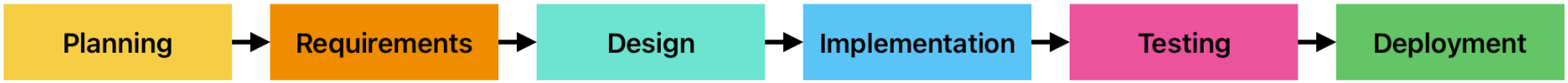
Agile & SDLC

CS 346 Application Development

How to build software

People often think that building software is like building anything else, e.g., a car, or a refrigerator.

At first glance, this *seems* reasonable: software is something that you manufacture. Your project includes determining requirements, designing and building something. You might envision a process that looks something like this:



How to build software

Planning - “What are our goals?”, “What is the budget?”, “Who is working on it?”

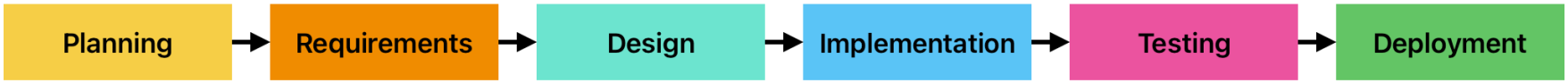
Requirements - “Who are our users?”, “What problem are we solving?”

Design - “What technical constraints exist?”, “What might it look like?”

Implementation - “How do we build it efficiently?”

Testing - “Does it meet specifications?”

Deployment - “How do we sell it and maintain it properly?”

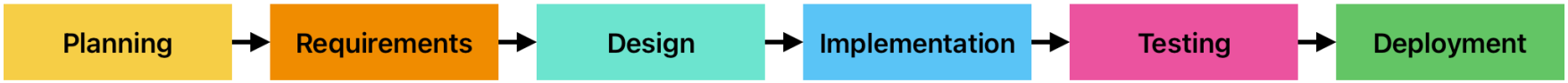


Process models

We use the term **process model** to describe this structure of activities.

> "A process model defines the complete set of activities that are required to specify, design, develop, test and deploy a product, and describes how they fit together."

A **software process model** is a process model adapted to describe how we might build software systems. We also refer to a software process model as the **Software Development Lifecycle (SDLC)**.

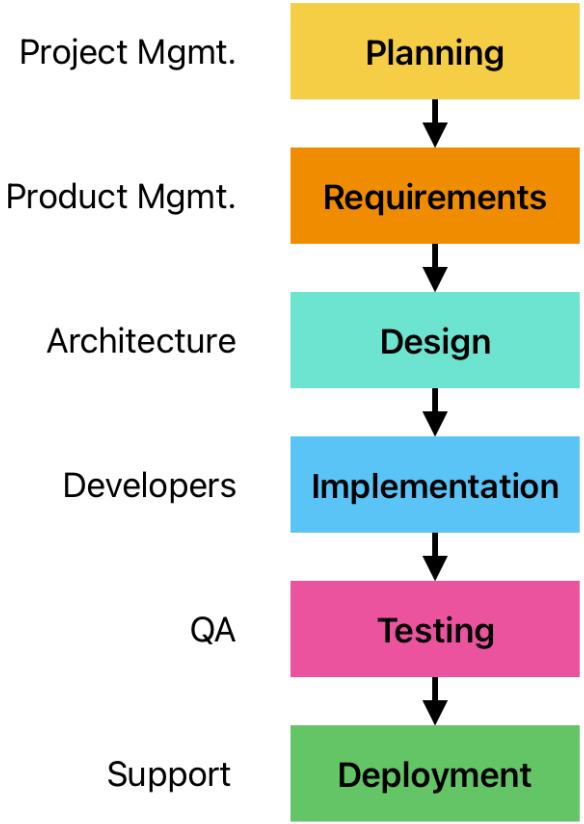


SDLC: Waterfall

In a 1970 paper, Winston Royce described a process model that envisions software production as a series of cascading steps.

He dubbed this this **Waterfall Model**:

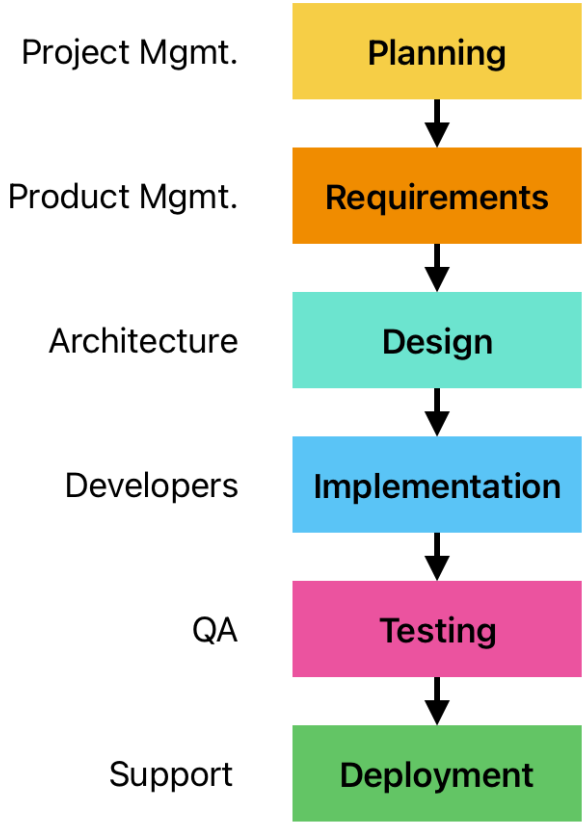
- Steps represent required work.
- Each step is “owned” and managed by a separate person or group.
- Steps need to be performed in order.
- Gatekeeping enforced e.g., requirements approval before design.



Challenges

This doesn't work very well. Why?

- Decisions made early in the process may need to be revisited. e.g., customer requirements.
- Your understanding of a problem will evolve. You will sometimes need to iterate to make a final decision e.g., uncovering issues in development that should change the design.
- Building silos discourages collaboration, and leads to limited decisions e.g., QA will have insights on what designs are testable.



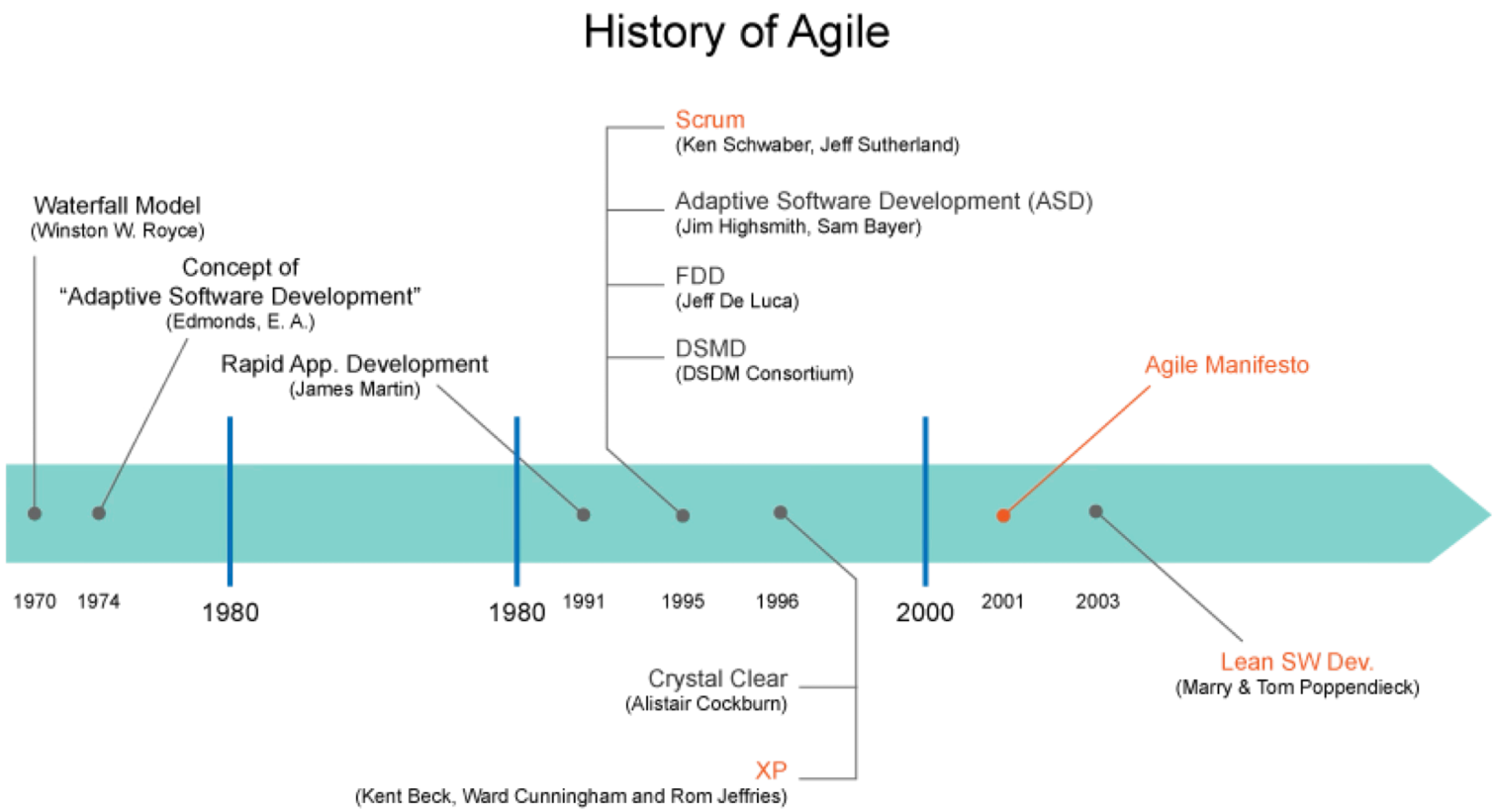
New Process Models

By the mid-1990s, there was a widespread recognition that this way of building software just didn't work:

- Developers were frustrated by rigid processes/changing requirements.
- Business owners were frustrated by the inability to make changes to projects once they were past the requirements phase.
- Projects were being delivered late and/or over-budget.

Alternate models were devised

- Extreme Programming (XP), Scrum, Lean, Rational Unified Process (RUP), Crystal Clear and others.



By the late 1990s, there was substantial discussion around how to create a “better” process model. Models tend to focus on different areas of the software process.

https://agilemanifesto.org/

The Agile Manifesto (2001)

Individuals and interactions (over processes and tools): Emphasis on communication with the user and other stakeholders.

Working software (over comprehensive documentation): Deliver small working iterations of functionality, get feedback and revise based on feedback. You will NOT get it right the first time.

Customer collaboration (over contract negotiation): Software is a collaboration between you and your stakeholders. Plan on meeting and reviewing progress frequently. This allows you to be responsive and correct your course early.

Responding to change (over following a plan): Software systems live past the point where you think you're finished. Customer requirements will change as the business changes.

What is Agile?

"Agile Software Development" isn't a single process, but rather an **approach to software development** that encompasses this philosophy. It **encourages team structures and attitudes that make communication easier** (among team members, business-people, and between software engineers and their managers). It **emphasizes rapid delivery of operational software but** also recognizes that planning has its limits and that a project plan must be flexible.

— Pressman & Maxim 2020.

There are many different Agile models. We'll focus on two: **Scrum**, and **XP**.

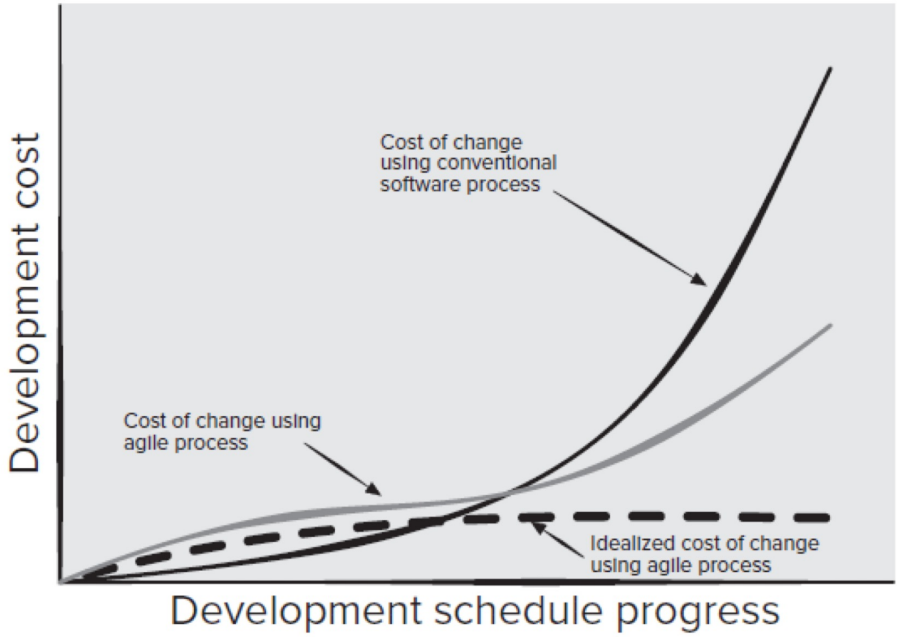
Common factor: iterative development

Agile models are iterative.

They recognize that the cost of change increases nonlinearly as a project progresses.

- Cost includes time, effort and money.
- The later you recognize a problem, or introduce a new requirement, the costlier it will be.

Iterative approaches encourage you to make required changes earlier in the process, when the cost of making changes is lower.



What does iterative development look like?

Getting feedback at every stage of development.

- Identify incorrect requirements earlier, so that you don't waste time designing something that isn't needed.
- Identify poor designs earlier, before you waste time refining and polishing and implementing the wrong design.
- Identify failing tests earlier, so that you can correct them through design changes and not just hacking together a fix.

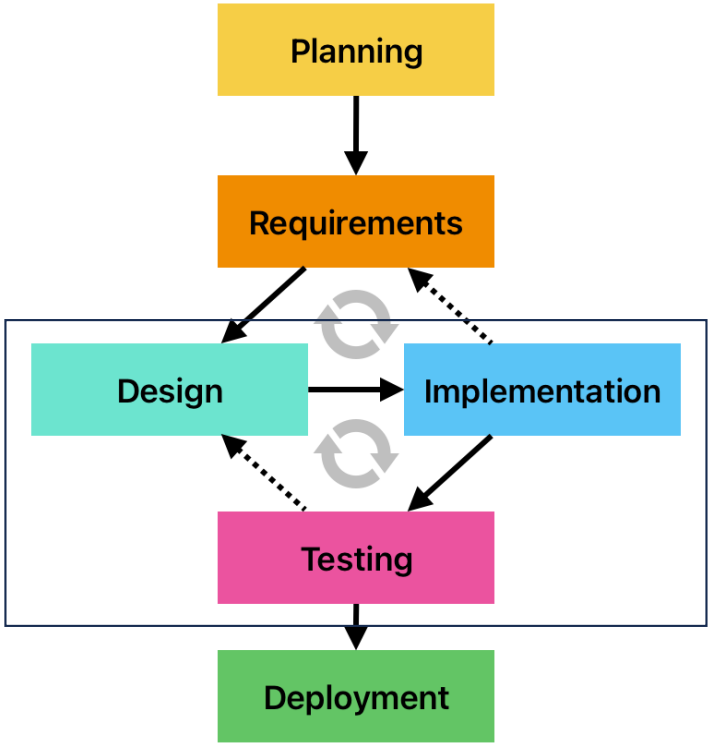
Focus on delivering one feature at-a-time and getting immediate feedback.

- Feedback from customer, development team, stakeholders.

Agile SDLC: first draft

An iterative SDLC could look like this:

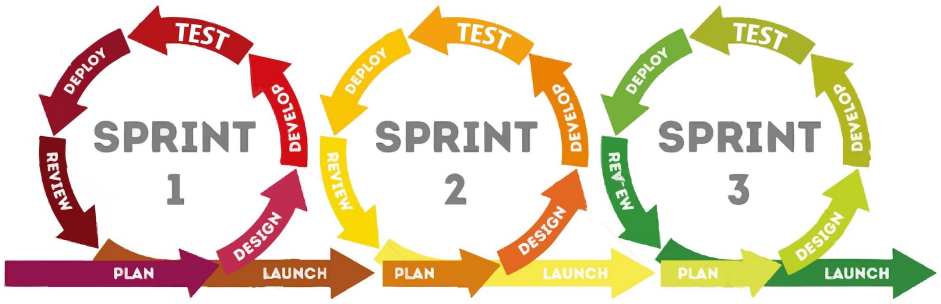
- Planning and requirements are done up-front, although we can revisit them.
- We iterate over design, implementation and testing.
  - You can “go backwards” at any time.
  - Testing often requires design changes; implementation may impact requirements.
- Deployment happens after a cycle.
- This entire cycle can be repeated.



Scrum as an iterative model

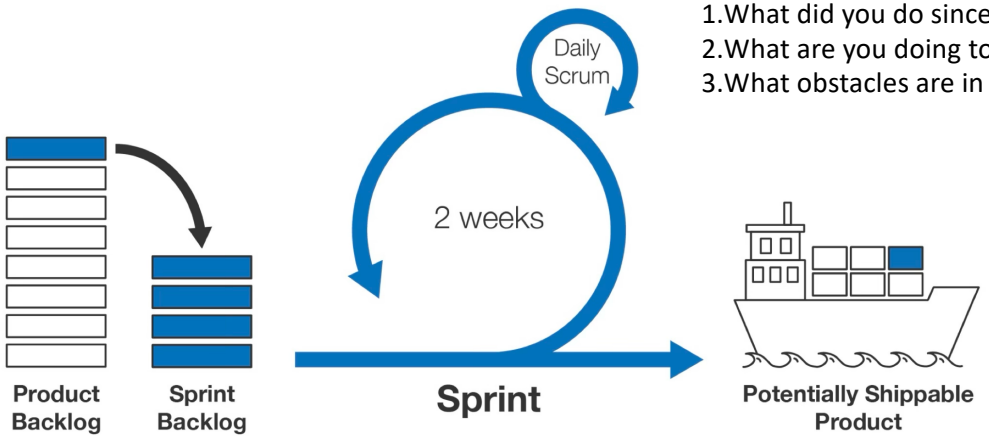
Scrum is a popular Agile process models that focuses on the design-development cycle.

Scrum breaks down a project into fixed-length iterations called **sprints** (where each spring is typically 2-4 weeks in length). A sprint is a focused development cycle where you design-develop-test several features and release them in a fully-tested and shippable product at the end of each sprint.



Daily scrum == Daily standup meeting
1.What did you do since the last meeting?
2.What are you doing today?
3.What obstacles are in your way?

List of all potential features or changes to consider.



https://kunzleigh.com/about/our-approach/

Product Backlog is a list of all possible features and changes that could be considered. Collected from the customer's feedback, or ideas that the team has (i.e. during the Requirements phase).

Product Owner is the person responsible for gathering requirements and placing in the product backlog (i.e. Product Manager or Customer).

Sprint Backlog is the set of features that are assigned to a sprint. It's "in-scope" for that sprint.

Scrum Master is the person that helps facilitate work during the sprint (sim. to a "team lead").

Scrum Flow

In this course, sprints are two-weeks long, and we will have four sprints in total (i.e. 4 x 2-week sprints). Each sprint includes the following activities:

1. **Feature Selection**. On the first day of the Sprint, the team meets and selects features. *Issues are moved from the Product Backlog into the Sprint Issues list and assigned. You are committing to work that you can complete in the upcoming sprint.*

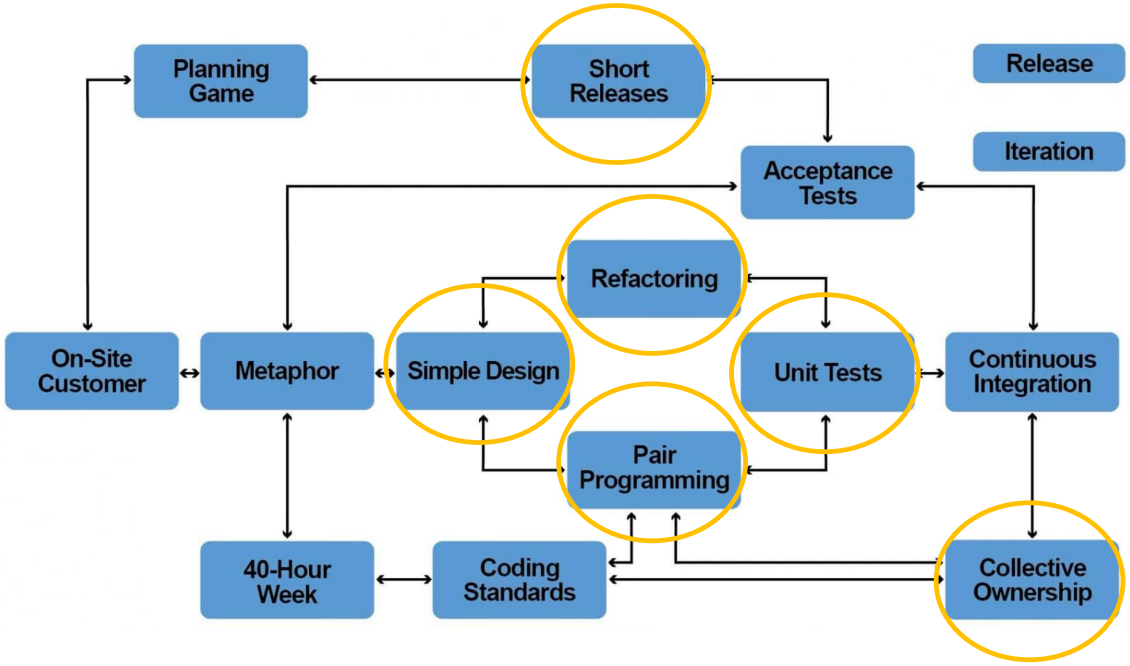
2. **Implementation**. During the sprint, the team iterates on their features. *As each feature is completed, unit tests are written/passed. When complete, the issue is closed.*

3. **Evaluation**. At the end of the Sprint, the team meets with the Product Owner to demo what they have completed and get feedback. *Only completed work is shown. Issues that are not completed are moved back into the Product Backlog to be reconsidered.*

+ **Retrospective**. The team should also reflect on what worked well, and what could improve. You should always be looking for ways to improve how you manage your project.

Extreme Programming (XP) Practices

XP is an Agile methodology focused on best-practices for programmers. It aims to produce higher-quality software and a higher quality-of-life for the development team.



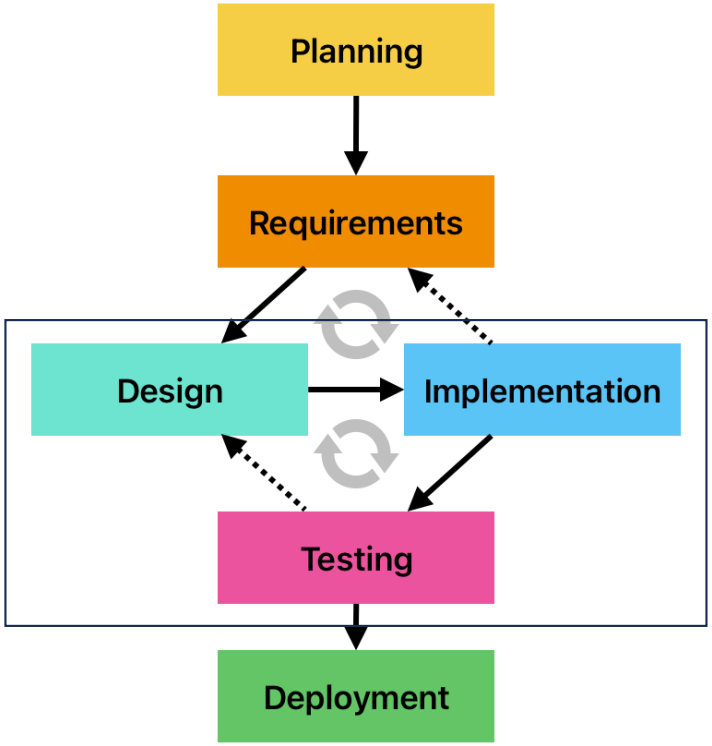
What next?

We'll continue to discuss activities and practices that are relevant to this process.

- We'll focus mainly on design, implementation and testing activities

If you want to learn more about requirements methodologies, recommended courses include:

- CS 445 Software Requirements.
- CS 449 Human-Computer Interaction.



Week 01: What to do this week?

Register for the course.

- Talk to me if you are not registered. I'll override you if I can.
- You **must** attend in-person; you cannot take this class remotely.

Form teams! See Project > Guidelines

- Teams of 4 people.
- Must all be in the same sections. Talk to me if you wish to switch sections.
- A mix of skills in a team is useful! You want to be well-rounded.
- Find people that share your interests and have a similar work ethics/schedule.
- When formed, **follow the signup instructions**.

Reference

- Robert C. Martin. 2003. **Agile Software Development: Principles, Patterns and Practices**. Pearson. ISBN 978-0135974445.
- Pressman & Maxim. 2014. **Software Engineering: A Practitioner's Approach**. McGraw Hill. ISBN 978-0078022128.
- Schwaber & Sutherland. 2020. The Scrum Guide. Online.