Software Engineering

CS 346 Application Development

Credits

- Some content adapted from: Sommerville. 2021. Engineering Software Products: An Introduction to Modern Software Engineering. Pearson. ISBN 978-1292376356.

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, as well as developing the software development process itself.” -- Wikipedia

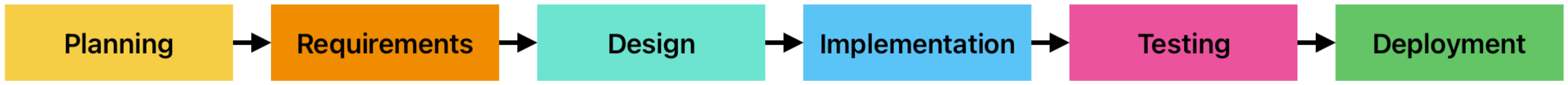
“The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.” -- SEBoK

Software projects

Start at the beginning…

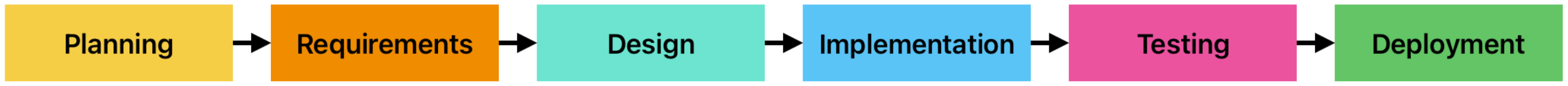
Software Projects

- We’re going to talk about designing, building, deploying software.
- When building software, we have a set of steps that we would normally follow. A software project is a planned activity, following these steps, which results in the delivery of a **software product.**
- The software development process is often represented like this, where each step is completed before the next one starts. The final step produces a software product:



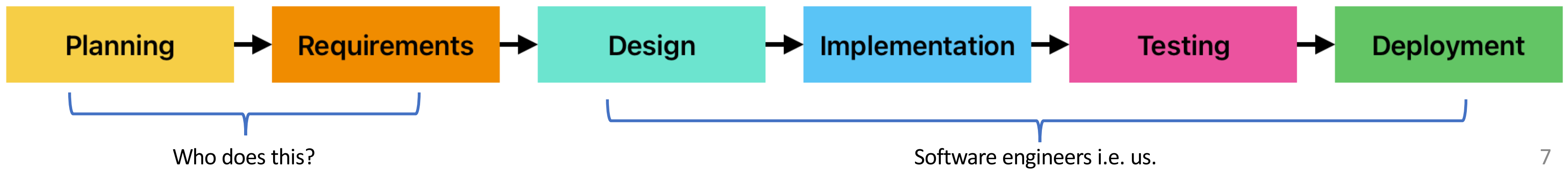
Steps in a Software Project

- **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 quickly and efficiently.
- **Testing**: Ensuring that it works before we sell it.
- **Deployment**: Delivering to customers (and getting paid).

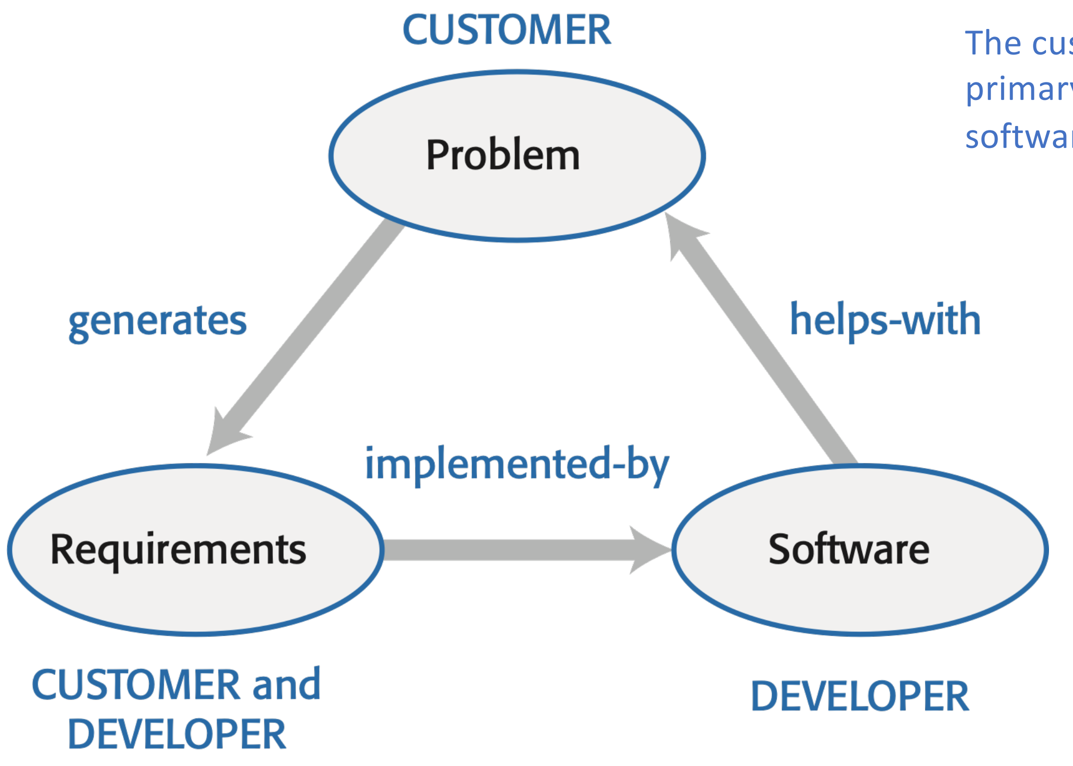


Software projects vs. products

- 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) and even phone apps and games (e.g. Flappy Bird).

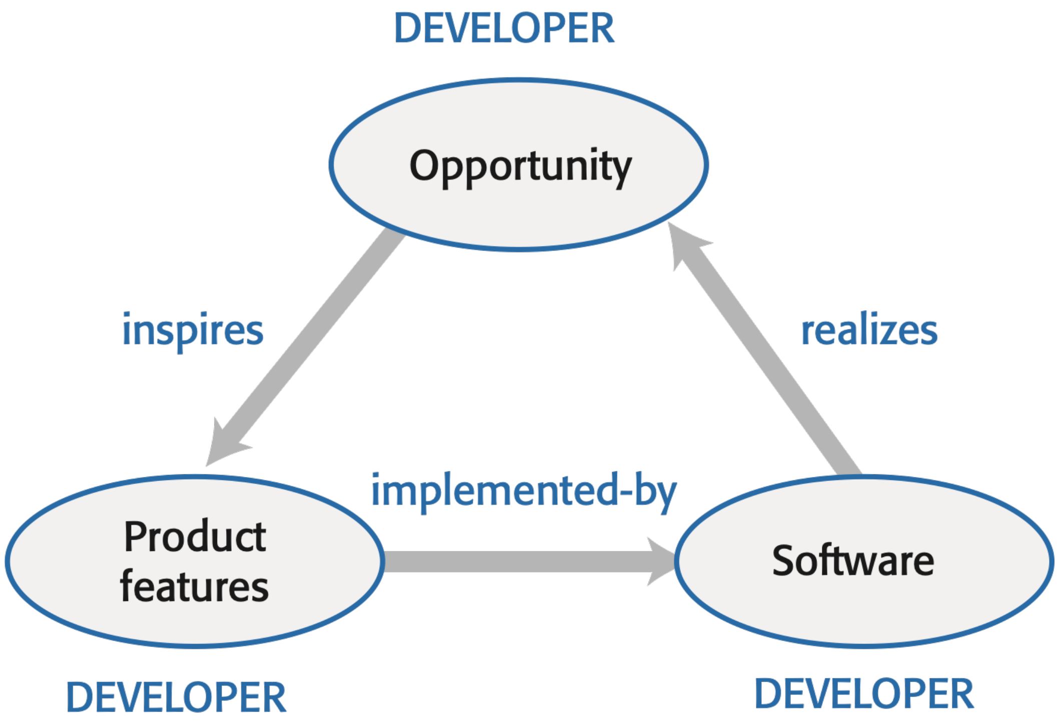


PROJECT software engineering



The customer is the primary driver of software requirements.

PRODUCT software engineering



Most commercial development now focuses on generic solutions. The developer decides who to target, and what features to deliver to them.

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.

Comparable Situations

- Similar situations exist for other types of software development:
  - **Student projects**: Individuals or student groups develop software as part of their course. Given an assignment, they decide what features to include in the software.
  - **Research software**: Researchers develop software to help them answer questions that are relevant to their research.
  - **Internal tool development**: Software developers may develop tools to support their work - in essence, these are internal products that are not intended for customer release.

Product Development Decisions

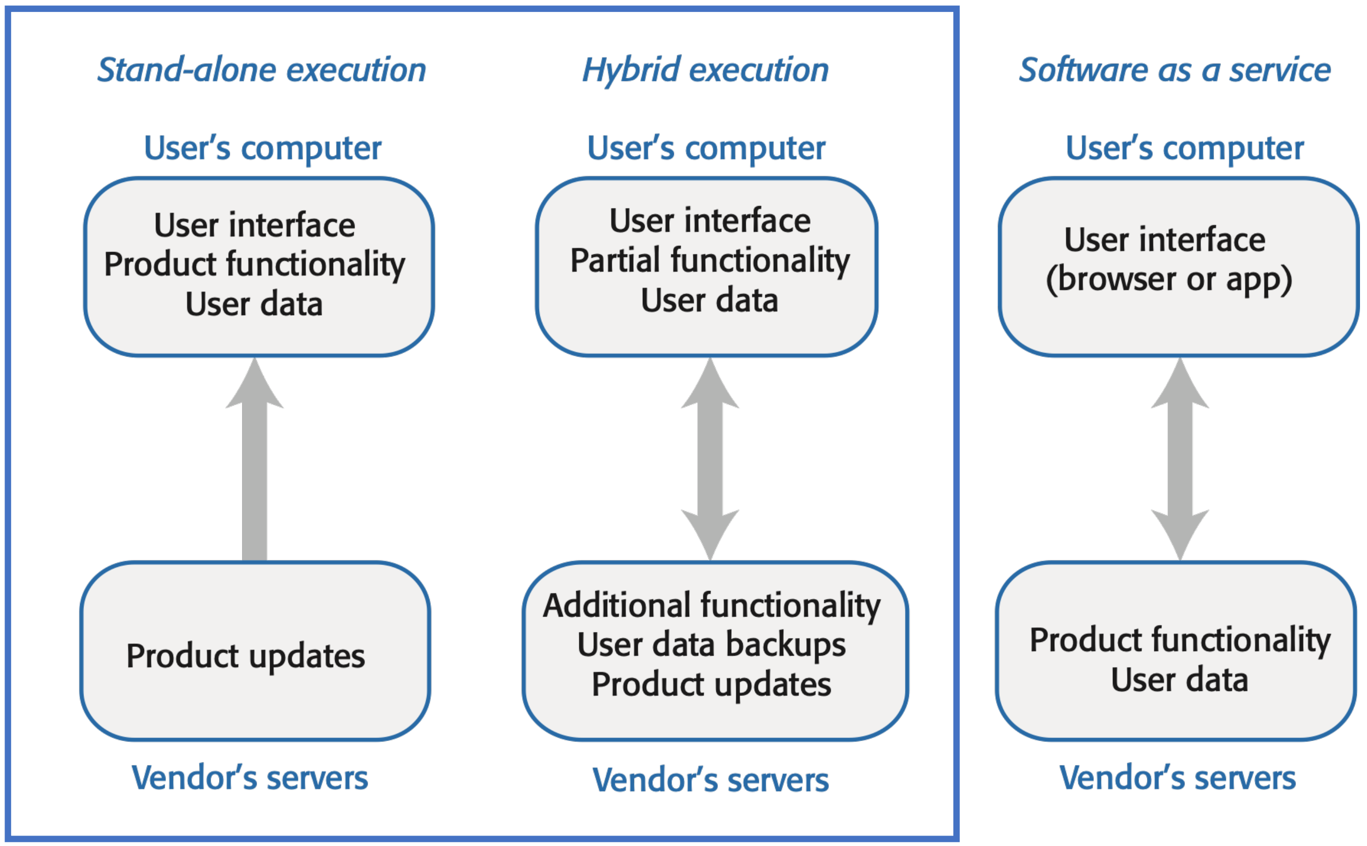
- The developer needs to make product decisions that drive the product direction:
    - **Platform**: A software (or software + hardware) product that includes functionality so that new applications can be built on it. e.g., Facebook with ‘Facebook Apps’, or Playstation, or Windows are all platforms.
    - **Software product line**: A set of software products that share a common core. Each member of the product line includes customer-specific adaptations and additions for requirements that a generic product couldn’t meet.
        - e.g., health care systems that need custom integrations.
        - e.g., CarPlay audio integration into car systems.
- External customers have little impact (although the developer may try and find prospective customers and ask for input!)

Software Execution Models

- **Stand-alone**: The software executes entirely on the customer’s computers. e.g., calculator, MS word, VS Code.
- **Hybrid**: Part of the software’s functionality is implemented on the customer’s computer, but some features are implemented on remote servers e.g., a collaborative drawing application, or a coop online game.
- **Software service**: All the product’s features are implemented on the developer’s servers and the customer accesses these through a browser or a mobile app. e.g., Gmail, Instagram.

- These are primarily product decisions, since they depend on what features you need to implement. (More on this later).

Software execution models



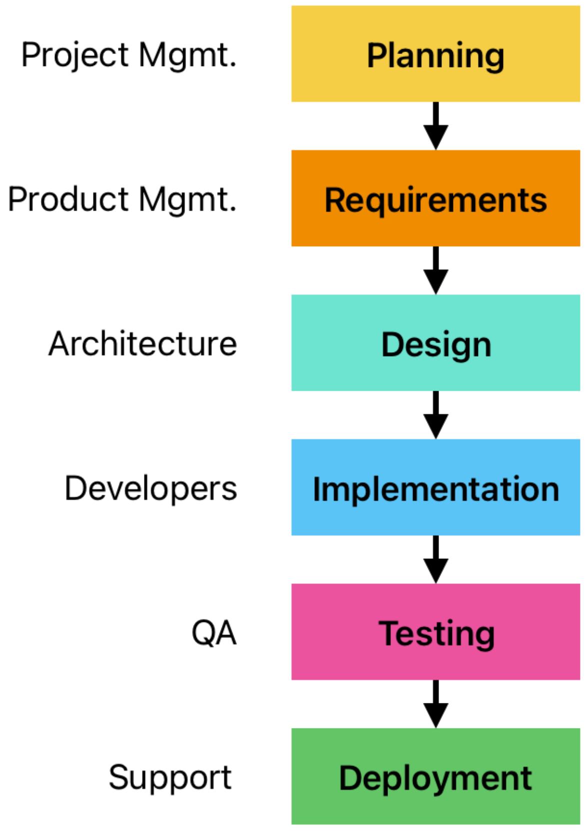
Software Process Models

Let’s think more about building products.

Software process models

The linear model we presented was a very common view of software development until recently. It is also known as the **Waterfall Model** (Royce 1970):

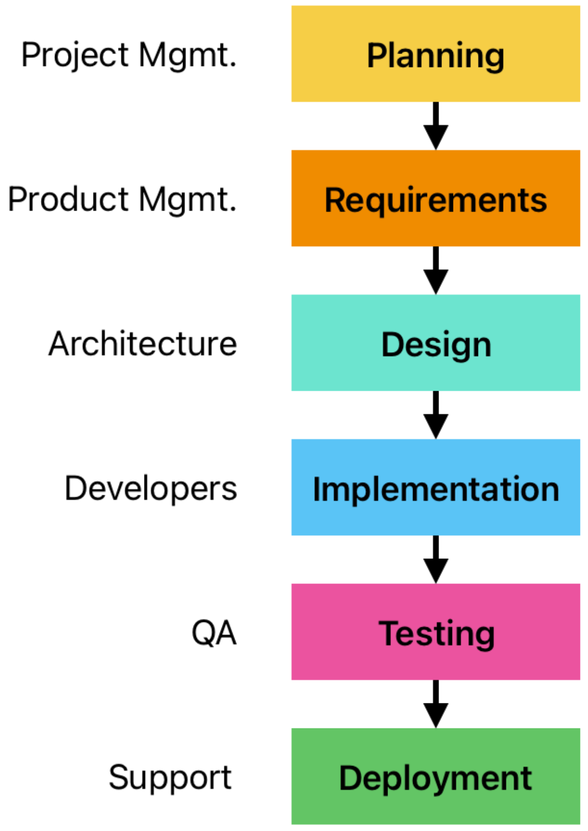
- Each step is “owned” and managed by a separate person or group.
- Each step represents significant work.
- Steps need to be performed in order.
- Gatekeeping enforced e.g., requirements approval before design.



Challenges

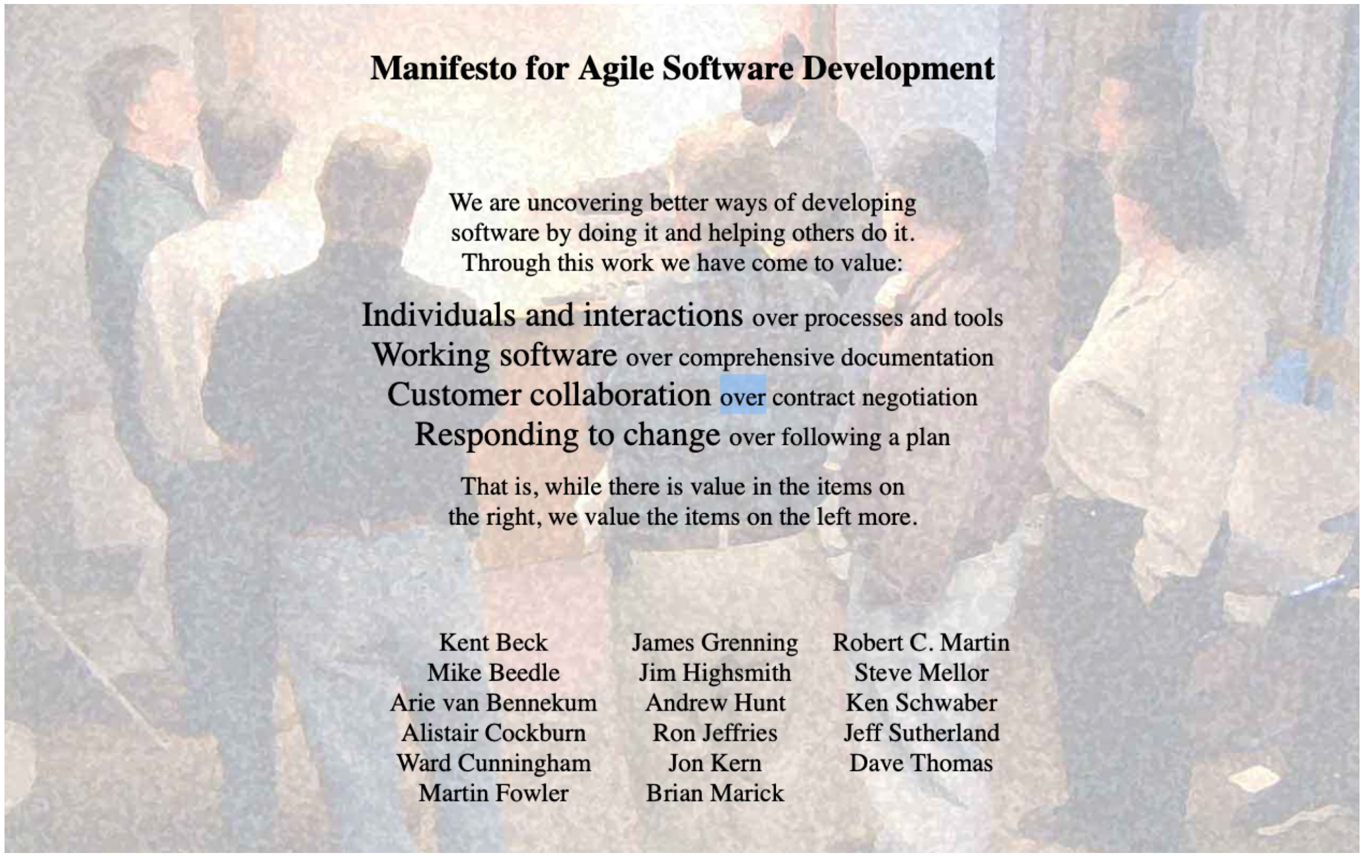
This doesn't work very well. Why?

- 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	

Manifesto for Agile Development

We are uncovering better ways of developing
software by doing it and helping others to 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 on the items on
the right, we value the items on the left more.

Agile software engineering

- Virtually all modern software products are developed using an Agile approach.
- Software products must be brought to market quickly
  - Rapid software development and delivery is essential.
  - Need to be flexible (*“Agile”, get it?*)
- Agile software engineering focuses on delivering functionality quickly, responding to changing product specifications and minimizing development overheads.
- 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

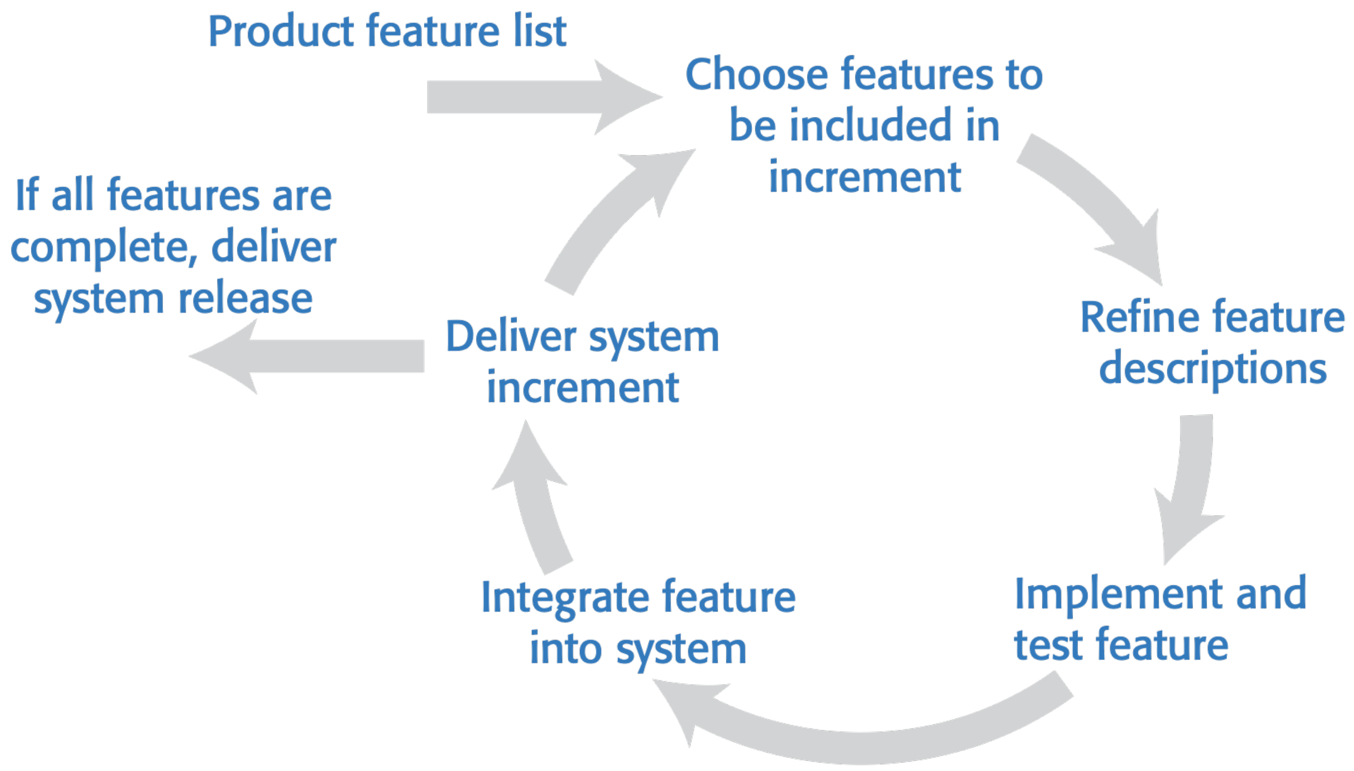
Agile methods

- Plan-driven development evolved to support the engineering of large, long-lifetime systems (such as aircraft control systems) where teams may be geographically dispersed and work on the software for several years.
  - This approach is based on controlled and rigorous software development processes that include detailed project planning, requirements specification and analysis and system modelling.
  - However, plan-driven development involves significant overhead and does not support the rapid development and delivery of software.
- 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.

Key concept: Incremental development

- All agile methods recognize the importance of incremental development and delivery.
- Product development focuses on the 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 surrogate users can try it out and provide feedback to the development team. You then go on to define and implement the next feature of the system.

Incremental development


Product feature list
Choose features to be included in increment
Refine feature descriptions
Implement and test feature
Integrate feature into system
Deliver system increment
If all features are complete, deliver system release

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.

Agile development principles

- ***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 development principles

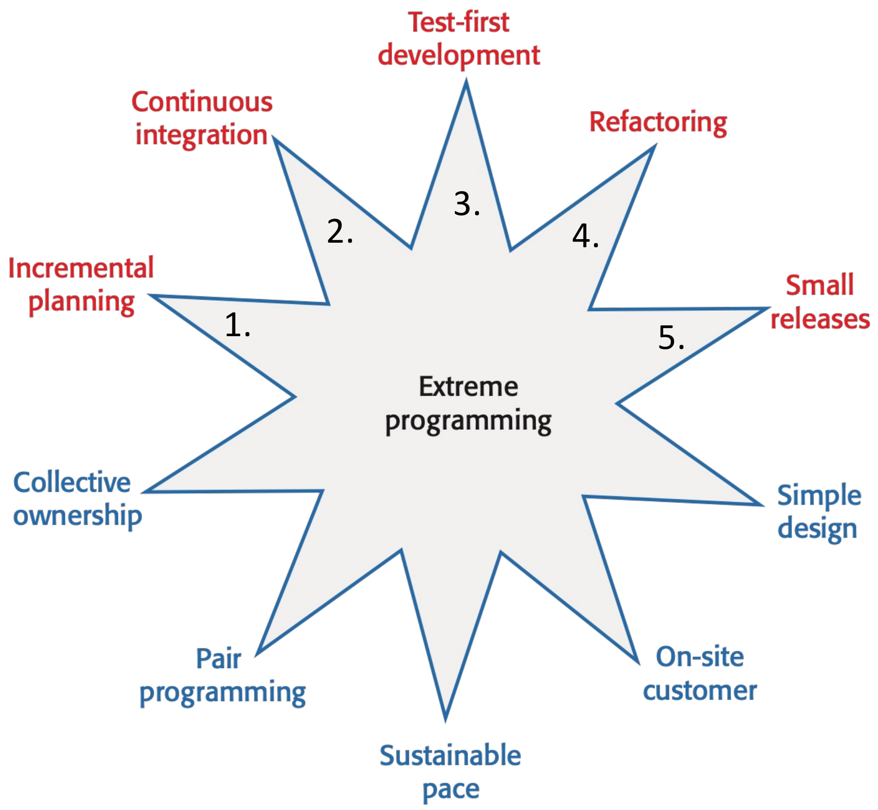
- ***Maintain simplicity***
  Focus on simplicity in both the software being developed and in the development process. Wherever possible, do what you can to eliminate complexity from the system.
- ***Focus on people, not things***
  Trust the development team and do not expect everyone to always do the 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)

Extreme programming (XP)

- The most influential work that has changed software development culture was the development of **Extreme Programming (XP)**.
- The name was coined by Kent Beck in 1998 because the approach was developed by pushing recognized good practice, such as iterative development, to ‘extreme’ levels.
- Extreme programming focussess on 12 development techniques that are geared to rapid, incremental software development, change and delivery.
  - Very much focused on 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 and a new version of the system is created. 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 incrementally add functionality.

We will revisit most of these later.

Scrum

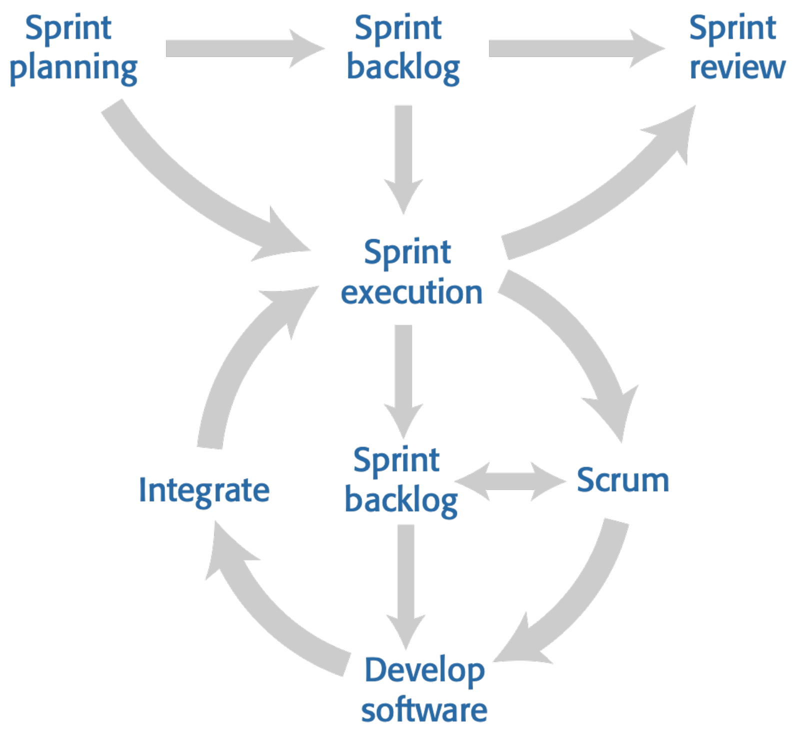
Scrum

- 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 and planning.
  - 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 iterations**
  - 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



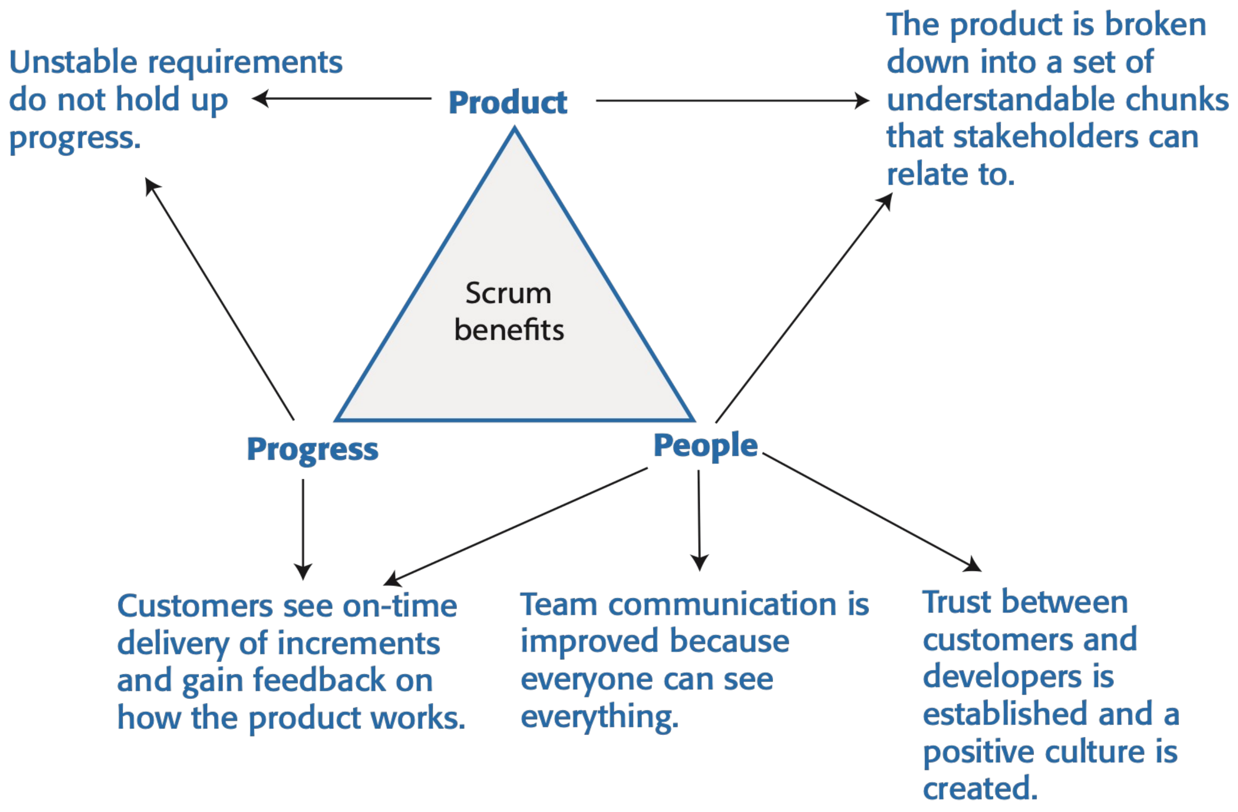
A sprint is 2-4 weeks in length.

- **Spring planning**: the team decides on goals and what to accomplish.
- **Sprint execution**: the team works on features, bugs, other assigned work.
- **Sprint review**: review the outcomes with a customer. The team also meets to review progress.

Sprint activities

- **Sprint planning**
  - Work items to be completed in that sprint are selected and, if necessary, refined to create a sprint backlog. This should not last more than a day at the beginning of the sprint.
- **Sprint execution**
  - The team work to implement the sprint backlog items that have been chosen for that sprint. If it is impossible to complete all of the sprint backlog items, the sprint is not extended. The unfinished items are returned to the product backlog and queued for a future sprint.
- **Sprint reviewing**
  - The work done in the sprint is reviewed by the team and (possibly) external stakeholders. The team reflect on what went well and what went wrong during the sprint with a view to improving their work process.

The top five benefits of using Scrum


Product
Unstable requirements do not hold up progress.
The product is broken down into a set of understandable chunks that stakeholders can relate to.
Scrum benefits
Progress
People
Customers see on-time delivery of increments and gain feedback on how the product works.
Team communication is improved because everyone can see everything.
Trust between customers and developers is established and a positive culture is created.

Phase 1: Sprint planning

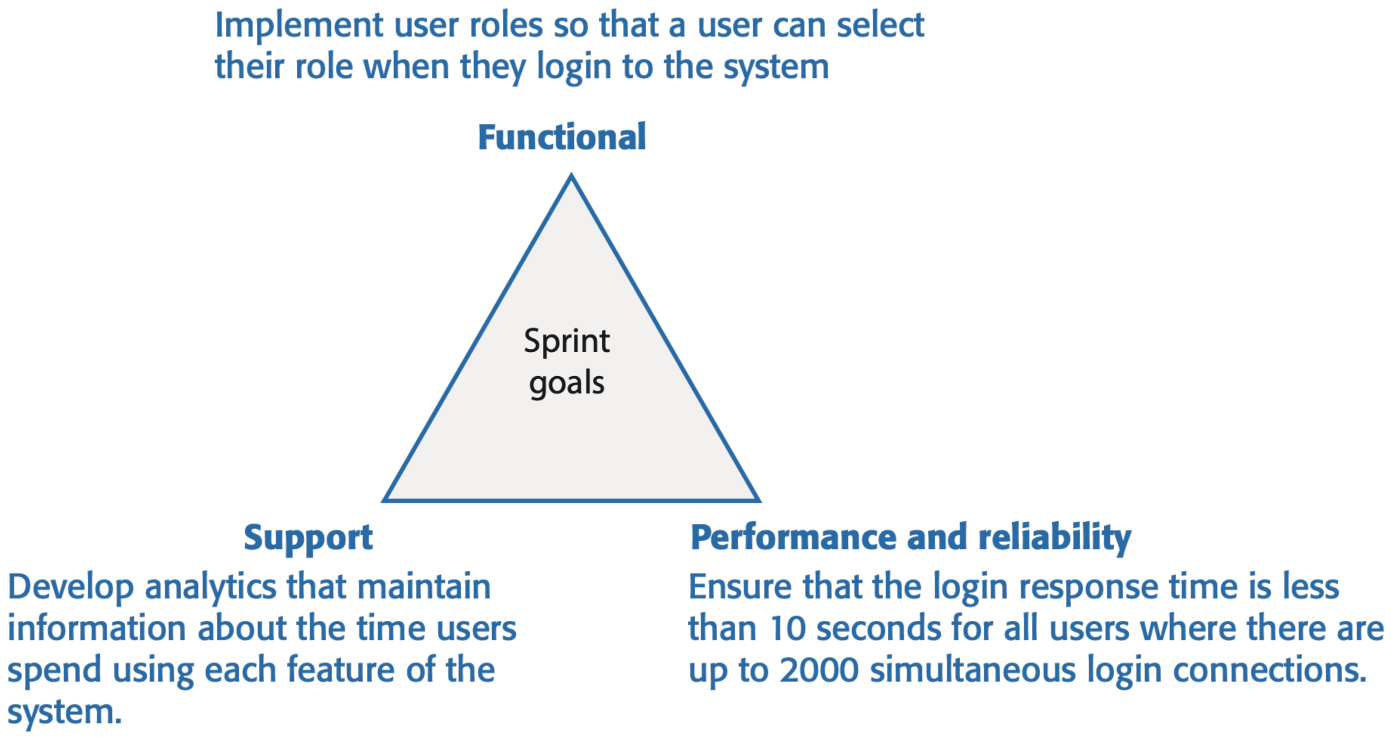
Sprint planning

- In a sprint plan, the team decides which items in the product backlog should be implemented during that sprint.
  - Key inputs are the effort estimates associated with PBIs and the team's velocity (i.e. *how much work they historically get done in a sprint*).
- The output of the sprint planning process is a sprint backlog.
  - The sprint backlog is a breakdown of PBIs to show the what is involved in implementing the PBIs chosen for that sprint.
- During a sprint, the team has daily meetings (scrums) to coordinate their work.
  - Check-in with each other, identify problem areas to address.

Sprint planning

- Establish an agreed sprint goal
  - Sprint goals may be focused on software functionality, support or performance and reliability,.
- Decide on the list of items from the product backlog that should be implemented
- Create a sprint backlog.
  - This is a more detailed version of the product backlog that records the work to be done during the sprint

Sprint goals


Implement user roles so that a user can select their role when they login to the system
Functional
Sprint goals
Support
Develop analytics that maintain information about the time users spend using each feature of the system.
Performance and reliability
Ensure that the login response time is less than 10 seconds for all users where there are up to 2000 simultaneous login connections.

Planning Activities

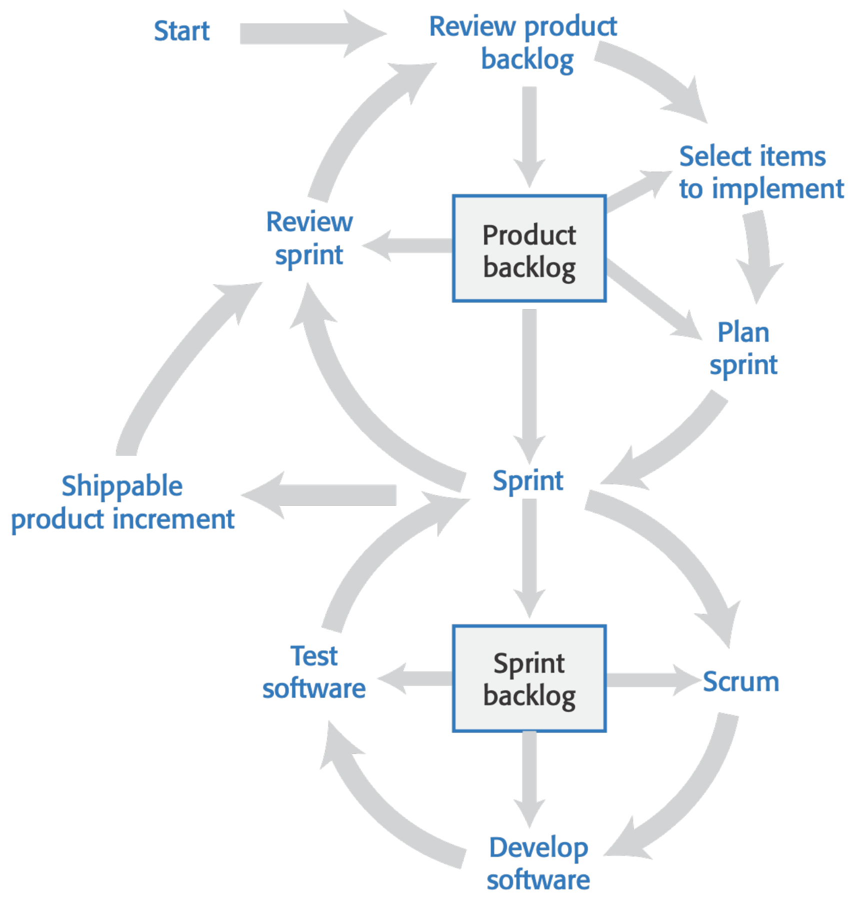
You work on product backlog items (PBIs) during sprint planning, or as a scheduled activity.

- **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. This may lead to the creation of new product backlog items.
- **Estimation:** The team estimates the amount of work required to implement a PBI and adds this assessment to each analyzed PBI.
  - 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

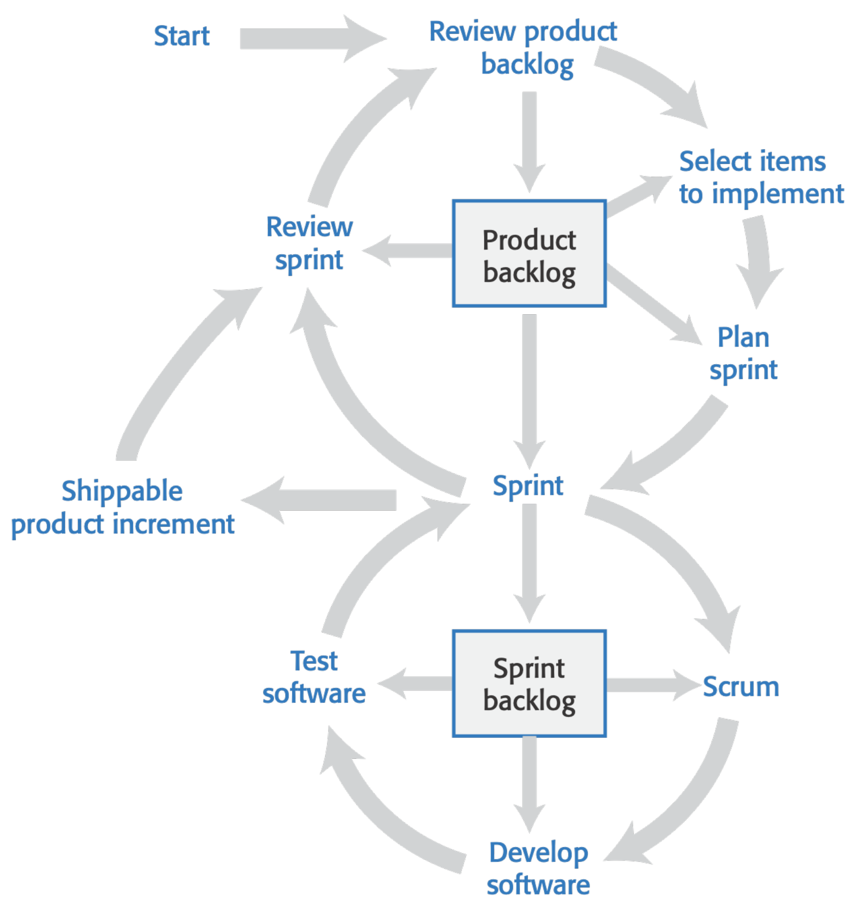
Scrum execution

- **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.



Scrum practices

- **Product backlog**
  - A list of items such as bugs, features that the team has not yet completed.
- **Sprint backlog**
  - The list of items that the team has agreed to complete for the sprint.
- **Scrum**
  - Daily team meeting to review progress.
- **Shippable product increment**
  - The output which should be of high enough quality to be deployed.



Key roles in Scrum

Your team lead is the ScrumMaster.

- Product owner
  - 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.
- ScrumMaster
  - 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.

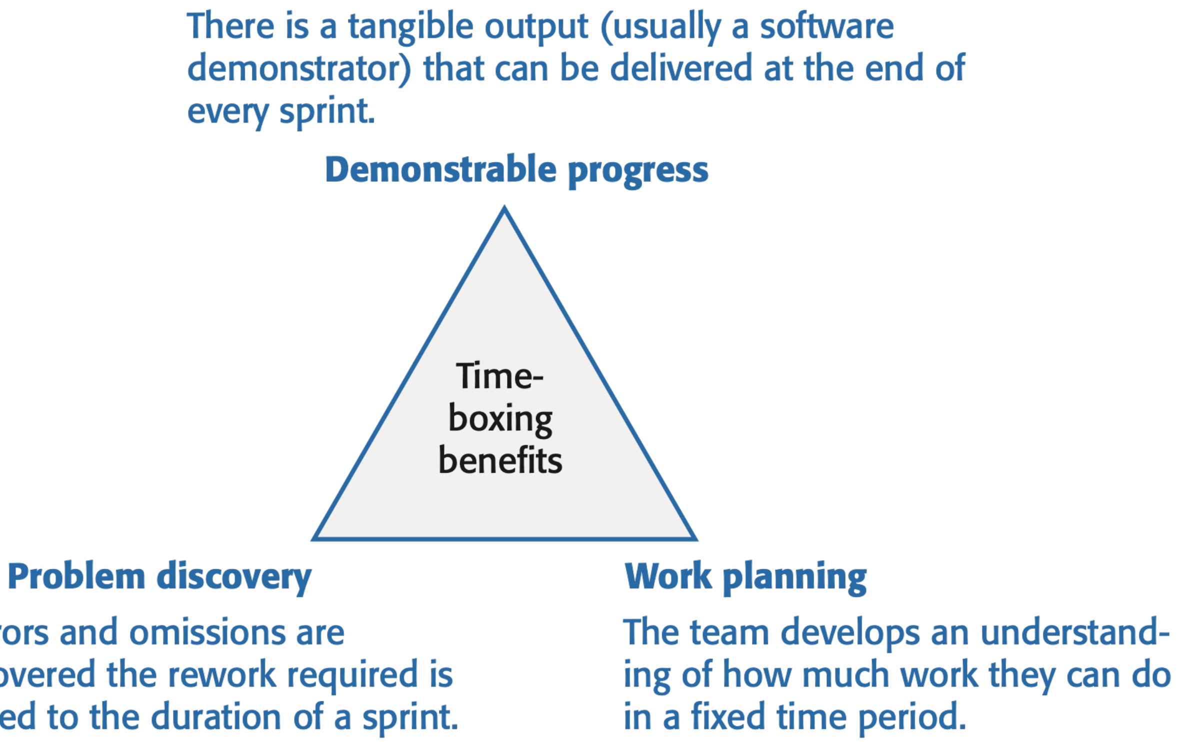
Scrums

- A scrum is a short, daily meeting that is usually held at the beginning of the day. During a scrum, all team members share information, describe their progress since the previous day’s scrum, problems that have arisen and plans for the coming day. This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them.
- Scrum meetings should be short and focused. To dissuade team members from getting involved in long discussions, they are sometimes organized as ‘stand-up’ meetings where there are no chairs in the meeting room.
- During a scrum, the sprint backlog is reviewed. Completed items are removed from it. New items may be added to the backlog as new information emerges. The team then decide who should work on sprint backlog items that day.

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, many sprints to complete your product.
  - Sprints are short duration activities (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 whether or not the work has 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.

Code completeness checklist

- ***Reviewed***
  The code has been reviewed by another team member who has checked that it meets agreed coding standards, is understandable, includes appropriate comments, and has been refactored if necessary.
- ***Unit tested***
  All unit tests have been run automatically, and all tests have executed successfully.
- ***Integrated***
  The code has been integrated with the project codebase and no integration errors have been reported.
- ***Integration tested***
  All integration tests have been run automatically, and all tests have executed successfully.
- ***Accepted***
  Acceptance tests have been run if appropriate and the product owner or the development team have confirmed that the product backlog item has been completed.

Phase 3: Sprint reviews

Demo to a customer

- Your sprint should always end with a demonstration to stakeholders (including the customer).
- 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 reviews

- At the end of each sprint, there is a review meeting, which involves the whole team. This meeting:
  - reviews whether the sprint has met its goal.
  - sets out any new problems and issues that have emerged during the sprint.
  - is a way for a team to 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.

Summary 1

- An effective way to develop software products is to use agile software engineering methods that are geared to rapid product development and delivery.
- Agile methods are based around iterative development and the minimization of overheads during the development process.
- Extreme programming (XP) is an influential agile method that introduced agile development practices such as user stories, test-first development and continuous integration into the mainstream.
- Scrum is an agile method that focuses on agile planning and management. Unlike XP, it does not define the engineering practices to be used.

Summary 2

- In Scrum, work to be done is maintained in a product backlog – a list of work items to be completed. Each increment of the software implements some of the work items from the product backlog.
- Sprints are fixed-time activities (usually 2–4 weeks) where a product increment is developed. Increments should be ‘potentially shippable’ i.e. they should not need further work before they are delivered.
- A self-organizing team is a development team that organizes the work to be done by discussion and agreement amongst team members.
- Scrum practices such as the product backlog, sprints and self-organizing teams can be used in any agile development process, even if other aspects of Scrum are not used.

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.