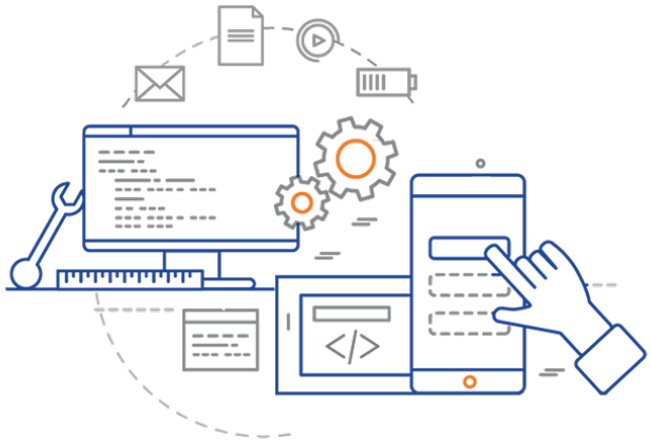


CS 398: Application Development

Week 01 Video: Software Process

Software is unusual, in that it's **virtual**. However, we want a lot of the same characteristics when building software that we do when building anything — quality, cost-effectiveness, useful features (low cost!).

A **software process** is a collection of activities, actions, and tasks that are performed when to create software. Process defines a framework that must be established for effective delivery of software engineering technology.

The process that we define needs to capture all of the activities that we perform to meet our objectives (e.g., communication with stakeholders) and is applied during the process.

Process Models

Process Activities

Example: building a house

These activities are common to any type of project - not just software.

- **Communication**: Discuss the goals and requirements with the customer, potential users, and other stakeholders. This is critical to ensure that we're solving the correct problem.

  Discuss with a builder.

- **Planning**: Identify people and resources, potential risks to the project, and developing a plan to mitigate these risks. These are what we would typically call "project management" activities.

  Hire a project manager.

- **Modelling**: Design abstract models or representations of the product that you need to build.

  Get blueprints!

- **Construction**: The process of building your product i.e. realizing your design. This may require successive iterations to complete, and should also include some level of testing and validation.

  The crew builds your house.

- **Deployment**: Tasks required to produce a working product, and deliver it to the customer. This includes collecting their ongoing feedback and making revisions as needed.

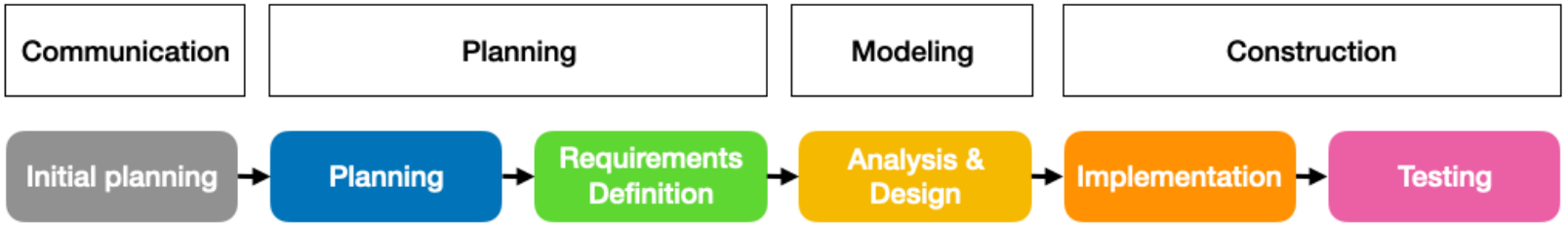
  You inspect the results and take possession!

Process Model?

We use the term **process model** to describe the structure that is given to these activities.

That is, it defines the complete set of activities that are required to specify, design, develop, test and deploy a system, and describes how they fit together. A software process model is a type of process model adapted to describe for software systems.

Below, we've listed the steps that we would typically consider as required for a software project.

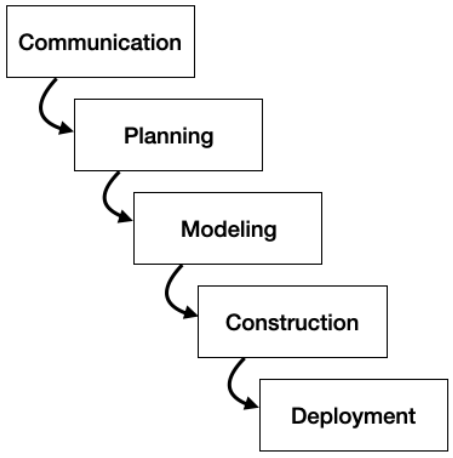


Software Development Activities

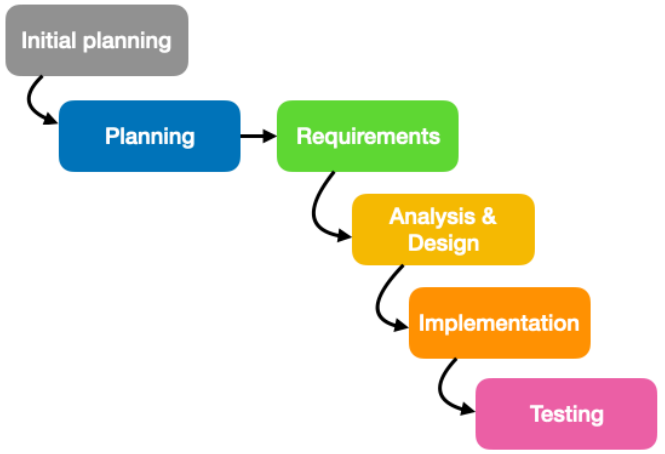
Waterfall Model

In a 1970 paper, Winston Royce laid out a mechanism for formalizing the large-scale management of software projects [Royce 1970], dubbed the Waterfall model. This envisions software production as a series of steps, each cascading into the next one, much like a waterfall.

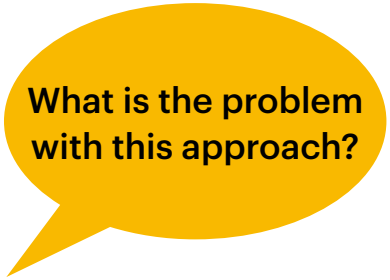
Software development is treated as a set of linear steps that are followed strictly in-order.



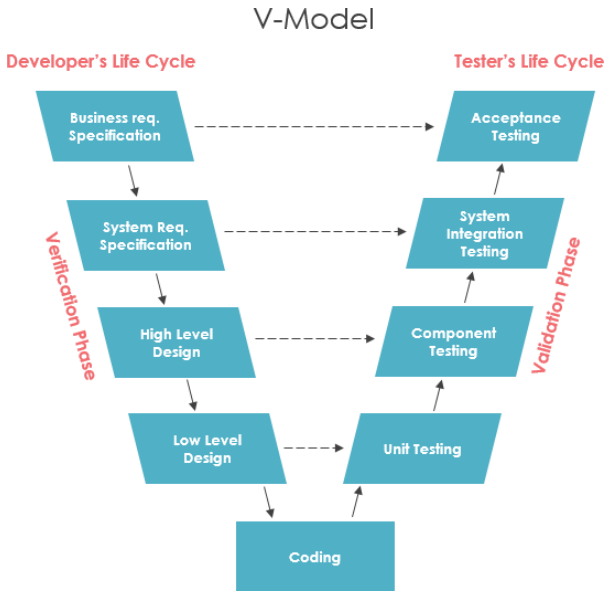
Linear Model (General)



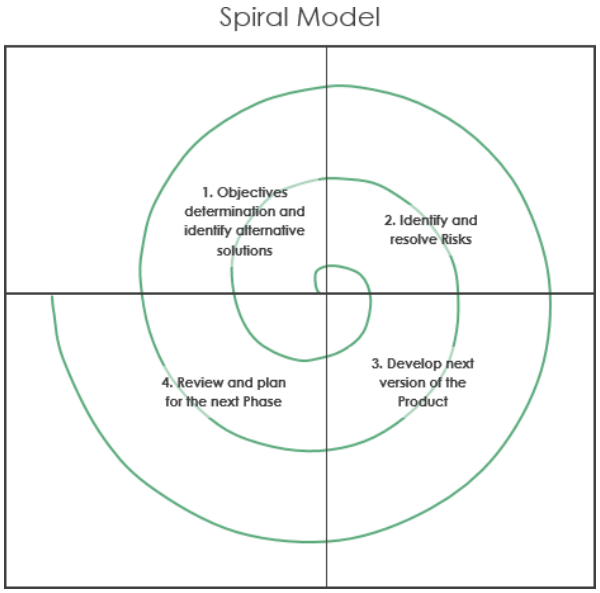
Waterfall Model (Software)



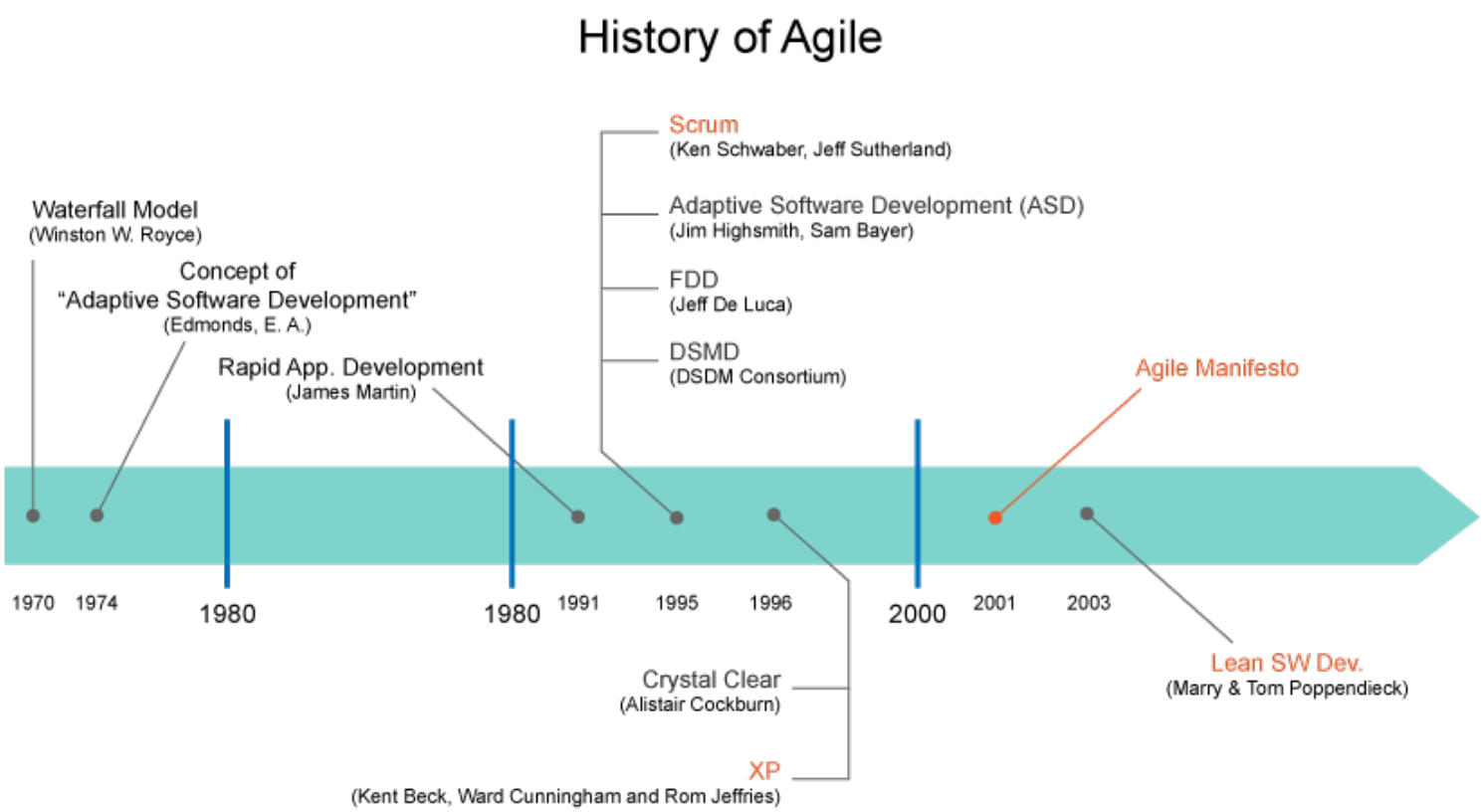
Other Process Models



V-Model lines up testing with the related area of interest. Acknowledges connections, but doesn't handle change very well.

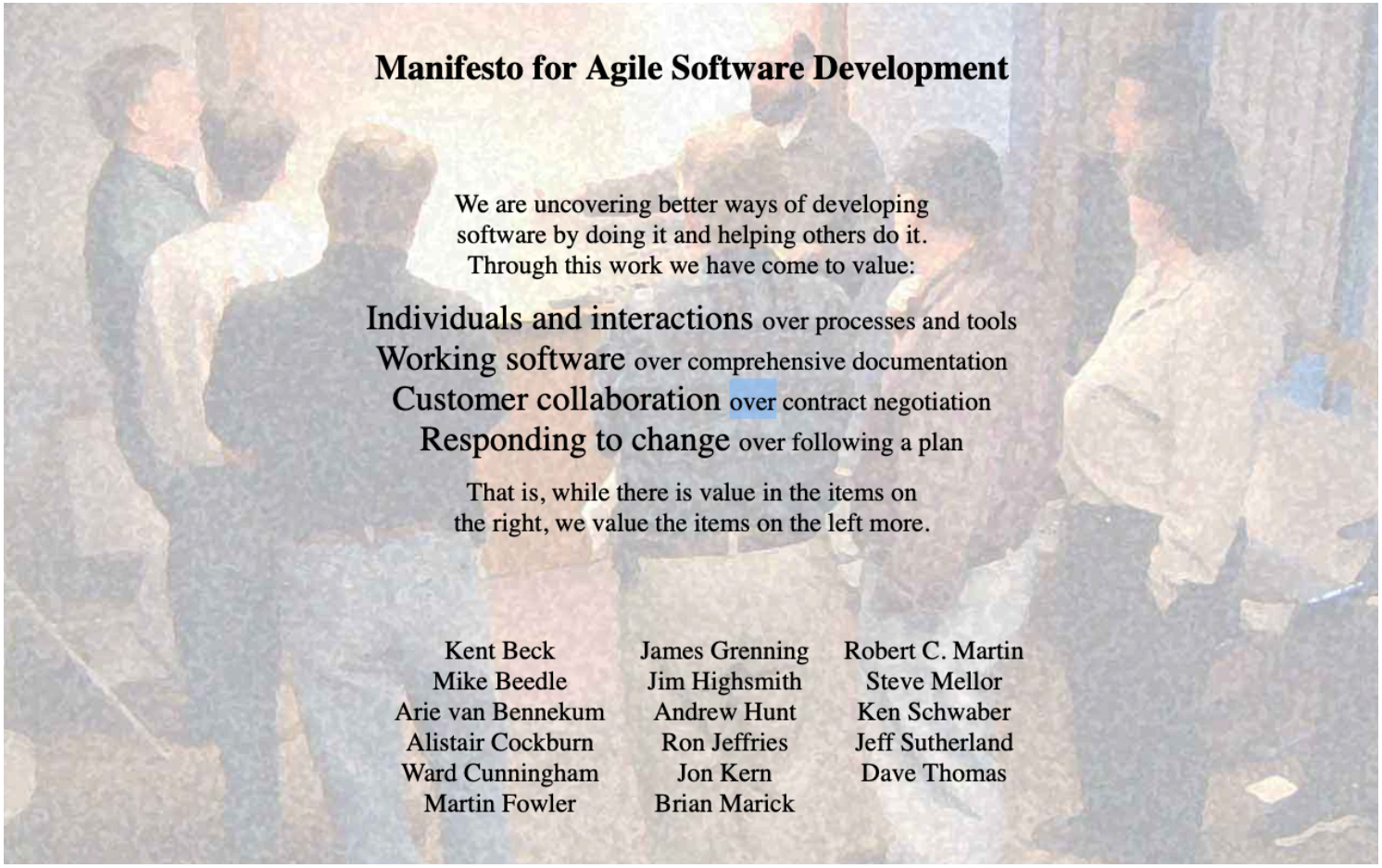


Spiral model defines iterations, starting with abstract and progressing through levels of detail. Defined by product manager to direct product versions.



The backlash was building against Waterfall.....

The Agile Movement



https://agilemanifesto.org

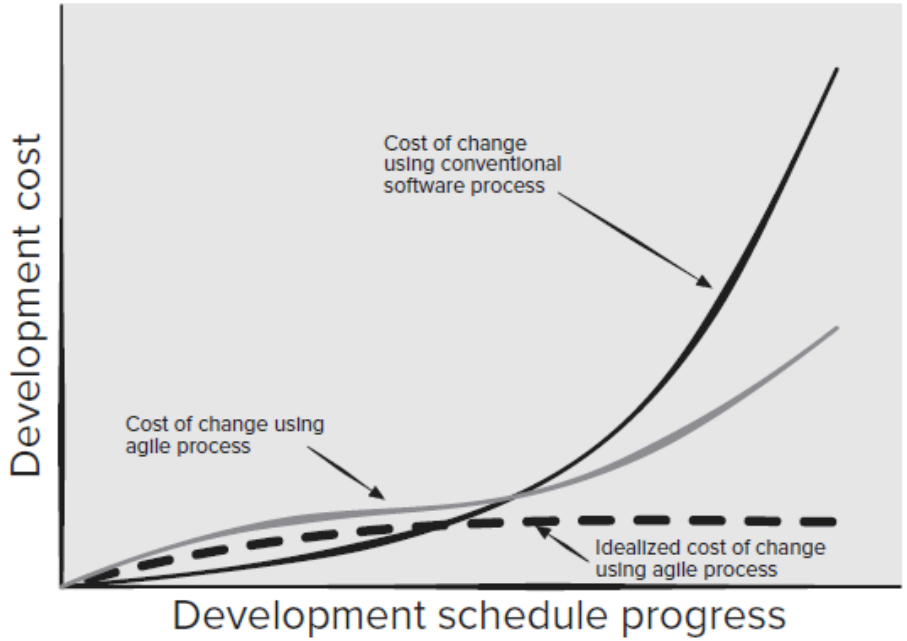
What is Agile?

“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.

Agility means recognizing that requirements and plans will change over time.

1. Software is too complex to design and build all at once. It’s more manageable to add features and test incrementally.
2. Software is in a constant state of change, and requirements will change during the development cycle.

What is the benefit?



The conventional wisdom in software development is that the cost of change increases nonlinearly as a project progresses. Agility means expecting project and requirements changes. The benefit of Agile is that it reduces (and tries to eliminate) breaking late-project changes.

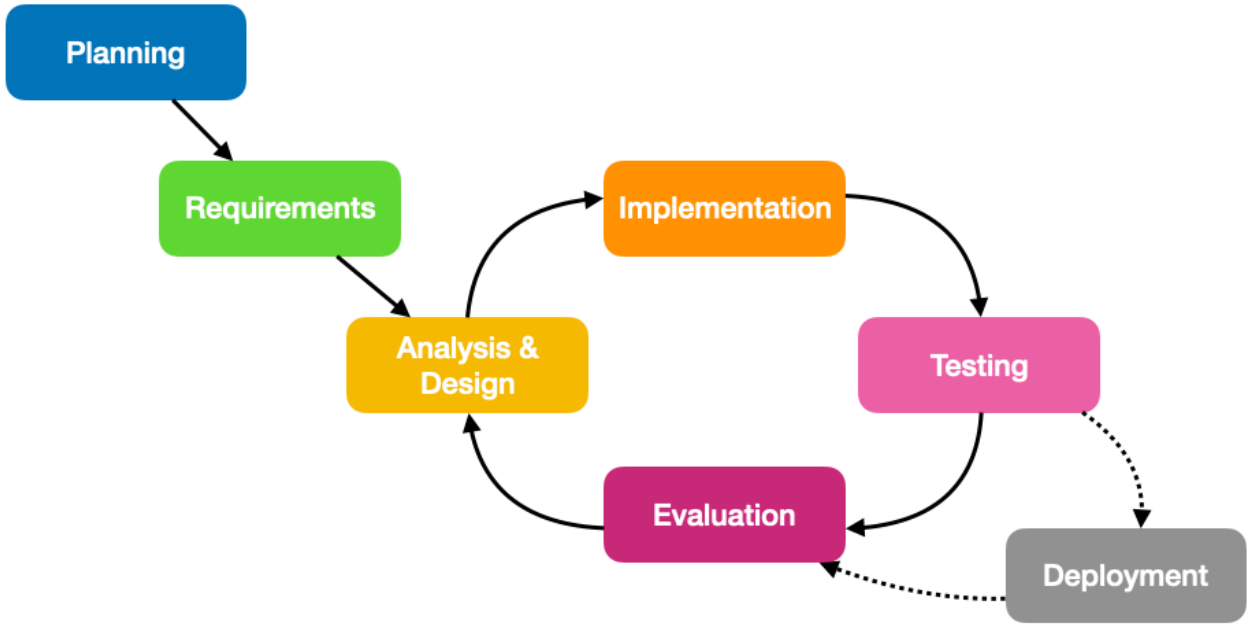
Challenges

Key challenges:

1. It is difficult to predict in advance which software requirements will persist and which will change. It is equally difficult to predict how customer priorities will change as the project proceeds.
2. For many types of software, design and construction are interleaved. That is, both activities should be performed in tandem so that design models are proven as they are created. It is difficult to predict how much design is necessary before construction is used to prove the design.
3. Analysis, design, construction, and testing are not as predictable (from a planning point of view) as we might like.

Given these assumptions, how do we create a process that can manage this type of unpredictability?

We use an **iterative, evolutionary development model**. Instead of building a “complete” system and then asking for feedback, we instead attempt to deliver features in small increments, in a way that we can solicit feedback continuously though the process.



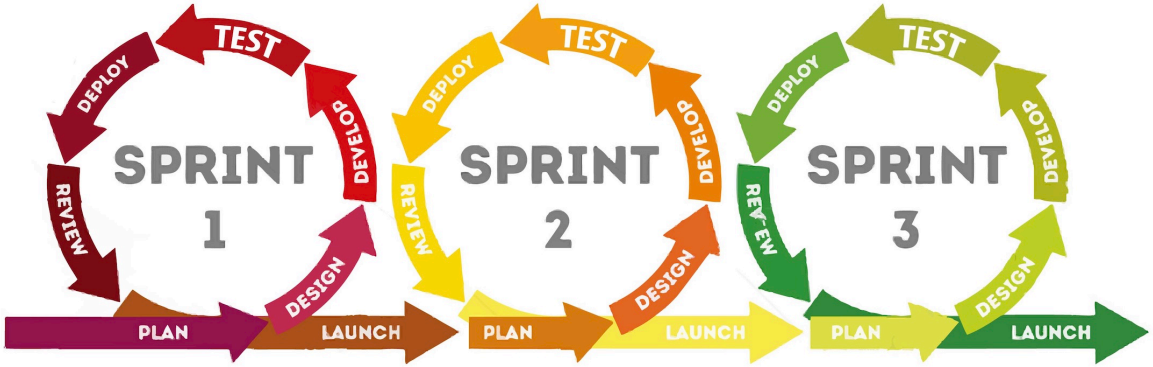
An “Agile” process *could* look like this.

Scrum

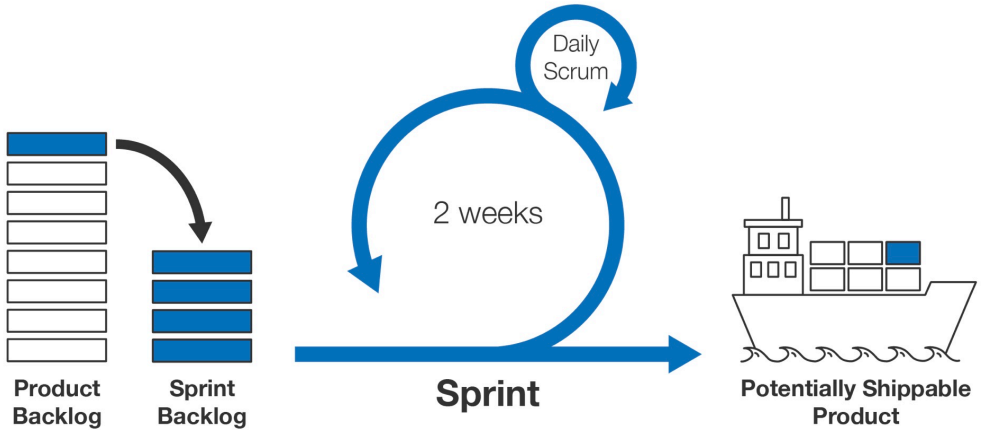
Scrum is one (of many) different Agile methods.

Scrum breaks down a project into fixed-length iterations called **sprints** (typically 2-4 weeks in length for each sprint). Sprints are defined so that you iterate on prioritized features in that time, and produce a fully-tested and shippable product at the end of each sprint.

Typically you will iterate until you and the customer together decide that you are “done”.



Key Concepts



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.
- **Product Owner** is the person responsible for gathering requirements and placing in the product backlog.
- **Sprint Backlog** is the set of features that are assigned to a sprint. This is the "scope" for that sprint.
- **Scrum Master** is the person that helps facilitate work during the sprint (like a "team lead").

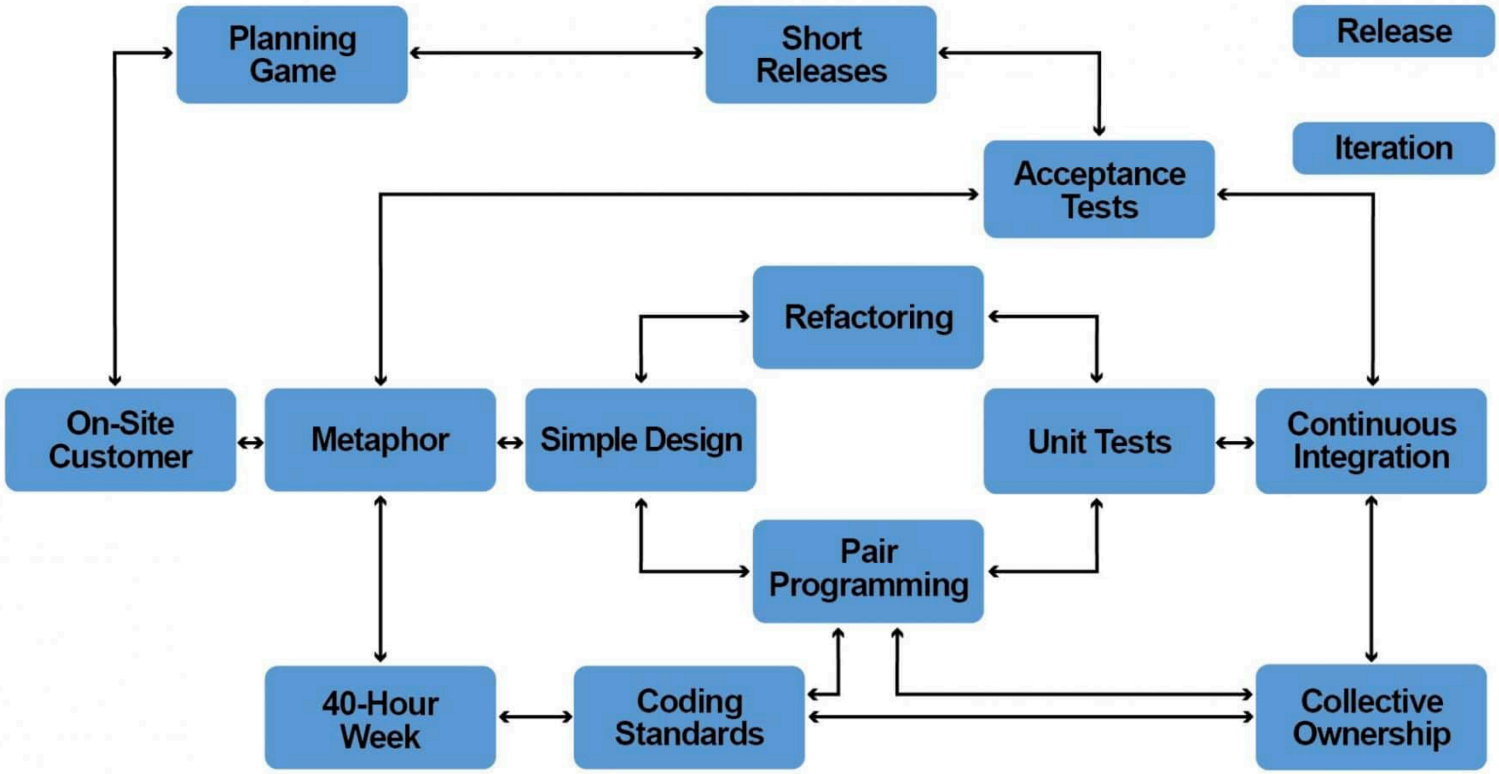
Extreme Programming

Extreme Programming (XP) is an Agile methodology focused on best-practices for programmers. It was based on a large-scale project that Kent Beck managed at Chrysler in the late 90s. It aims to produce higher-quality software and a higher quality-of-life for the development team.

The five core values of XP are:

- **Communication**: The key to a successful project. It includes both communication within the team, and with the customer. XP emphasizes face to face discussion with a white board (figurtively).
- **Simplicity**. Build the “simplest thing that will work”. Follow YAGNI (You Ain’t Gonna Need It) and DRY (Don’t Repeat Yourself).
- **Feedback**. Team members solicit and react to feedback right away to improve their practices and their product.
- **Courage**: The courage to insist on doing the “right thing”. The course to be honest with yourselves if something isn’t working, and fix it.
- **Respect**: Respect your team members, and remember that development is a collaborative exercise.

XP launched with 12 “best practices” of software development. *Have you heard of any of these?*



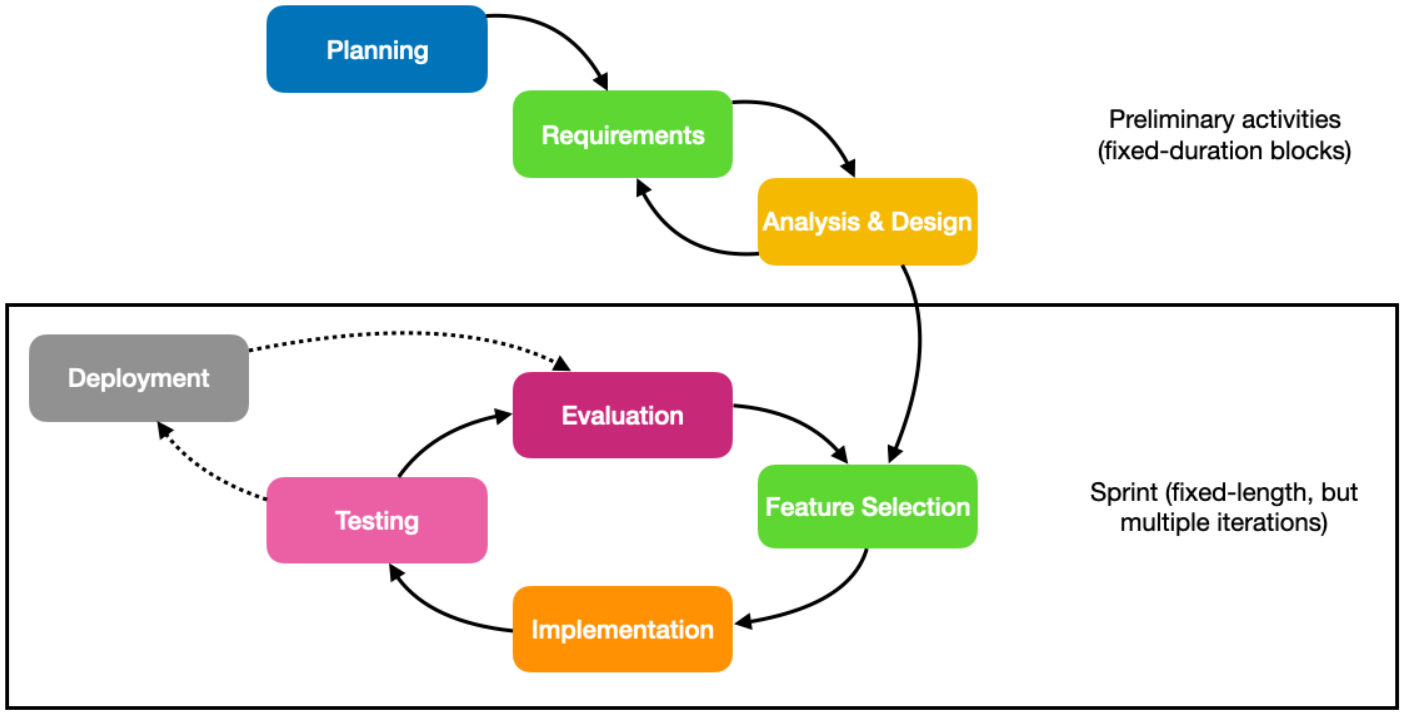
Being Agile

Agile Principles

From these different Agile models, we can extract a set of useful guiding principles [Pressman 2018]. This is what we aspire to do with our practices.

- **Principle 1**. **Be agile**. The basic tenets of agile development are to be flexible and adaptable in your approach, so that you can adjust if needed between iterations. Keep your technical approach as simple as possible, keep the work products you produce as concise as possible, and make decisions locally whenever possible.
- **Principle 2**. **Focus on quality at every step**. The focus of every process activity and action should be the quality of the work produced.
- **Principle 3**. **Be ready to adapt**. When necessary, adapt your approach to constraints imposed by the problem, the people, and the project itself.
- **Principle 4**. **Manage change**. The approach may be either formal or informal, but mechanisms must be established to manage the way changes are requested, assessed, approved, and implemented.
- **Principle 5**. **Build an effective team**. Software engineering process and practice are important, but the bottom line is people. Build a self-organizing team that has mutual trust and respect.
- **Principle 6**. **Establish mechanisms for communication and coordination**. Projects fail because important information falls into the cracks and/or stakeholders fail to coordinate their efforts to create a successful end product. Keep lines of communication open. When in doubt, ask questions!

Agile Process



Software Development Lifecycle (SDLC)

Phases

Planning, Requirements, Analysis & Design

We have roughly one week for each of these phases.

Sprints (which subsume Implementation, Testing and Evaluation).

In our 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 decides what features to add (and what bugs to fix) during that iteration.
2. **Implementation/Testing**. During most of the sprint, the team iterates on their features. As each feature is completed, it is tested.
3. **Evaluation**. At the end of the Sprint, the team meets with the Product Owner to demo what they have completed, and get feedback. The team also has a Retrospective, where they reflect on how to improve.

Agile Practices

Being Agile includes both the process AND some best practices that help us meet our goals.

We will discuss these in upcoming sections, in relation to the phase where they apply.

- **User stories**: describe features in a way that makes sense to customers.
- **Test-driven development**: tests are written before the code. This helps to enforce contracts/ interfaces.
- **Refactoring**: small continual improvements to code should be done routinely.
- **Pair programming**: critical code is written by two people working as a team.
- **Code reviews**: code changes need to be reviewed by one or more other developers on the team.