

# Software Engineering

---

CS 346 Application  
Development

# Software Engineering

Today we're going to talk about the *process* of designing and building software. This is a significant topic within the discipline of Software Engineering:

“Software engineering is a branch of both computer science and engineering focused on designing, developing, testing, and maintaining software applications ...

A software engineer applies a **software development process**, which involves defining, implementing, testing, managing, and maintaining software systems...”

-- [Wikipedia](#)

# Software Projects

Start at the beginning...

# Software Projects

- A project is a planned activity, following a set of defined steps, which results in some desired outcome e.g., building a fence.
- A software project is a specialized project which results in the delivery of a **software product** e.g., a new release of macOS.
  - Generic process, with terms and concepts adapted for software.
  - Often represented as a set of linear steps, where each step must be completed before the next one starts.



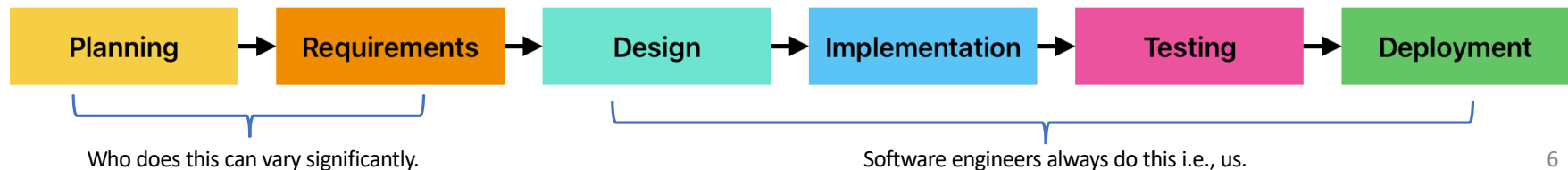
# Steps in a Software Project

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

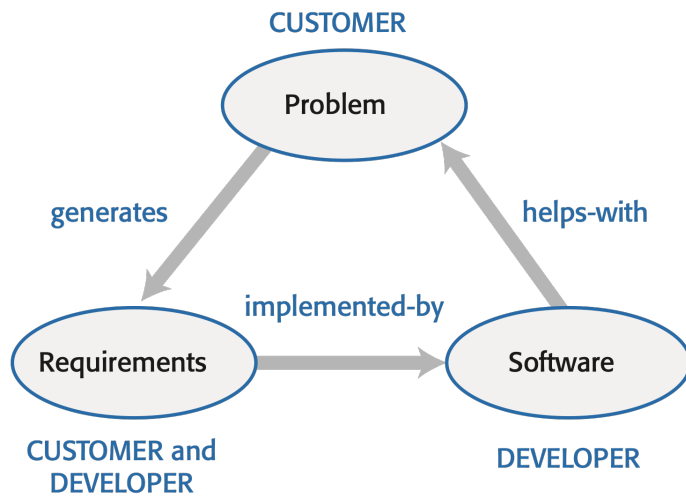


# Software Projects vs. Products

- 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).

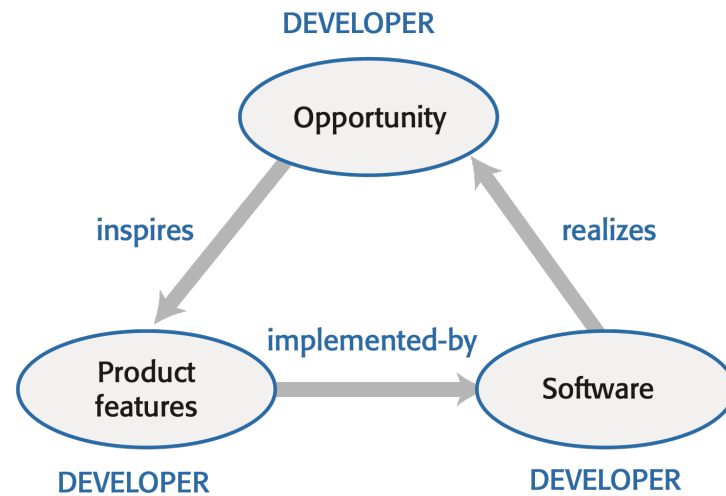


## Project Development



The customer is the primary driver of software requirements.

## Product Development



The developer decides who to target, and what features to deliver.

# Product Development

- The starting point for **product development** is a business opportunity that is identified by individuals or a company.
  - The company decides to develop a software product to take advantage of this opportunity and sell this to (hopefully many) customers.
  - They design and implement a set of software features that take advantage of this opportunity and that will be useful to customers.
- All decisions are being made by the developer!
  - No external customer is paying for anything (yet).
  - The company is responsible for deciding on the development timescale, what features to include and when the product should change.
  - Rapid delivery of software products is essential to capture the market.



# Software Process Models

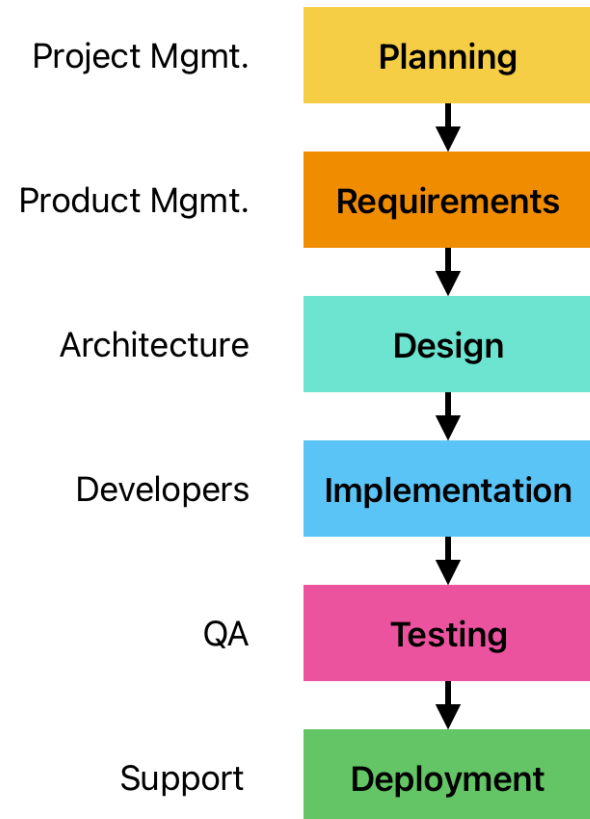
Let's think more about building products.

# Software process models

The linear model we presented was a common view of software development until recently.

It is also known as the **Waterfall Model** (Royce 1970):

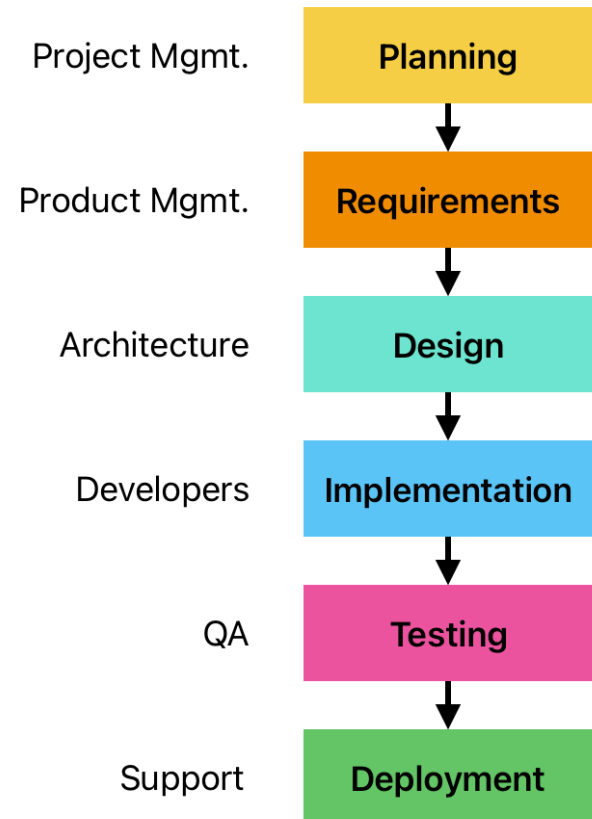
- Each step is “owned” and managed by a separate person or group.
- Each step represents significant work.
- Steps need to be performed in order.
- Gatekeeping enforced e.g., requirements approval before design.



# Challenges

## **This model 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.



# The Agile Manifesto (2001)

<https://agilemanifesto.org>



**Manifesto for Agile Software Development**

We are uncovering better ways of developing software by doing it and helping others do it.  
Through this work we have come to value:

**Individuals and interactions** over processes and tools  
**Working software** over comprehensive documentation  
**Customer collaboration** over contract negotiation  
**Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck	James Grenning	Robert C. Martin
Mike Beedle	Jim Highsmith	Steve Mellor
Arie van Bennekum	Andrew Hunt	Ken Schwaber
Alistair Cockburn	Ron Jeffries	Jeff Sutherland
Ward Cunningham	Jon Kern	Dave Thomas
Martin Fowler	Brian Marick	

# 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.

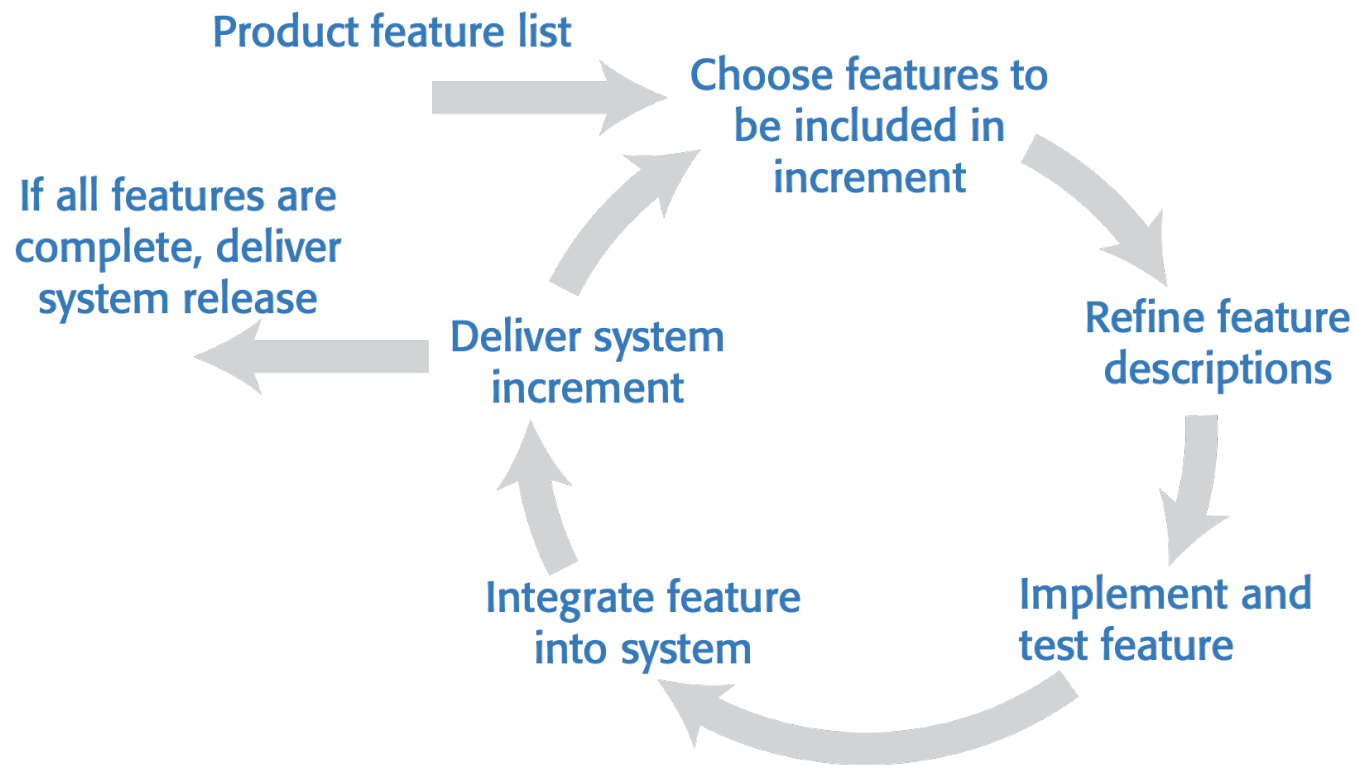
# Agile software engineering

- Agile software engineering focuses on:
  - Rapid software development and delivery,
  - Responding to changing product specifications and
  - Minimizing development overhead.
- Many 'Agile methods' have been developed.
  - There is no 'best' Agile method or technique.
  - It depends on who is using the technique, the development team and the type of product being developed.
  - Teams often combine aspects of different models.
- Modern software development tends to be Agile.

# Key concept: Incremental development

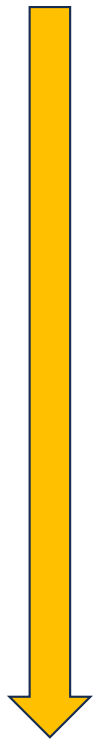
- All agile methods recognize the importance of incremental development and delivery.
- Product development focuses on software features, where a feature does something for the software user.
- With incremental development, you start by prioritizing the features so that the most important features are 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





# Incremental development activities



- ***Choose features to be included in an increment***  
Using the list of features in the planned product, select those features that can be implemented in the next product increment.
- ***Refine feature descriptions***  
Add detail to the feature descriptions so that the team have a common understanding of each feature and there is sufficient detail to begin implementation.
- ***Implement and test***  
Implement the feature and develop automated tests for that feature that show that its behaviour is consistent with its description.
- ***Integrate features and test***  
Integrate the developed feature with the existing system and test it to check that it works in conjunction with other features.
- ***Deliver system increment***  
Deliver the system increment to the customer or product manager for checking and comments. If enough features have been implemented, release a version of the system for customer use.

# Why Agile Processes?

Compared to plan-driven development, Agile processes:

- ***Involve the customer***  
Involve customers closely with the software development team. Their role is to provide and prioritize new system requirements and to evaluate each increment of the system.
- ***Embrace change***  
Expect the features of the product and the details of these features to change as the development team and the product manager learn more about it. Adapt the software to cope with changes as they are made.
- ***Develop and deliver incrementally***  
Always develop software products in increments. Test and evaluate each increment as it is developed and feed back required changes to the development team.

# Agile Principles

Foundational Agile principles include:

- ***Maintain simplicity***

Focus on simplicity in both the software being developed and in the development process. Wherever possible, do what you can to eliminate complexity from the system.

- ***Focus on people, not things***

Trust the development team and do not expect everyone to always do the development process in the same way. Team members should be left to develop their own ways of working without being limited by prescriptive software processes.



# Extreme Programming (XP)

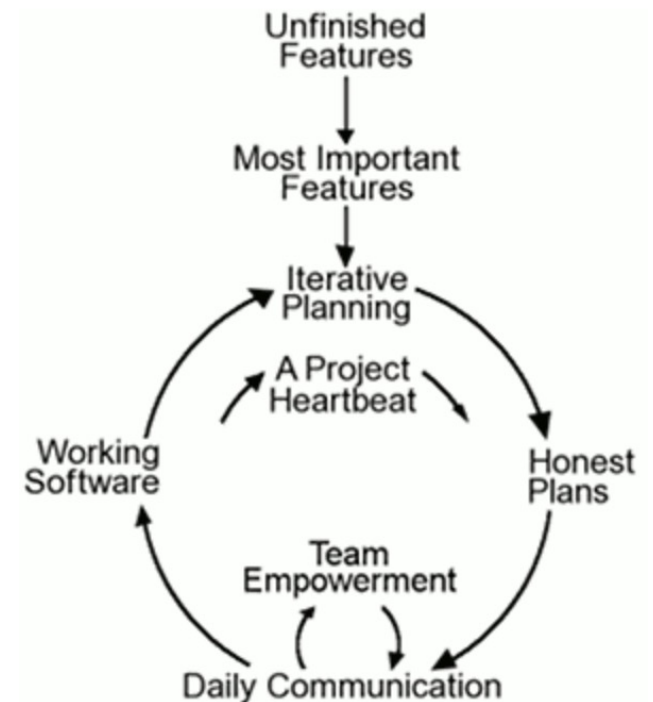
A foundational Agile process model.

# Extreme programming (XP)

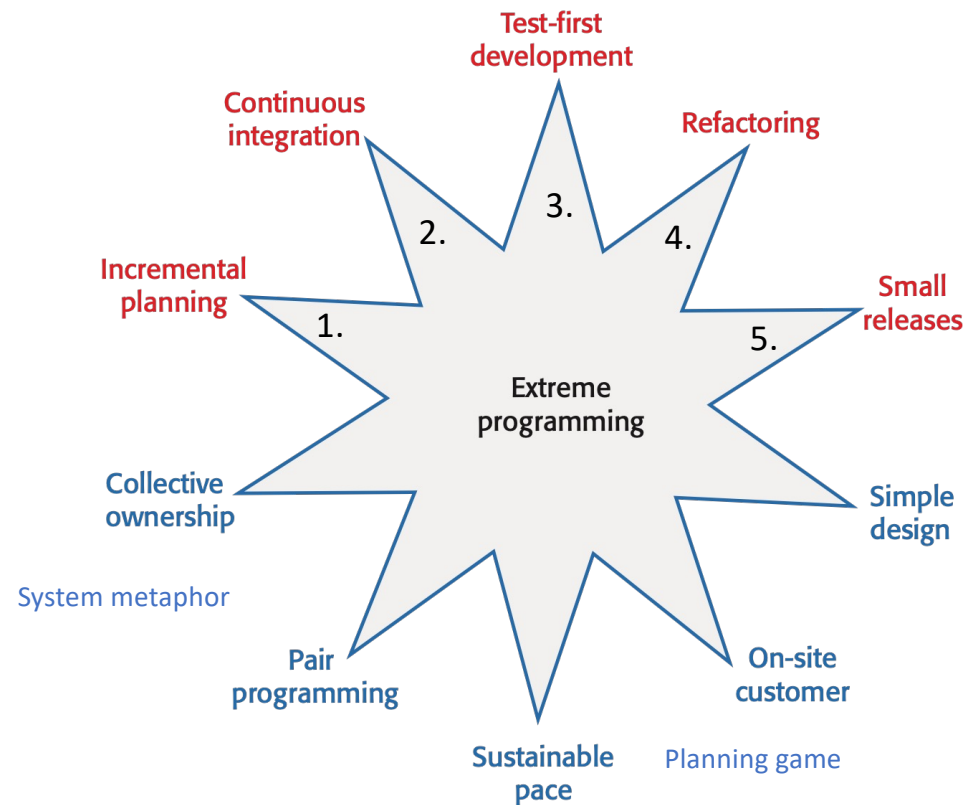
One of the most influential process models is **Extreme Programming** (XP), designed by Kent Beck in the late 1990s.

Extreme programming focuses on 12 development techniques that are geared to rapid, incremental software development.

- Very much concerned with improving quality-of-life for software developers.
- Less focused on project management and tracking concerns.



# Extreme programming practices



widely used  
less popular

# Widely adopted practices (that we will use)

## 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. **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.

## 3. **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.


# Widely adopted practices (that we will use)

## 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.



We will revisit  
these later.



# Scrum

Another foundational Agile process model.

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

# Key Scrum concepts

- **Self-organizing teams**

- Self-organizing teams make their own decisions and work by discussing issues and making decisions by consensus. No single person is “in charge”.

- **Timeboxed iteration (aka “sprint”)**

- The team has a fixed period (usually 2-4 weeks) where they decide on goals, select items to work on, implement them and then demo them to a customer at the end.

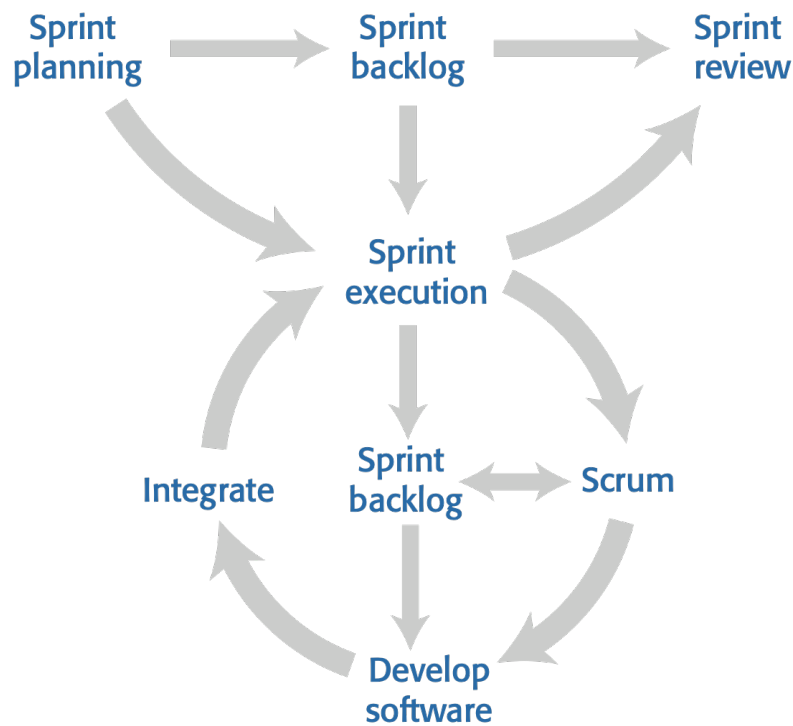
- **Customer feedback**

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

- **Work from a backlog**

- Work is normally only scheduled for the next sprint; you don’t try and plan the entire life of a product. Unscheduled work is tracked in a “backlog”.

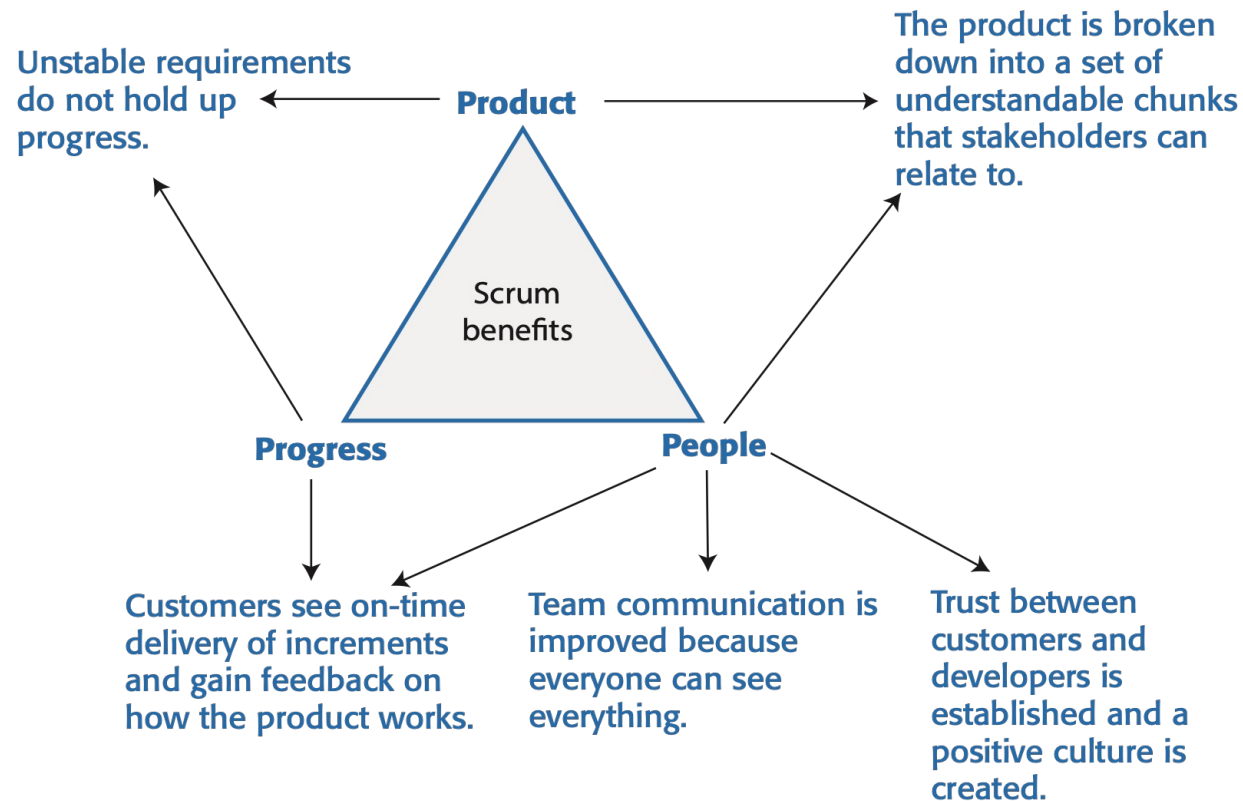
# Sprint activities

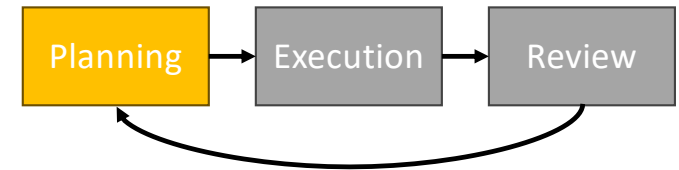


A sprint is 2-4 weeks in length.

- **Sprint planning:** set goals.
- **Sprint execution:** execute on goals.
- **Sprint review:** review the outcomes with a customer.

# The top five benefits of using Scrum





# Phase 1: Sprint planning

# Sprint planning

- In a sprint plan, the team decides which items in the product backlog should be implemented during that sprint.
  - Establish an agreed sprint goal. It may be focused on software functionality, support or performance and reliability.
  - Key inputs are *how much work they historically get done in a sprint (velocity)*.
- Planning activities (next slide) help prioritize items.
- The output of the sprint planning process is a sprint backlog.
  - Records the work to be done during the sprint.
- During a sprint, the team has regular (daily) meetings to coordinate their work.
  - i.e. check-in with each other, identify problem areas.

# Planning activities

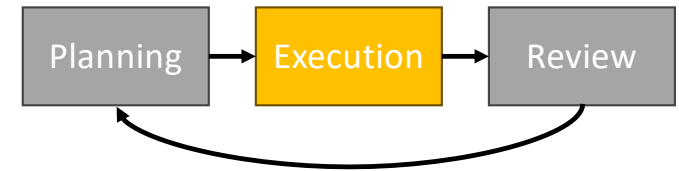


You work on product backlog items (PBIs) during sprint planning.

Consider PBIs that meet your goals for the sprint.

- **Creation:** New items are added to the backlog. These may be new features suggested by the customer, feature changes, engineering improvements, or process
- **Refinement:** Existing PBIs are analyzed and refined to create more detailed PBIs.
- **Estimation:** The team estimates the amount of work required and updates online.
  - Story points (arbitrary number denoting “effort”) or half-day estimates.
- **Prioritization:** The product backlog items may be reordered based on circumstances.
  - Recommend adding a priority label (high, medium, low) to track priorities.

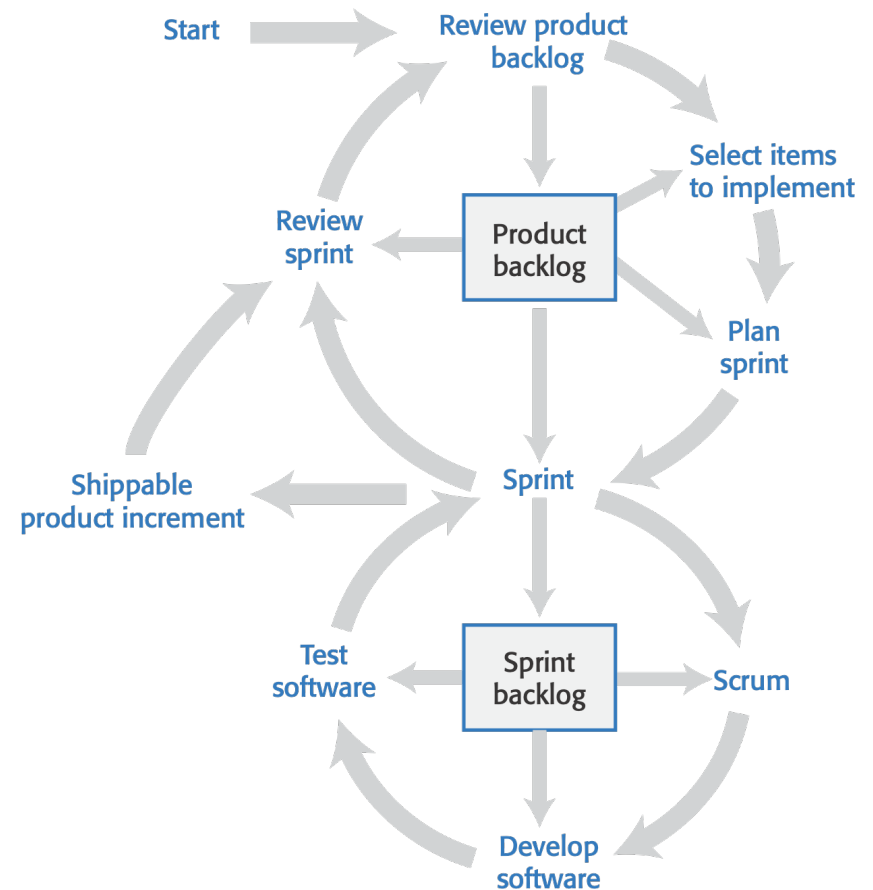




## Phase 2: Sprint execution

# Sprint execution

- **Sprints** are fixed-length periods (2 - 4 weeks) in which software features are developed and delivered.
- During a sprint, the team has daily meetings (**scrums**) to review progress and to update the list of work items that are incomplete.
- Sprints should produce a 'shippable product' i.e. complete and ready to deploy at the end of the sprint.



# Key roles in Scrum

- Product owner ← team lead (with feedback from TA aka user)
  - A team member who is responsible for identifying product features and attributes. They review work done and help to test the product.
  - In product development, the product manager should normally take on the Product Owner role.
- Scrum Master ← technical lead
  - A team coach who guides the team in the effective use of Scrum.
  - In many companies that use Scrum, the ScrumMaster also has some project management responsibilities.
- Development team
  - A small self-organising team of five to eight people who are responsible for developing the product.

# Scrums

- A **scrum** is a short, daily meeting that is usually held at the beginning of the day. The goal is information sharing among team members.
  - Team members share information, describe their progress since the previous day's scrum, problems that may have arisen and next plans.
  - Scrum meetings should be short and focused. They are sometimes organized as 'stand-up' meetings where there are no chairs.
  - During a scrum, the sprint backlog is reviewed. Completed items are removed from it. The team then decides if changes need to be made.

# How many sprints do you need?

- Products are developed in a series of sprints, each of which delivers an increment of the product or supporting software.
- The expectation is that you will require many sprints to complete your product.
  - Sprints are short duration (1-4 weeks) and take place between a defined start and end date.
  - Sprints are timeboxed, which means that development stops at the end of a sprint even if the work has not been completed.
- You often do not know how long unscheduled items will take to complete.
  - If you need to know this, you need to plan the time to do estimates of the work!

# Benefits of using timeboxed sprints

There is a tangible output (usually a software demonstrator) that can be delivered at the end of every sprint.

## **Demonstrable progress**



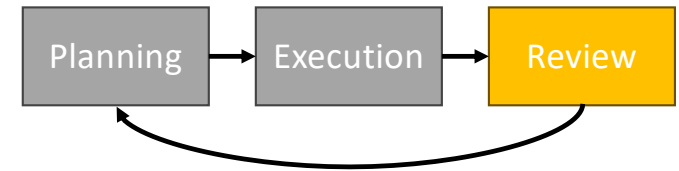
Time-boxing benefits

## **Problem discovery**

If errors and omissions are discovered the rework required is limited to the duration of a sprint.

## **Work planning**

The team develops an understanding of how much work they can do in a fixed time period.



## Phase 3: Sprint reviews

# Demo to a customer

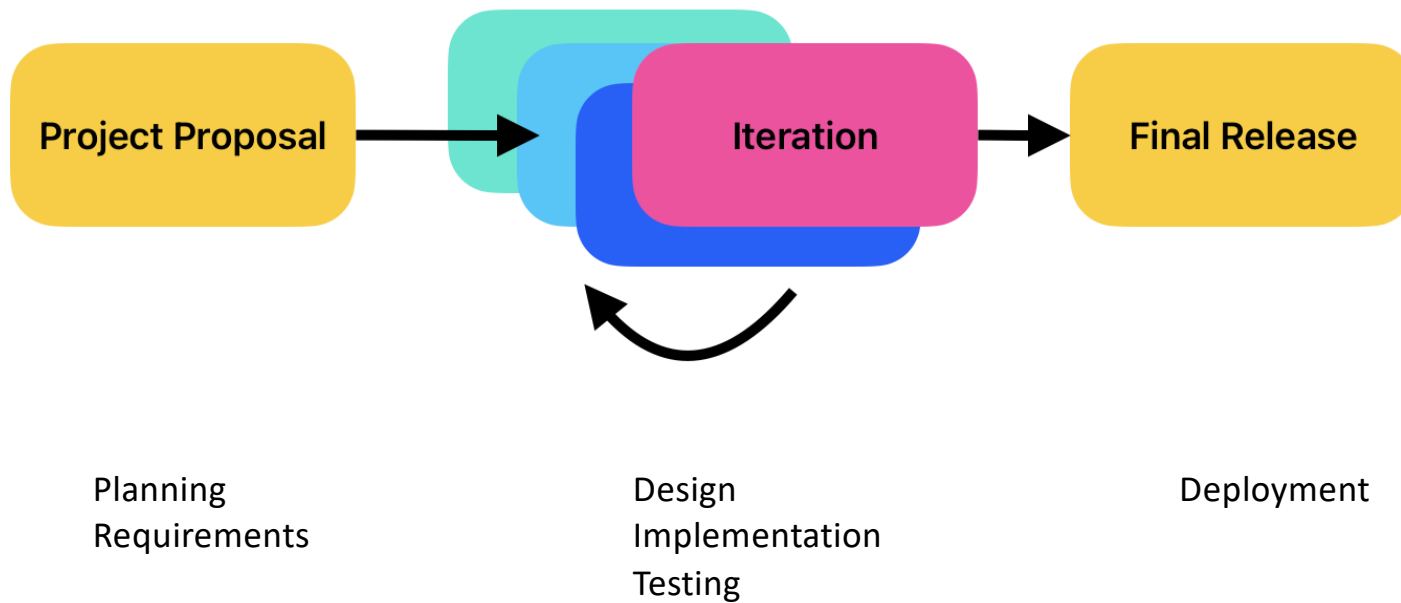
- Your sprint should always end with a demo to stakeholders
  - The customer being the most important stakeholder.
- Identify
  - What your goals were for the sprint?
  - Which of these goals were met (with a demonstration of functionality).
  - Which goals were not met. If they weren't met, how and when will you meet them?
- The goal is to get feedback!
  - Further refinements may be required.



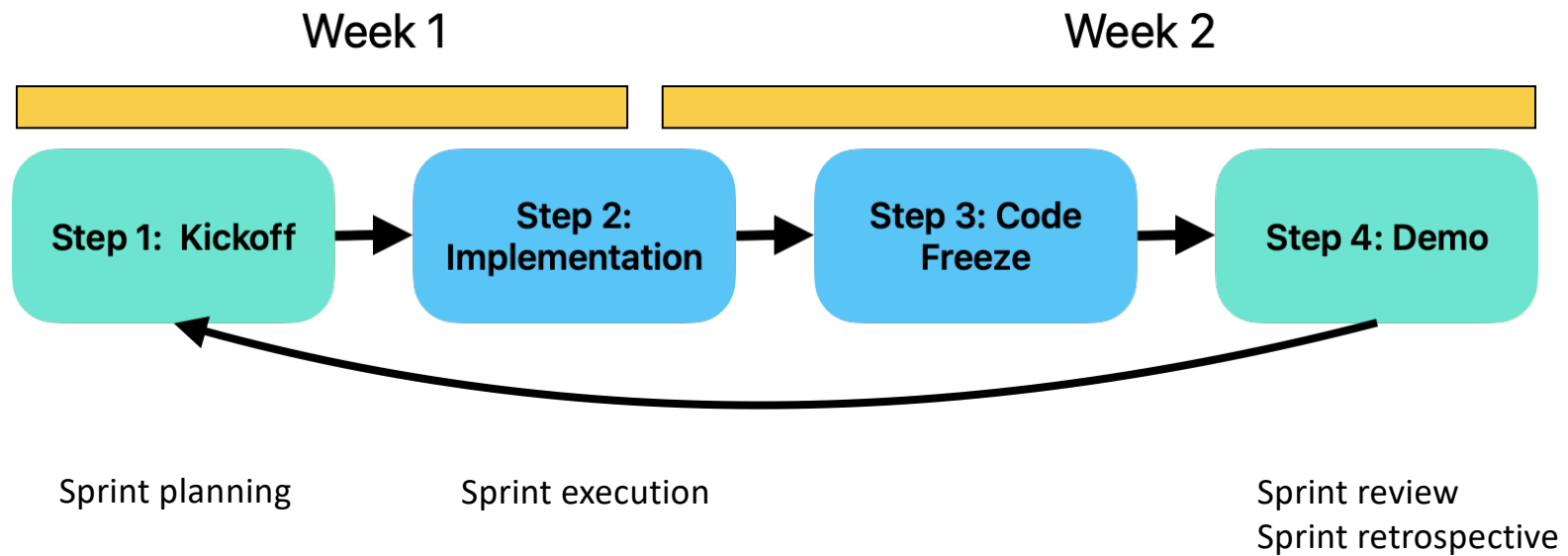
# Sprint retrospective

- At the end of each sprint – *after the demo* - there is a review meeting, which involves the whole team. The project team should:
  - Review whether the sprint has met its goal.
  - Discuss any new problems and issues that have emerged during the sprint.
  - Reflect on how they can improve the way they work.
- The sprint review should include a process review, in which the team reflects on its own way of working and how Scrum has been used.
  - The aim is to identify ways to improve and to discuss how to use Scrum more productively.
- This information feeds back into the next sprint's planning session!

# Project Structure



# *Recommended* Iteration Plan



# Reference

- Beck & Andres. 2004. [Extreme Programming Explained](#). Addison-Wesley Professional. ISBN 978-0134051994
- Schwaber & Sutherland. 2020. [The Scrum Guide](#). CC-licensed.
- Sommerville. 2021. [Engineering Software Products: An Introduction to Modern Software Engineering](#). Pearson. ISBN 978-1292376356.
- Shore & Warden. 2021. [The Art of Agile Development, 2nd Edition](#). O'Reilly. ISBN 978-1492080695.