Software Projects

CS 346 Application Development

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

Contents

Introduction

What is a Project?

A **project** is a planned activity, following a set of defined steps, which results in some desired outcome e.g., building a fence, baking a birthday cake.

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

Projects describe scope (what we're building), time (how long it will take) and budget (cost). These are the major factors we need to consider for any project.

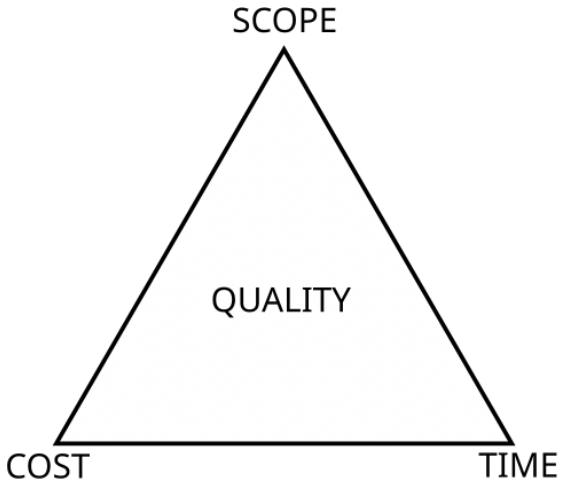


Figure 1: This is the project management triangle, which suggests that you can trade-off cost, time and scope e.g., reducing time required, but only by increasing cost or reducing scope.

Software Projects

Software production in the 1960s and 70s focused on bespoke projects: 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).

Software Projects

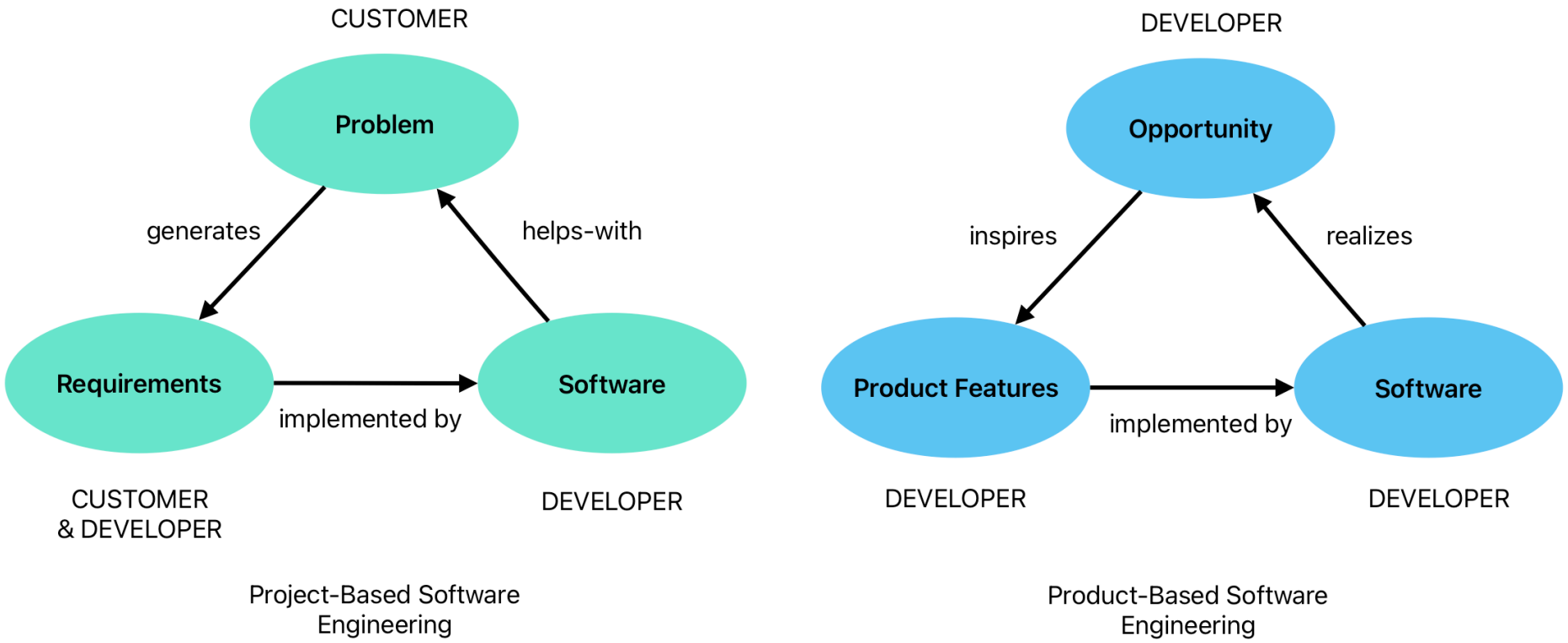


Figure 2: In modern product development, most decisions are made by the development organization on behalf of a customer. [1].

Project Steps

Projects are usually presented as a set of steps, where each step must be completed before the next one can start.

1. **Planning**: Determining up-front costs.
2. **Requirements**: What do we want to build? Who will buy it?
3. **Design**: How do we build it? What constraints do we have?
4. **Implementation**: Building it to meet project and design goals.
5. **Testing**: Ensuring that it works as expected before we sell it.
6. **Deployment**: Delivering to customers and getting paid.

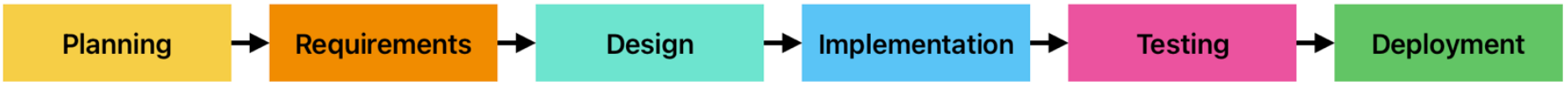
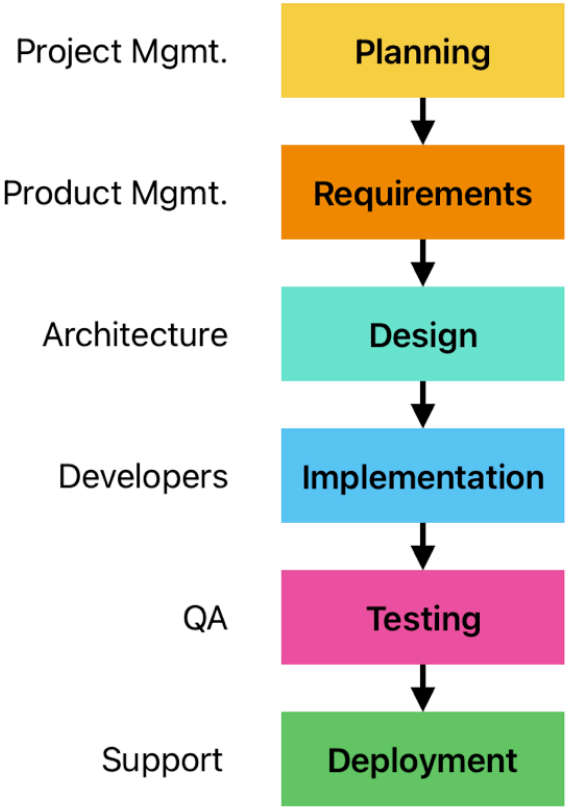


Figure 3: Following steps is supposed to lead to consistency in delivering a project.

Waterfall Model

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

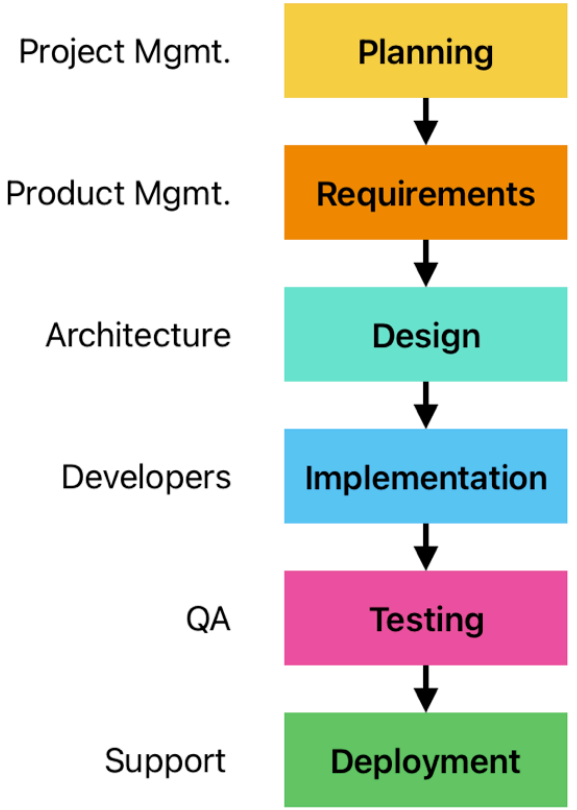
- Each step is "owned" and managed by a separate person or group.
- Steps need to be performed in order, and a step must be completed before the next step can be started.
- There is often gatekeeping enforced e.g., requirements approval before design.
- This project structure discourages collaboration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

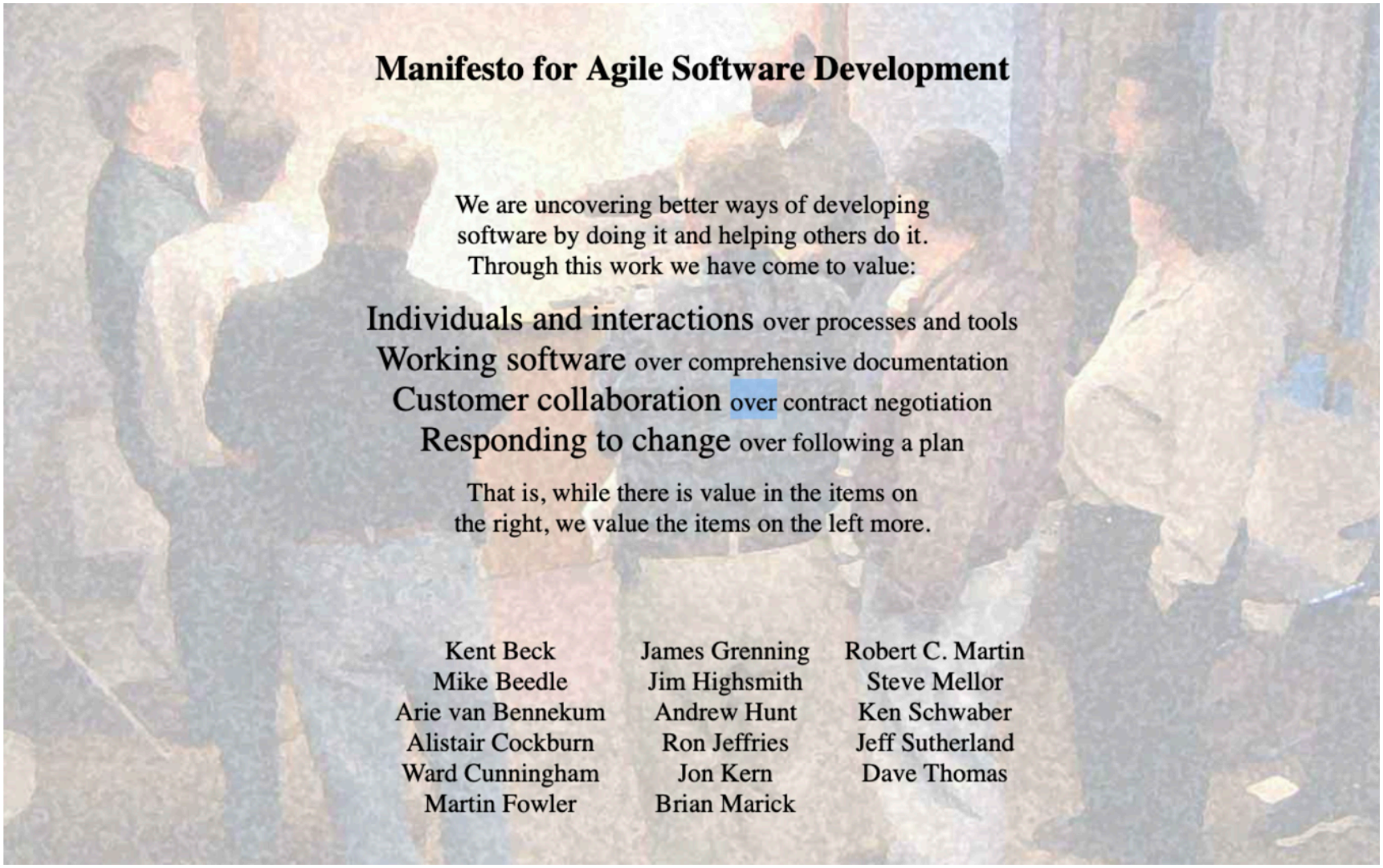


Figure 4: Released in 2001, the Agile Manifesto attempted to reframe how we manage software projects [2]. It inspired a large-scale shift in software development practices.

Agile Methods

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

- Teams may be geographically dispersed and work on the system for years.
- 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 address this problem.

- These methods focus on the software rather than its documentation, develop software in a series of increments and aim to reduce process bureaucracy as much as possible.

Incremental Development

All agile methods recognize the importance of incremental development and delivery.

Product development focuses on software features, where a feature does something for the software user.

With incremental development, you start by prioritizing the features so that the most important features are implemented first.

- You only define the details of the feature being implemented in an increment.
- That feature is then implemented and delivered.

Users can try it out and provide feedback to the development team.

- Use feedback to define and implement the next feature of the system.

Incremental Development

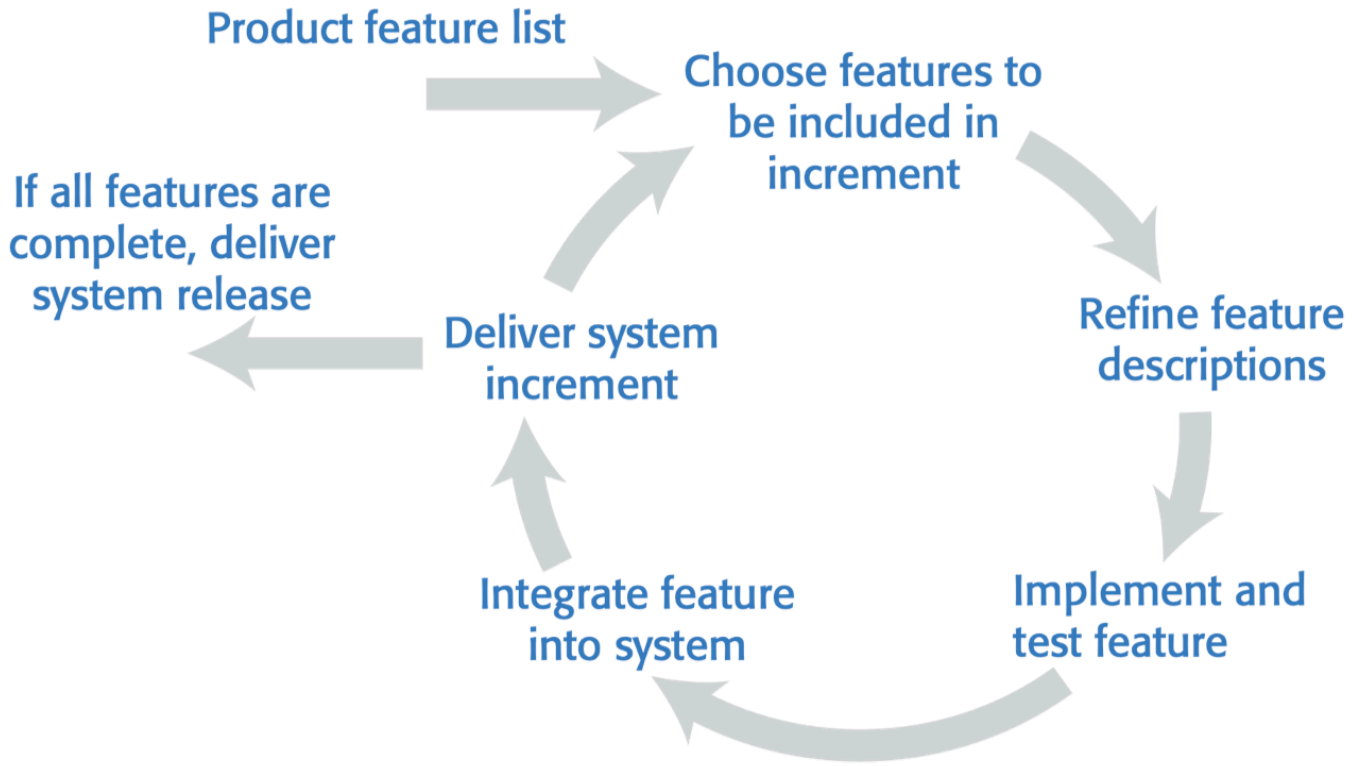


Figure 5: Incremental development doesn't attempt to design and build the entire system, but rather focuses on a cycle of continuous feedback and system increments. Diagram from Sommerville, Engineering Software Products, 2019.

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.

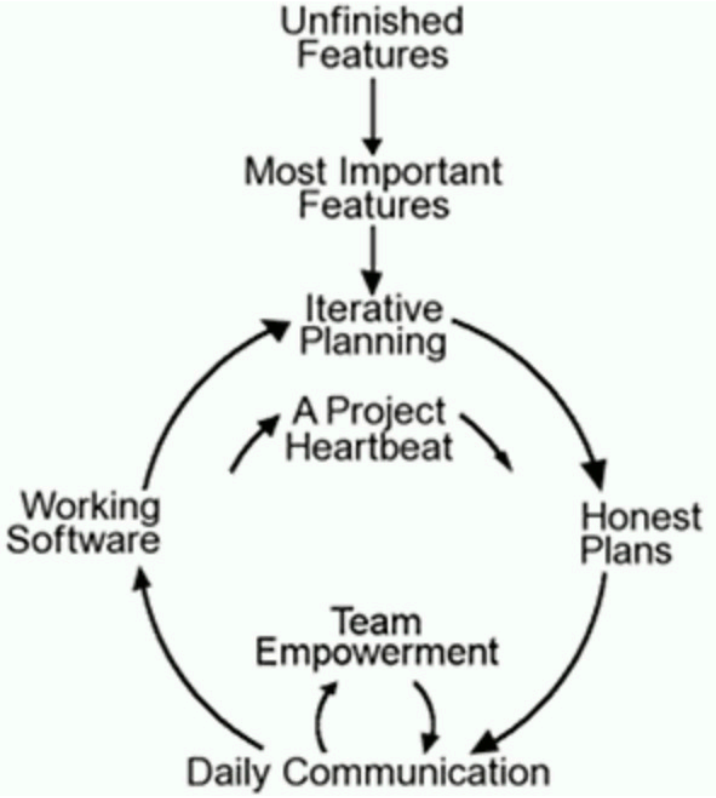


Figure 6: XP introduced the modern iterative development cycle.

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

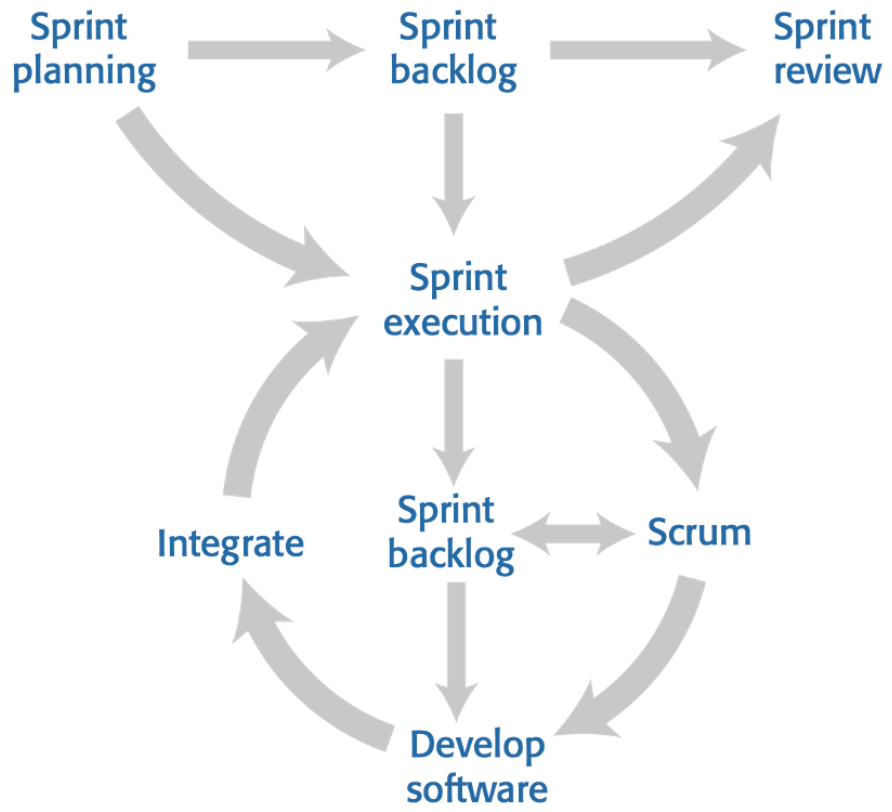


Figure 7: Sprints consist of planning activities, execution and delivery of software. Diagram from Sommerville, Engineering Software Products, 2019.

Being Agile

What We Will Use

In this course, we will adopt practices from both XP and Scrum.

Scrum

- We will work in three-week iterations.
- You will have a backlog of work items (`issues` in GitLab, identified during your Project Proposal).
- You and your team will prioritize and dcide what to work on each sprint.
- Each sprint will end with a demo to your user (aka TA).

XP

- User stories. We will use user stories as a mechanism to describe related features and functionality.
- We will use TDD and produce automated tests.
- We will produce working software with each sprint. You will be expected to produce a `software release` and demonstrate your progress at regular intervals.

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*, Second. 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