# Software Development Processes

Software development processes are essentially different ways to organize the activities of the software development life cycle mentioned in the previous reading. As mentioned there, each software process does all of the activities listed but in different orders, frequencies, and often with different people involved with the various tasks.

There are two categories of software development processes, plan driven and incremental.

Plan driven processes involve following a well thought out plan made at the beginning of the project, whereas incremental processes focus on building software in short increments of planning, developing, and demoing new parts of the software. By demoing new parts of the software to key stakeholders feedback is gathered in order to course correct along the way.

Let’s take a look at both in more detail:

# Plan Driven / Waterfall

Plan Driven Development, also known as Waterfall Development is characterized by its doing and finishing of each Software Engineering activity at a time before moving onto the next, with the result of each step flowing forward like a waterfall, as shown in Figure 1.

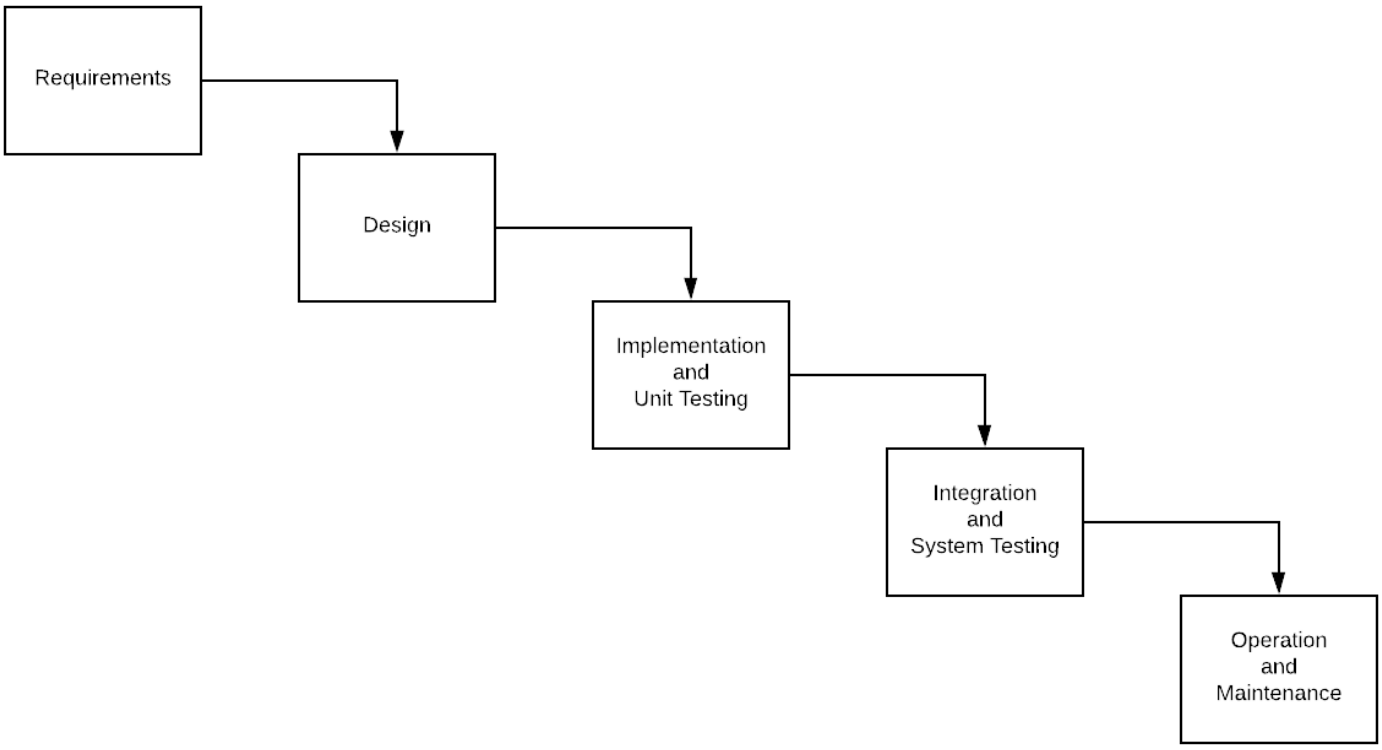


*Figure 1: Waterfall Development Activities*

You'll notice that it basically follows the activities listed in the previous reading in roughly the order they are listed, and progress is measured against the plan.

The idea is first turned into a vision.

That vision is analyzed to see whether it is worth pursuing in terms of the amount of resources (time, money, people) it will require for the amount of benefit it will bring.

# Requirements

If the project passes vision analysis phase, analysts begin breaking down the vision into a large and detailed set of requirements called a Software Requirements Specification. This document might then be presented to stakeholders for feedback and approval.

# Design

After this document is finished the design phase begins in which the look and feel is prototyped or mocked up and some of the important software architecture structures are established, often all contained in a design document. Again, once finished the design of the project will get feedback and be approved or redone.

## Development

Development work will then begin, and the project is coded, followed by another sequence of approval and feedback.

## Release And Maintenance

When coding is finished, the project is released/deployed and enters its final phase of support and maintenance.

## Waterfall in Modern Software

This process is fairly rigid and not well suited for when there are any required changes stemming from a stage that has already been completed.

However this process can work well for projects that have a very robust and non changing set of requirements that requires approval at every step, and still sees use in the military and medical fields.

Up until fairly recently, waterfall was about the only process that existed. When projects failed or required major changes it was generally thought that the requirements or design phase had not been complete enough, had missed something, or had been done incorrectly.

However as more and more technologies were introduced that caused computer usage to be widespread and caused applications to vastly increase in complexity, many waterfall projects kept failing.

A significant portion of projects were being a cancelled outright, and a majority that were released were significantly late, over budget, or both. Many people in the software industry could see that there was a problem with the way software was being developed, and that changes needed to made.

Waterfall required a large amount of time up front planning many key decisions before real users ever got a chance for meaningful feedback. What software developers wanted was the opportunity to make changes to the project stemming from feedback from real users.

# Incremental Software Development

Throughout the 90s, a new set of processes began to emerge that focused on being “lighter-weight” and more easily adaptable to change than plan driven techniques. These techniques included:

- Rapid Application Development (RAD)

- Unified Process (UP)
- Extreme Programming (XP)
- Scrum

Note: In this class we will primarily be using Scrum which has picked up some things from the others, especially XP. Scrum is probably the most popular development process used in the field today by far, even over Waterfall.

In 2001, 17 professional software developers, many of them instrumental in creating these new development processes met together in order to share insight into what they had learned from these new processes as well as codify a set of values and principles for guiding the development of the new processes. The document that this meeting produced is titled “Manifesto for Agile Software Development”.

# The Agile Manifesto:

This document starts by outlining 4 "key values" that all Agile projects should adhere to:

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

One thing that many people get wrong in their understanding of these values is that while the authors of the Agile Manifesto value the bolded items on the left more than the items on the right, they are by no means arguing that those on the right side should be or even can be eliminated from making software. Instead they are suggesting that when given a choice between these two options, projects should lean towards the left when possible.

Let's break down each of these values and attempt to consider them within the software development world that the authors were facing:

## Individuals and interactions over processes and tools

What this value is saying is that when managing a project, we should be doing everything we can to make sure that we are attempting to create meaningful interactions between individuals within the team. This is the most useful way to make information flow. It just so happens that having strict processes for communication such as long structured meetings is one of the worst ways for information to flow. While there exist many processes and tools to "enhance" communication and interactions, nothing is going to beat direct face to face communication.

## Working software over comprehensive documentation

This stems directly from issues that the authors had regarding the Waterfall process. In Waterfall, there is a reliance on producing a huge amount of comprehensive documentation about how software is going to work, before any software is going to work. Agile is about providing small amounts of usable software at a time and presenting it to customers for validation and clarification. If they got something wrong, they are quickly able to adapt to new directives.

## Customer collaboration over contract negotiation

In Waterfall, customers were mostly involved in the early stages, establishing a very complex contract identifying everything they wanted from the software. Often the result they were presented with at the end did not meet their expectations. This is in part due to the fact that customers often don't fully know what they want until they start seeing examples of the product. In order to change this, Agile attempts to keep customers involved at every step of the way, routinely getting their feedback on demonstrable software.

## Responding to change over following a plan

With Waterfall, the farther along a project was, the more difficult it became to make a change. Furthermore, there was a high degree of resistance to changes that were not in the carefully constructed plan. The problem with this is that outside of some really specific applications, change is a huge part of every software project. The Agile Manifesto authors wanted to make sure to keep plans light and fluid in order to keep up with the changing demands of the customers.

## Agile's 12 Principles

In addition to the key values, the authors of the Agile Manifesto also created a set of 12 more actionable principles.

1. The highest priority is to satisfy the customer through early and continuous delivery.
2. Welcome changing requirements, even late in development.
3. Deliver working software frequently, from a couple of weeks to a couple of months.
4. Stakeholders and developers must collaborate on a daily basis.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. Face-to-face meetings are deemed the most efficient and effective format for project success.
7. A final working product is the ultimate measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—maximizing the work not done—is an essential element.
11. The best architectures, requirements, and designs emerge from self-organizing teams.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Essentially these principles reinforce the concepts in the key values listed above. They encourage customer involvement, freedom for the development team, and opportunities for process improvement.

It is important to note that what is listed here makes up all that Agile is, a set of values and actionable principles. Agile is not a software development process, it is meant to guide software development processes. Think of it as sort of an interface rather than a fully implemented object.

Excluding Waterfall, almost all of the popular development processes in use today are based on these Agile values and principles. In this class, we will be using Scrum, the most popular Agile based approach. While Scrum was created before the Agile Manifesto, the creators of Scrum were part of the Agile Manifesto authors. Therefore parts of Agile are based on Scrum, and future changes to Scrum were based on Agile.

# Incremental Versus Waterfall

Waterfall certainly has its problems and may not be suitable for a vast majority of software projects, however this doesn't mean that Waterfall is bad or useless. Waterfall continues to find use in areas of applications that have a very strict set of well defined requirements, little change in feature expectation, and a need for approval at every step of the software development lifecycle. For example, a missile guidance system would be a terrible fit for an incremental process and a great fit for a Waterfall one, as would a pacemaker device. There are many companies that exist that still use Waterfall, or aspects of Waterfall mixed with some incremental practices, and it is fully possible that they are making the correct choice.

# Scrum

In this class we will be using a modified version of Scrum for our development process. Remember, Agile processes do not need to be rigidly adhered to "Individuals and interactions over processes and tools" and instead the focus should rightfully be on making the process work for you and your company.

In this case, working with students is quite different than working with real world employees, and so trying to fully apply a system meant for employees who are paid and can be fired would not work particularly well.

The areas where what we do in this class is different than what you could expect in industry Scrum will be identified as they come up.

There will be additional readings going into detail about the Scrum and it’s various activities within Scrum but an overview of the key features is provided below.

# Scrum Activities:

This first difference between how Scrum is implemented in this class and how it is implemented in industry is illustrated in Figures 2 and 3. In these figures you can see that the team selection may take place well after the vision, analysis, and product backlog are formed. This is largely due to the fact that in industry these tasks are often handled by business analysts and possibly executives in the company.

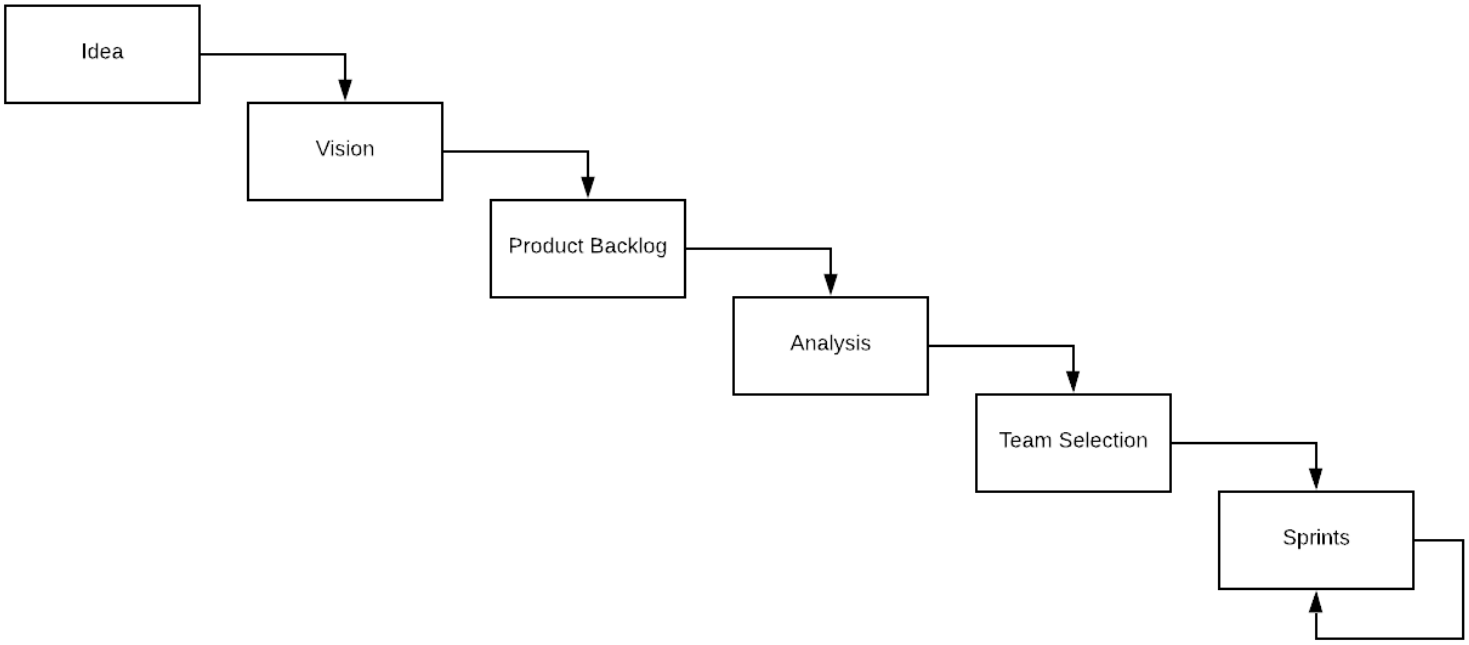


*Figure 2: A typical order of activities for Scrum in industry.*

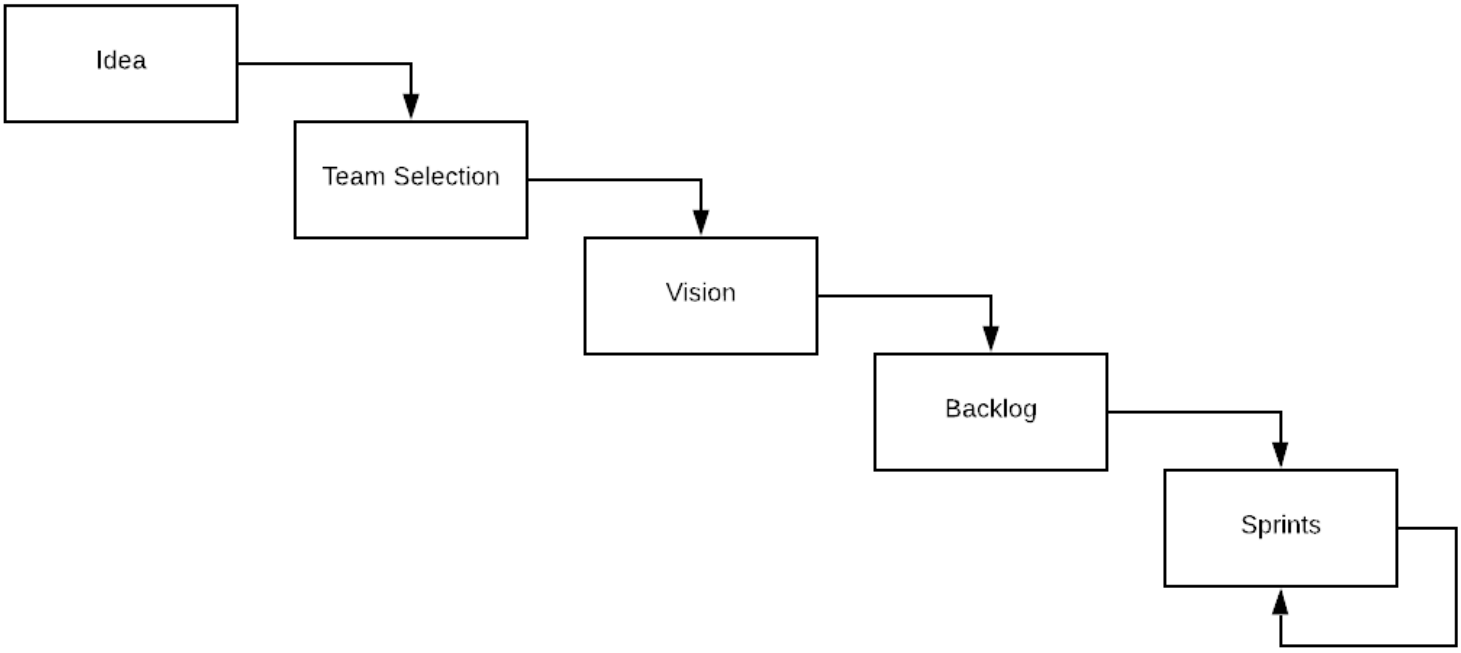


*Figure 3: The order of Scrum activities in this class.*

While this may look similar to Waterfall, at the time development begins during the sprints, only some light requirements have been done, with no design work.

# The Product Idea

In both industry and in class, the project begins with a stakeholder having an idea for a project. In industry, this idea belongs to someone usually high up in the corporate hierarchy, whereas in class, this comes from a member of the future development team.

For much more information on project ideas see the reading from week 1 on “App Ideas”.

# The Product Vision

In industry, this idea is then elaborated into a vision. This vision clearly defines what the app is, what it isn’t, who it appeals to, its competitors, and its primary features. This is extremely important for allowing other people besides the idea originator to gain an understanding and contribute to what will be involved in implementing this idea.

In class, we do the vision phase after team selection because we want to make sure teammates have a chance to help shape and improve the initial app idea, as well as to increase early familiarity with it. This helps spread the idea ownership amongst the whole team, and reduce reliance on the idea originator to do all the early heavy lifting.

# Product Backlog

After the vision is created, in industry the product backlog is created by the idea originator, referred to in Scrum as the Product Owner. The product backlog is essentially a prioritized list of things called “User Stories” that the product seeks to accomplish. The next reading will be all about Product Backlogs and User Stories but for now think of a User Story as a written statement that says

“I want to be able to _____ with the app.”

For example:

- “I want to be able to search for a hotel room.”
- “I want to be able to book a hotel room.”

This is an oversimplification of all the information that is contained within a User Story but is good enough to get an overall understanding of Scrum.

In industry the product owner creates and prioritizes all of the initial User Stories, as well as provides early estimates as to the relative difficulty of implementing each one.

In the class, your second group assignment (after creating the Vision Document) will be to create this backlog together. This will further give you familiarity with what the project is going to be, give you experience with creating and discussing User Stories, as well as planning, prioritizing, and estimation.

## Estimation and Analysis

In industry, now that the vision and backlog have been established and there has been some estimation done, there is typically a round of analysis to decide whether it worth moving forward with the project before development begins.

This is not done in our class, we partially rely on you to submit reasonable ideas during the pitch phase, and tend to give feedback when an idea does not seem possible within the scope of the class. Occasionally some projects make it through to the pitch presentation that are not reasonable, and this factors into how votes are processed. This is necessary to create positive outcomes for a majority of the teams in the class.

## Team Selection

In industry, after the analysis phase the teams are actually formed that will work on the app. We will go into detail on how teams are formed and how they should work in a separate reading. Team sizes tend to be small, from 3-8 people to work on a project. Typically careful thought is put into the composition of a team, you want to have some specialists and some generalists, and members from diverse backgrounds to encourage a wide range of perspectives.

As for how the team selection process works in this class, students indicate via voting which projects they are most interested in, and teams are formed around projects which receive the most interest. Unfortunately, there is simply not enough time, available information, or previous experience to do the type of team building that happens in industry. Placing people on a project that they are interested in greatly increases the chances they will become committed to working hard on the project.

## Sprints

In industry, after team selection, the sprints begin. Sprints are a Scrum term that refer to a fixed period of time (usually from 2 to 6 weeks) in which team members select a subset of the product backlog and commit to building it into a demonstrable/shippable increment of the software project.

The Scrum Framework is summarized in Figure 4.

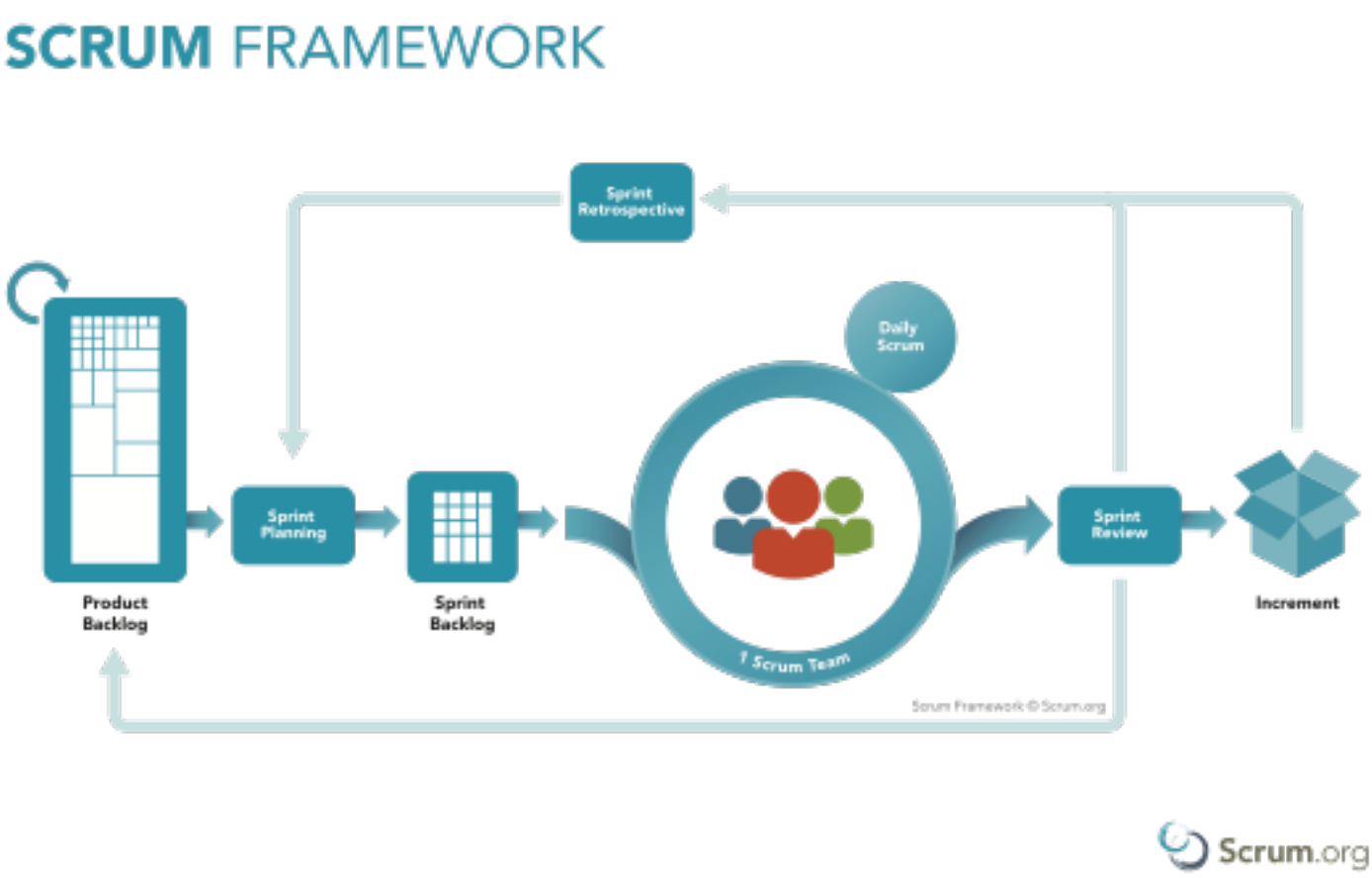


*Figure 4: The Scrum Framework [Image Credit: Scrum.org].*

There will be several future lectures on all of the activities of the scrum framework, but a high level overview is provided below.

A sprint begins with a period of sprint planning where a subset of the product backlog is selected to become part of the sprint backlog. Additionally, a clear goal for what the end product of that sprint will be is developed.

Each day after the sprint begins team members hold a brief “daily standup” or "daily scrum" in which every team member answers the following three questions:

- What have you done since the last meeting?
- What are you going to do before the next meeting?
- What are the obstacles blocking your progress?

These questions are intended to ensure everyone is on the same page about what has been done, what is going to be done, and any problems that are occurring.

These meetings are intended to be brief, so discussions and issues that come up are put off until after the meeting so that those not involved with those issues and discussions can leave.

The sprint concludes with a sprint review meeting where improvements to the product are demonstrated to key stakeholders for feedback. This feedback is then used by product owner to modify the product backlog if necessary.

Also at the end of a Sprint, the team holds a separate meeting known as a Sprint Retrospective, of which the goal is to improve the teams use of the Scrum process. There will be a separate reading that fully addresses Sprint Retrospectives.

After a Sprint Retrospective, the next sprint planning begins, completing a cycle that continues until the project is considered finished or time or budget has run out.

# Scrum Roles

There are three roles in Scrum, two of which have already been mentioned:

- Product Owner
- Team Member
- Scrum Master

Sprint meetings and retrospectives are typically attended by an additional person known as a Scrum Master. The Scrum Master should neither be the Product Owner or a member of the development team, and is there for two principle roles:

- Help the team in improving its use of the Scrum process.
- Help identify and remove impediments blocking progress.

The Scrum Master is not the teams manager and should not be making decisions for the team, and should instead be offering advice and helping to coordinate solutions.

In class the roles of the team member, the product owner, and scrum master have to be somewhat jumbled together. In some ways every team member has to be part Product Owner, part Scrum Master, and part member of the development team. Meanwhile the instructors will sparingly act as aspects of the Product Owner and Scrum Master.

All of this is bad practice in industry, each role should not be the same people, and there should be one product owner and one scrum master per development team. However this arrangement seems to be impossible in a classroom environment if the goal is for every student to have equal access to gaining experience with every part of the software development process

The product owner should be someone with the power to decide whether what the team is doing is in line with the vision for the project, and to make changes when that is not the case. They also should not be a member of

the development team, because of this confusing power dynamic. In addition some of the managerial aspects of product ownership is a lot of work.

This is why the ownership of the project spread out amongst the team during the early phases, it is not intended that a student with no management experience or even application development experience becomes the “boss”. Instead every team member should feel partially responsible for making sure the project is headed in the right direction and have a clear vision for the project’s end result. When the team makes some clearly bad decisions and moves in the wrong direction, the instructors will lightly nudge them in the right direction. However, only in extreme circumstances, in which team communication has broken down, do instructors get heavily involved with the team decisions.

The Scrum Master is likewise a difficult role to fill, mainly because students do not have enough experience with Scrum to help guide the Scrum process. A Scrum Master is a typically a full time position that serves one or more software development teams.

# Summary

In this lecture we have discussed the two main types of software processes:

- Plan Driven (Waterfall)
- Incremental (Agile)

We have discussed Agile, its key values, and its principles.

We have also gone into detail on the Scrum development process, what its main activities are, its roles, and how all of this is incorporated into the class.

In the next reading we will go into detail on User Stories and the creation of the Product Backlog, which will also be the next step in your software project.