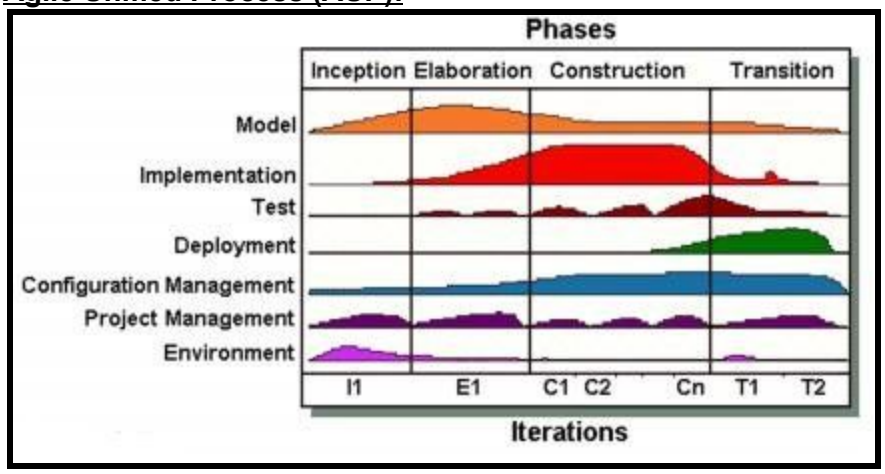
Agile Unified Process (AUP):



- I1, E1, C1, C2, Cn, T1, and T2 are just sprints.
- The agile phases are **inception**, **elaboration**, **construction**, **transition**.
- Table of the agile phases:

	Inception	**Elaboration**	**Construction**	**Transition**
Description	Is the phase where you design your app.	Is the phase where you take the design and you construct the CRC diagrams from it.	Is the phase where you start implementation and do testing.	Is the phase where you deploy your code.
Model	This is the stage where you start designing your app.	You take your design from inception and you make it more concrete by using CRC cards or other tools.	You might discover details that you did not catch in the previous 2 phases, but there should not be much designing in this stage.	There is very little, if any, designing at this stage.
Implementation	There is very little implementation in this phase. At most, you will only install the necessary software and tools needed.	Here, you start thinking about all the different classes you need.	This is where you do most of the implementation.	There is very little, if any, coding at this stage.
Test	There is very little, if any, testing at this stage.	There is very little, if any, testing at this stage.	This is where you do most of the testing.	You can do some testing at the start of this stage.
Deployment	There is very little, if any, deployment at this stage.	There is very little, if any, deployment at this stage.	Deployment may start near the end of this stage.	This is where you deploy your code.

Release Planning:

- During the release planning meeting the following things are established:
  - **Major release goals** (What you're aiming to deliver at the very end.)
  - **Release plan** (What you're going to do in each sprint. This doesn't have to be very detailed.)
  - **Potential sprint goals**
  - **Completion date**
- As each sprint progresses the **burndown of story points** measures the velocity of work, which can be used to determine progress and adapt the plan as we go.
- **Burndown charts** show the progress of completing all of the committed points within a single sprint on a day to day basis. That is, it starts off with total points in the sprint and tracks them on a day to day basis.
- The burndown shows how much of points or sprint is left at any point in time. Any issues added after the start of the sprint will be added in the burndown as scope change.
- E.g. You start off with 30 points, it will be tracked at the top left corner of the chart and a horizontal line would be sketched down to 0 across the chart.
  If you complete a story with 5 points after 1 day, the total burndown would be 25 and it would be reflected on the chart accordingly.

Sprint Planning:

- The stories with the highest priority from the product backlog items (PBI) are estimated.
- Stories are broken into tasks and tasks assigned to team members.
- Sprint planning meetings may include additional domain experts, who are not part of the team, to help answer any questions and aid in time estimations.
- The Scrum Master helps identify constraints that might impact the team's ability to commit to the sprint goal.
- Implementation details are discussed here.
- The product owner must be available to answer any questions related to the design.

Story Points:

- We need a common way to compare story sizes.
- It can be hard to find common ground between a programming story and a database management story.
- **Story points** are a relative measure of a feature's size or complexity.
  They are not durations nor a commitment to when a story will be completed.
  Different teams have different velocities, so they may complete stories at different rates depending on experience.
- A good tool to do the estimation is **planning poker**. It is a series similar to the Fibonacci Series that can be a useful range for story points. Here, each number is almost the sum of the two preceding numbers: 0, 1, 2, 3, 5, 8, 13, 20, 40, 100.
- 0-points estimates are used for trivial tasks that require little effort, though too many zero-pointers can add up.
- Only use numbers within the set and avoid averages. We avoid averages because if a user story turns out to be harder than expected, then the people who picked a higher number will say "I told you" to the people who picked a lower number. The average does not convince other people.
- What happens is this:
  1. A feature is mentioned.
  2. Each person in the team takes a number from the set, but doesn't show/tell anyone else yet.
     They choose the number based on how difficult they think implementing the feature will be.
     A feature with point 0 means that it requires very little effort.

3. After 3 seconds, everyone shows their number.
4. If everyone or most people have the same number, it's good.
5. If everyone or most people have different numbers, then each person has to defend why they picked their number.
   Once the discussion has been carried out, there is a second round of voting.
6. The process repeats until everyone agrees.
   If people consistently do not agree with one another, then the user story is not a good user story. This is because one of the features of a user story is that it must be estimable. (Remember INVEST).

- Powers of 2 is also an effective tool to do the estimation.

Ideal Days:

- Another unit of measure. It can be used as a transition for teams that are new to agile.
- It represents an ideal day of work with no interruptions (phone calls, questions, broken builds, etc.)
  However, it doesn't mean an actual day of work to finish.
- Tasks are estimated in hours.
  An estimation is an ideal time (without interruptions/problems).
  Smaller task estimates are more accurate than large.
- After all tasks have been estimated, the hours are totaled up and compared against the remaining hours in the sprint backlog. If there is room, the PBI is added and the team commits to completing the PBI.
  If the new PBI overflows the sprint backlog, the team does not commit and
  - the PBI can be returned to the product backlog and a smaller PBI chosen instead or
  - we can break the original PBI into smaller chunks or
  - we can drop an item already in the backlog to make room or
  - the product owner can help decide the best course of action.

Determining a Sprint Length:

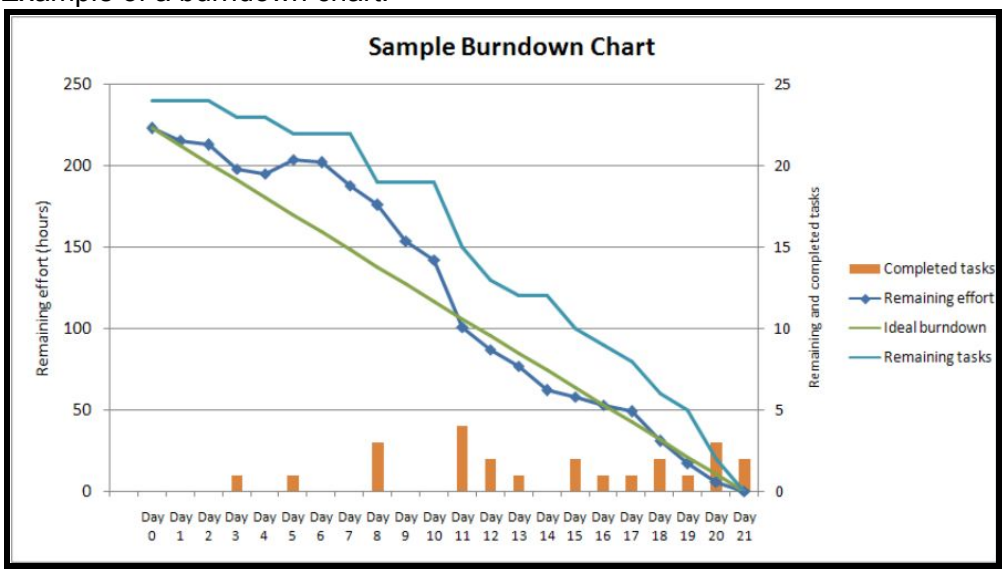
- It is typically 2-4 weeks.
- Some factors to consider:
  1. **Frequency of customer feedback:**
     - How long can the stakeholders go without seeing progress/giving input.
  2. **Team's level of experience:**
  3. **Time overhead for planning and reviews:**
     - Review and planning require a good chunk of the day regardless of sprint length.
  4. **Ability to plan the entire sprint:**
     - If there is uncertainty about how to achieve the sprint goal a shorter duration is advisable.
  5. **Intensity of the team over the sprint:**
     - Avoid mini-crunch periods.

Tracking Progress:

- Information about progress, impediments and sprint backlog of tasks needs to be readily available.
- How close a team is to achieving their goals is also important.
- Scrum employs a number of practices for tracking this information:
  1. Task cards
  2. Burndown charts
  3. Task boards
  4. War rooms (standups)

Burndown Charts:

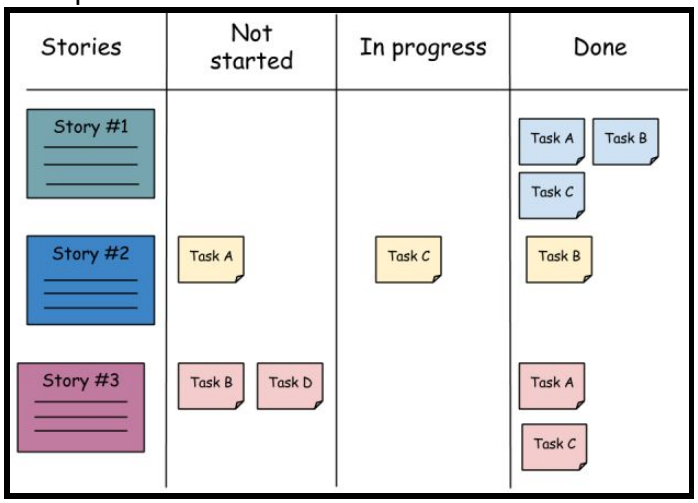
- Example of a burndown chart:



- Indicates how much work has been done in terms of how many user stories have been completed and when.
- Burndown charts are on Jira.
- On the beginning of the sprint, on the vertical axis, is the number of user stories you'd like to implement.
- At the end of the sprint, the number of user stories should be decreased to 0. That means all the stories have been implemented.
- A burndown chart is a graphical representation of work left to do versus time. It is often used in agile software development methodologies such as Scrum.
- Typically, in a burndown chart, the outstanding work is often on the vertical axis, with time along the horizontal. It is useful for predicting when all of the work will be completed.

Taskboard:

- Example of a taskboard:



- Both taskcards and taskboards are on Jira.
- The leftmost column are the stories to be implemented and there are 3 columns describing the progress of the tasks.

- In scrum the **task board** is a visual display of the progress of the scrum team during a sprint. It presents a snapshot of the current sprint backlog allowing everyone to see which tasks remain to be started, which are in progress and which are done.
- Simply put, the task board is a physical board on which the user stories which make up the current sprint backlog, along with their constituent tasks, are displayed. Usually this is done with index cards or post-it notes.
- The task board is usually divided into the columns listed below.
  Stories: This column contains a list of all the user stories in the current sprint backlog.
  Not started: This column contains sub tasks of the stories that work has not started on.
  In progress: All the tasks on which work has already begun.
  Done: All the tasks which have been completed.

Daily Scrum Meeting:

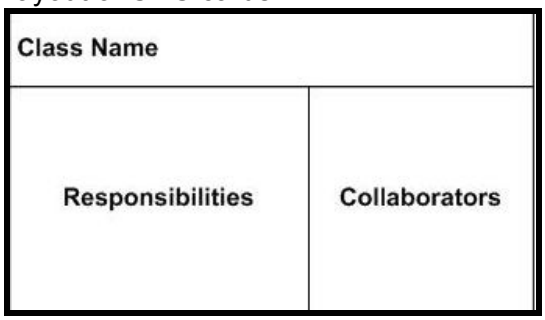
- It is a 15 minute meeting that everyone must attend.
- No sitting down, team stands in a circle and answers the following questions:
  1. What have I done since the last meeting?
  2. What am I going to accomplish between now and the next meeting?
  3. What are the problems or impediments that are slowing me down?
- It is not for solving problems. The Scrum Master must ensure that all side conversations are kept to a minimum. Solving problems happens throughout the rest of the day.
- It can be evolved to meet a specific team's requirements, but the purpose must remain the same (status, commitment, improvement).

Sprint Reviews:

- Occur on the last day of the sprint.
- The team and stakeholders come together to discuss the work accomplished.
- The product owner accepts or declines the results of the sprint.
- If a feature is declined, the owner will decide if it is returned to the backlog or simply dropped.
- Honesty is crucial.
- Cannot discourage criticism simply because a lot of work was put in.

CRC:

- CRC cards are a technique/tool, used for high-level sketching of our Object-Oriented architecture.
- We write information on index cards, using a few basic conventions.
- We include just enough details to allow us to "play out" scenarios (that are based on User Stories).
- Layout of CRC cards:



- **Classes** are the components of our system.
  They include:
  - Class/interface name
  - Abstract/Interface indicator
  - Super/parent class, and subclasses.
- **Responsibilities** are what each component does.
  I.e. The class variables.
  Traditionally, they are written as English sentences, but some people prefer to specify them as method signatures.
  Private methods of a component should not be included, as they are irrelevant to all other components.
- **Collaborators** are the dependencies of a component.
  I.e. The class methods
- Arguments for using CRC cards:
  - Quickly sketch out an architecture, play out scenarios, and validate your design.
  - Collaborative process that includes more team members in the design process.
  - Communicate our design to less technical people, or to developers who are not fluent in the programming language we are using.
- Arguments against using CRC cards:
  - Takes just as long as coding interfaces and classes with empty implementations.
  - Easier for developers to browse/search the info in their IDE, rather than using index cards.
  - The high-level nature of CRC cards means that some developers may interpret them differently, and an additional architecture discussion will be necessary.

Cohesion vs Coupling:

- **Cohesion** refers to the degree to which the elements inside a module belong together.
- High-cohesion means that each class takes care of one thing, and one thing only.
- Low cohesion implies that a given module performs tasks which are not very related to each other and hence can create problems as the module becomes large.
- Modules with high cohesion tend to be preferable, because high cohesion is associated with several desirable traits of software including robustness, reliability, reusability, and understandability. In contrast, low cohesion is associated with undesirable traits such as being difficult to maintain, test, reuse, or even understand.
- Think of building a physical robot. Many small parts (highly cohesive), versus a few "mega parts" (low cohesion, monolithic).
- **Coupling** refers to the interdependencies between modules.
  I.e. Components that are mutually dependent are also called coupled.
- A **loosely coupled** system is one in which each of its components has, or makes use of, little or no knowledge of the definitions of other separate components.
- **Tight coupling/tightly coupled** is a type of coupling that describes a system in which hardware and software are not only linked together, but are also dependent upon each other.
- Loose coupling is important because it enables isolation.
- We want to keep the coupling low and the cohesion high.
- Good software design/architecture should include the following:
  - Increases cohesion.
  - Makes the system loosely coupled.
  - Reduces dependencies.

- Goals of good design are:
    - Test different modules separately.
    - Work on one component without affecting other components.
    - Change implementations of different modules.
    - Move our application to a better (suited) infrastructure.
    - Easy to automate.
    - Easy to reuse different components.
- The same principles of good software design/architecture are applied in different places in the stack.
- For very small teams, who are trying to be agile, good design provides the highest "bang for the buck".
  Coordination is easy with a team of few responsible people.
- Good architecture guides developers. If the convenient way to implement something is also the right way, you will end up with better code.
- In general, software systems model/simulate real world problems/domains.
- A model is made up of classes that represent the different pieces of the real world problem.

DAO (Data Access Objects):

- The DAO is an object or interface that provides access to an underlying database or any other persistence storage.
  I.e. It is an object/interface, which is used to access data from a database or data storage.
- We use DAO because it abstracts the retrieval of data from a data resource such as a database. The concept is to separate a data resource's client interface from its data access mechanism.
- The problem with accessing data directly is that the source of the data can change.
- They are utility classes used by other classes.
- They are usually long-lasting.
  I.e. They are created when the application starts, and are there until the application exits.
- They are usually expressed as interfaces, to make good modular design.

Epics:

- An **epic** is a large body of work that can be broken down into a number of smaller stories.
- Epics are almost always delivered over a set of sprints.
- An epic can be defined as a big chunk of work that has one common objective. It could be a feature, customer request or business requirement. In backlog, it is a placeholder for a required feature with few lines of description. It tells compactly about final output of user needs. In the beginning, it may not contain all the details that the team needs to work on. These details are defined in user stories. An epic usually takes more than one sprint to complete.