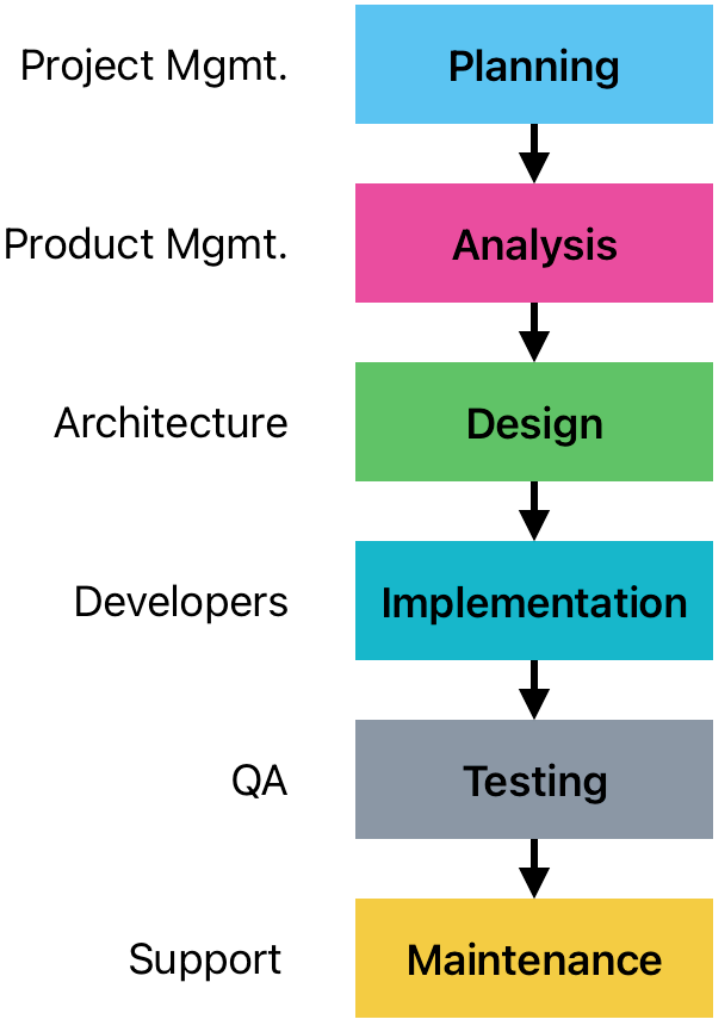

Figure 3: SDLC unrolled.

Software Activities

What's wrong with this model?

- Sometimes you need to revisit previous decisions, which this model discourages
  - e.g., when testing uncovers a design issue that should be revisited.
- Many decisions should have involvement from different people in the organization, which this model discourages
  - e.g., your ability to maintain software can depend on design and implementation decisions.

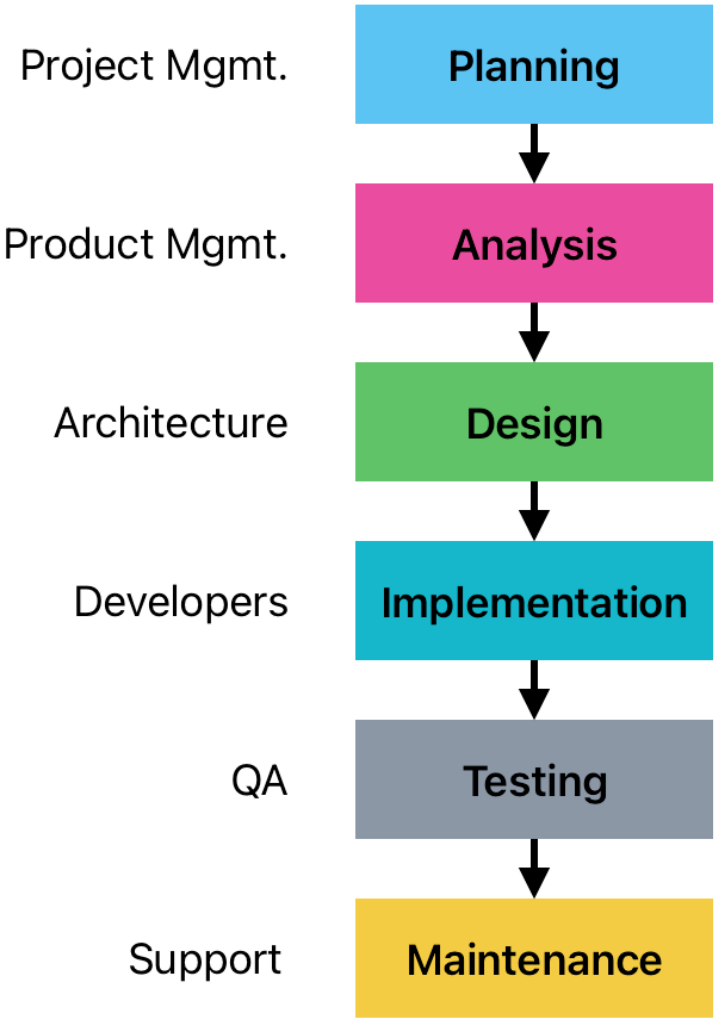

Figure 4: SDLC unrolled.

Agile Models

The Agile Manifesto

Figure 5: 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 Principles

Traditional plan-driven development is based on a rigorous software development processes, with significant overhead. It was rigid, slow, and didn't adapt well to change at a time when the market was rapidly changing.

Agile principles were a reaction this problem, and led to practices that supported faster and more responsive software development. Key Agile principles include:

- Collaboration with customers is critical to understand their context. Requirements should fall naturally out of user discussions.
- Design and implementation should be done incrementally. Work with your users to get early feedback and use it to iterate on your product.
- Expect requirements to change as you progress. Your ability to respond to changing conditions is fundamental to building usable and effective systems.
- Teams should include all of the necessary project roles e.g., designers, architects, developers, technical writers, testers.
- Decisions should be made by the project team.

Agile Principles

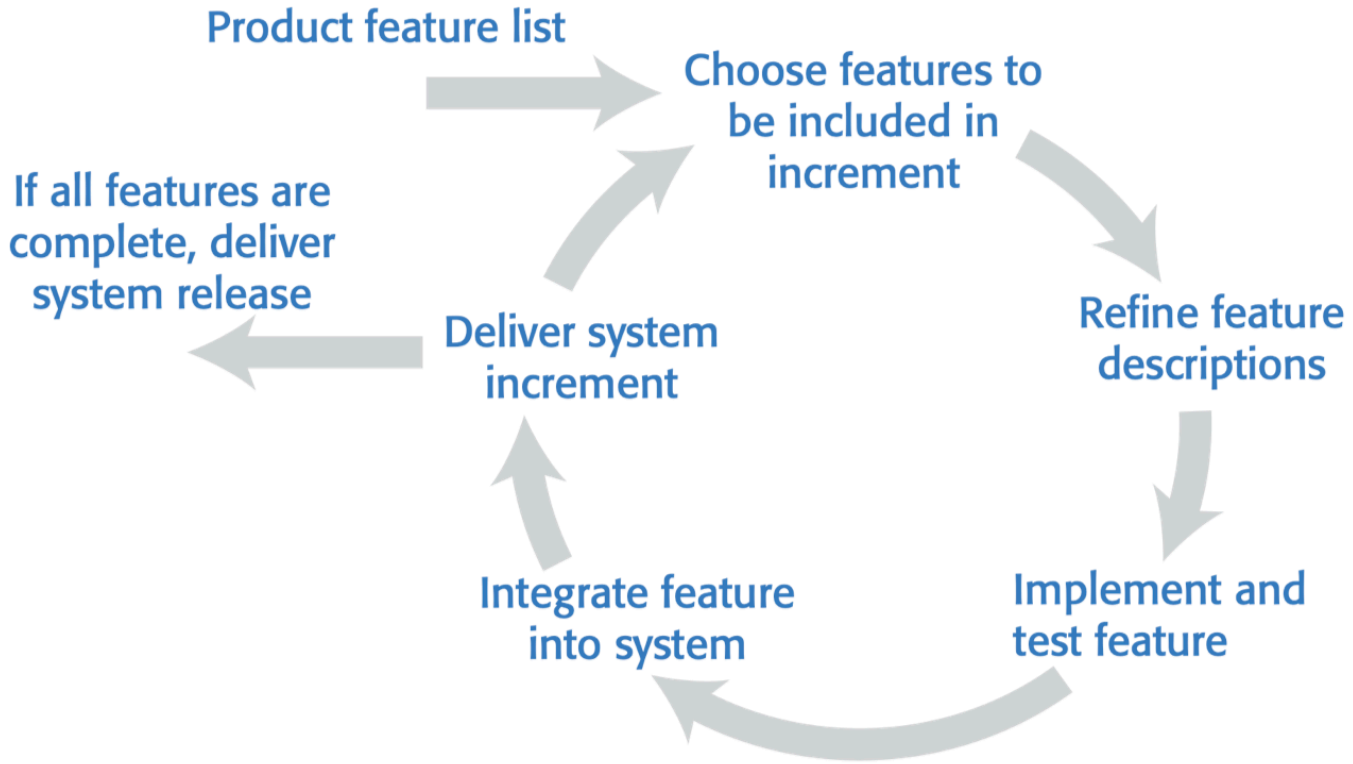

Figure 6: Incremental development doesn't attempt to design and build the entire system, but rather focuses on a cycle of continuous feedback and system increments. Diagram from Sommerville, Engineering Software Products, 2019.

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.

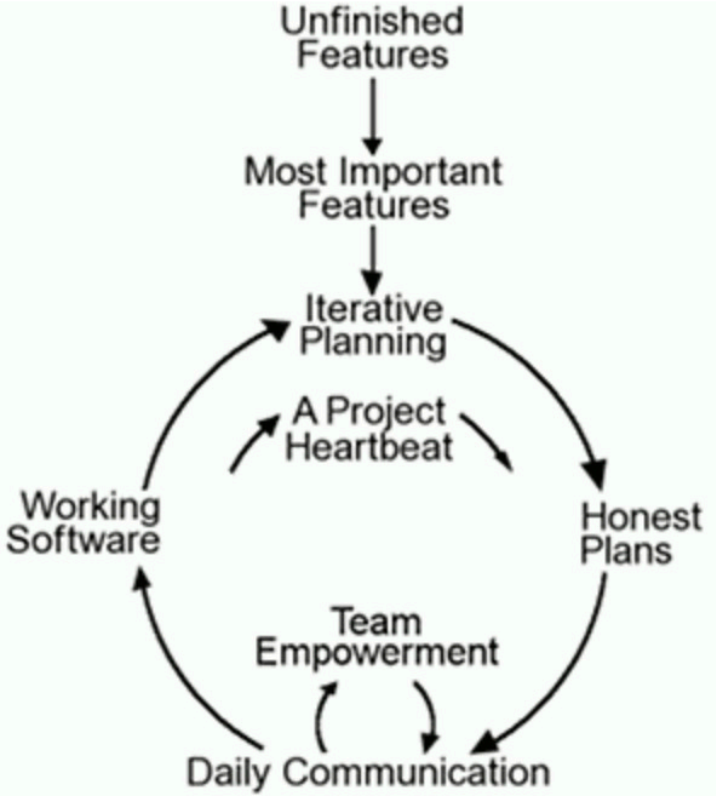

Figure 7: XP introduced the modern iterative development cycle.

Extreme Programming (XP)

Practices Commonly Adopted

1. **Incremental planning/user stories**. There is no ‘grand plan’ for the system, but it is built up over time based on discussions between the team and customer. Initial requirements are written as user stories, and features are delivered based on the time, resources 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. **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.

Extreme Programming (XP)

4. **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.
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.

> **Note**
>
> XP focuses on development practices that lead to higher-quality software. Beck's focus was on building motivated and empowered teams.

Scrum

Scrum focuses on tracking project progress [4].

- All work is “roughly” defined up-front, and logged into a Product Backlog.
- The project consists of regular iterations, called Sprints.
- The team decides what work to do in each sprint, completes it, and then shows a demo to users for feedback.

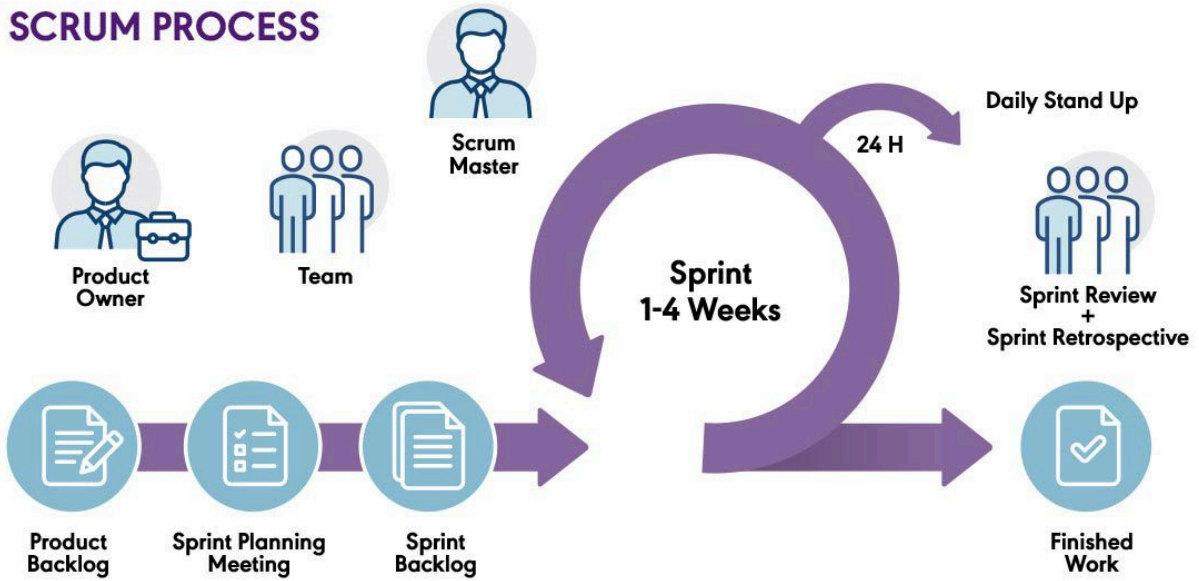

Figure 8: Scrum focuses on task management and the delivery pipeline.
Diagram courtesy of pm-partners

Scrum

Key Concepts

Sprint planning

- Kickoff meeting where the team decides what to include.
- Work items are moved to the Sprint Backlog, and may be assigned.
- ALL scope decisions are made as a team during this meeting.

Daily scrum

- Team members work and track their assigned work each day.
- Teams meet daily, usually at the start of the day. The meeting is run by a Scrum Master who runs the meeting and helps keep it moving.
- Each team member answers three questions:
  - What did you do yesterday?
  - What are you doing today?
  - Do you have any roadblocks?

Sprint review

- Each sprint ends with a demo of working software to a user/customer.
- The purpose is to get feedback! Take notes.

Being Agile

What Will We Do?

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 documented, and the team has decided what to prioritize.

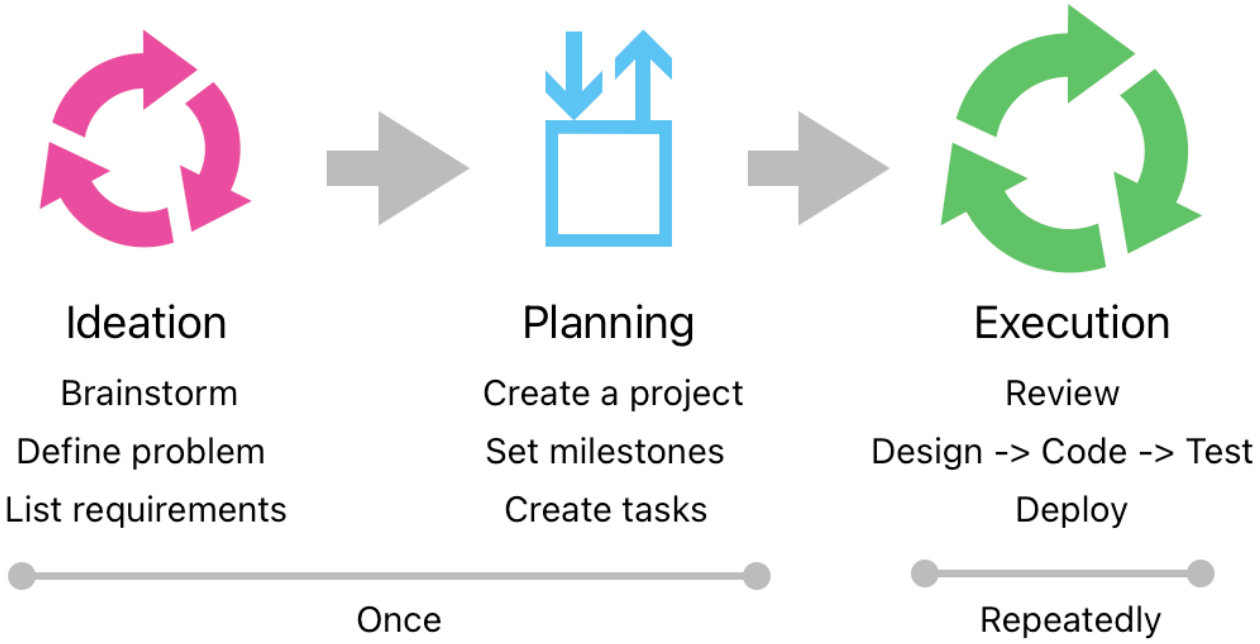

Figure 9: The first few steps set us up for iterative development starting in week 04.

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.

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