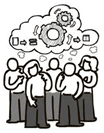
## Part 2 –Managing and prioritization of requirements

## Agile Development cycle

## What is Agile Methodology?

Agile Methodology meaning a practice that promotes **continuous iteration** of development and testing throughout the software development lifecycle of the project. In the Agile model in software testing, both development and testing activities are concurrent, unlike the Waterfall model.



Agile Methodology

## What is Agile Software Development?

The **Agile software development** methodology is one of the simplest and effective processes to turn a vision for a business need into software solutions. Agile is a term used to describe software development approaches that employ continual planning, learning, improvement, team collaboration, evolutionary development, and early delivery. It encourages flexible responses to change.

The agile software development emphasizes on four core values.

1. Individual and team 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

## Agile Model Vs Waterfall Model

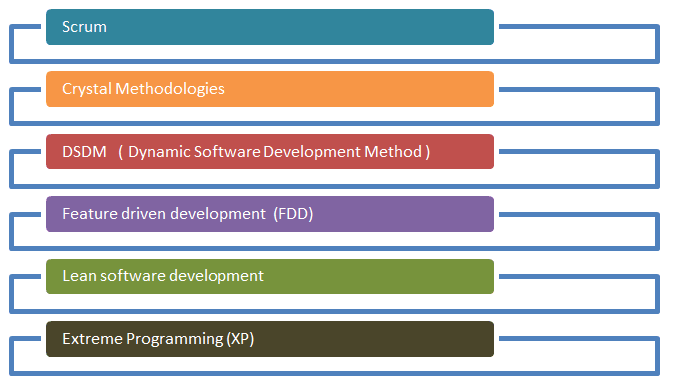
Agile and Waterfall model are two different methods for software development process. Though they are different in their approach, both methods are useful at times, depending on the requirement and the type of the project.

|  |  |
| --- | --- |
| **Agile Model** | **Waterfall Model** |
| * Agile methodology definition: Agile methodologies propose incremental and iterative approach to software design | * Waterfall Model: Development of the software flows sequentially from start point to end point. |
| * The **Agile process** in software engineering is broken into individual models that designers work on | * The design process is not broken into an individual models |
| * The customer has early and frequent opportunities to look at the product and make decision and changes to the project | * The customer can only see the product at the end of the project |
| * Agile model is considered unstructured compared to the waterfall model | * Waterfall model are more secure because they are so plan oriented |
| * Small projects can be implemented very quickly. For large projects, it is difficult to estimate the development time. | * All sorts of project can be estimated and completed. |
| * Error can be fixed in the middle of the project. | * Only at the end, the whole product is tested. If the requirement error is found or any changes have to be made, the project has to start from the beginning |
| * Development process is iterative, and the project is executed in short (2-4) weeks iterations. Planning is very less. | * The development process is phased, and the phase is much bigger than iteration. Every phase ends with the detailed description of the next phase. |
| * Documentation attends less priority than software development | * Documentation is a top priority and can even use for training staff and upgrade the software with another team |
| * Every iteration has its own testing phase. It allows implementing regression testing every time new functions or logic are released. | * Only after the development phase, the testing phase is executed because separate parts are not fully functional. |
| * In agile testing when an iteration end, shippable features of the product is delivered to the customer. New features are usable right after shipment. It is useful when you have good contact with customers. | * All features developed are delivered at once after the long implementation phase. |
| * Testers and developers work together | * Testers work separately from developers |
| * At the end of every sprint, user acceptance is performed | * User acceptance is **performed** at the end of the project. |
| * It requires close communication with developers and together analyze requirements and planning | * Developer does not involve in requirement and planning process. Usually, time delays between tests and coding |

## Ref: <https://www.guru99.com/agile-scrum-extreme-testing.html>

### Agile Process

Check the below [Agile methodology](https://www.guru99.com/agile-testing-course.html) process to deliver successful systems quickly.

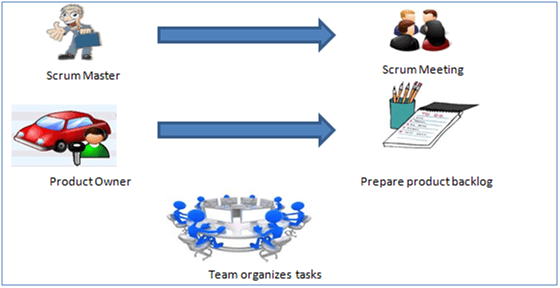


Agile Process Model

There are various **Agile methods** present in agile testing, and those are listed below:

## Scrum

SCRUM is an agile development method which concentrates specifically on how to manage tasks within a team-based development environment. Basically, Scrum is derived from activity that occurs during a rugby match. Scrum believes in empowering the development team and advocates working in small teams (say- 7 to 9 members). Agile and Scrum consist of three roles, and their responsibilities are explained as follows:



Scrum Method

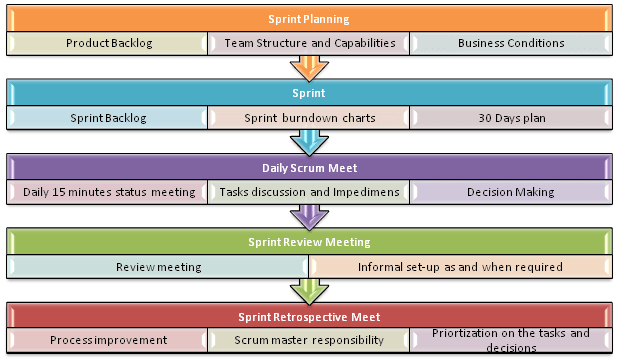
* Scrum Master
  + [Scrum Master](https://www.guru99.com/scrum-master-training.html) is responsible for setting up the team, sprint meeting and removes obstacles to progress
* Product owner
  + The Product Owner creates product backlog, prioritizes the backlog and is responsible for the delivery of the functionality at each iteration
* Scrum Team
  + Team manages its own work and organizes the work to complete the sprint or cycle

## Product Backlog

This is a repository where requirements are tracked with details on the no of requirements(user stories) to be completed for each release. It should be maintained and prioritized by Product Owner, and it should be distributed to the scrum team. Team can also request for a new requirement addition or modification or deletion

## Scrum Practices

Practices are described in detailed:



Scrum Practices

## Process flow of Scrum Methodologies:

Process flow of [scrum testing](https://www.guru99.com/scrum-testing-beginner-guide.html) is as follows:

* Each iteration of a scrum is known as Sprint
* Product backlog is a list where all details are entered to get the end-product
* During each Sprint, top user stories of Product backlog are selected and turned into Sprint backlog
* Team works on the defined sprint backlog
* Team checks for the daily work
* At the end of the sprint, team delivers product functionality

https://www.knowledgehut.com/blog/agile/product-backlog-refinement-scrum

Product Backlog Refinement

also referred to as Product Backlog Grooming, is a method for keeping the backlog updated, clean and orderly. It is a basic process in Scrum. PBR is a collaborative discussion process which starts at the end of one sprint to confirm whether the backlog is ready for the next sprint. Backlog can be defined as a set of user stories which are not present in the current sprint that defines project’s scope context. The stories which are left unattended, may interfere with the functioning of the development team. When this happens, the status of user stories will not be clear, and even the team can lose focus and fail to deliver within the project completion date.

The backlog grooming meeting is attended by the [scrum master](https://www.knowledgehut.com/agile-management/csm-certification-training), who facilitates everything for team members, the team and the product owner. They decide among the top items from the product backlog. The team comprises mainly of Developers, testers and Scrum Masters. The team can raise queries during the sprint planning session if they find any unresolved issue. The expected doubts can arise in the following forms :

* How to handle the situation if the user enters invalid data?
* Which part of the system are the users authorized to operate on?

For the product owner, it will be easy to get the conclusion over the queries, by asking these questions in the early stages. If there is a question which is unanswered by too many people, it is time to make some changes in your backlog items by curating higher priority items to the top of the list and assigning highest priority to the unanswered questions.

**The Objective of PBR meeting:**

A lot of time is saved at sprint planning meetings, if the backlogs are well maintained. If the backlog item is clearly specified in the acceptance criteria and cross-checked properly by the team members, the planning process can be accomplished prior to the meeting. PBR offers the team members the opportunity to interact with each other regarding stories.

**Why is PBR important?**

PBR and its sessions are important mainly due to the following features-

* It increases the efficiency of the team by reducing uncertainty
* Properly refined stories are easy to estimate, test and implement.
* PBR session increases the efficiency of the team due to the knowledge shared among the team members.
* If PBR meeting is maintained properly, it helps reduce the time for a Sprint planning meeting.

The Product Backlog grooming can be made effective if the following aspects are considered-

* Do not schedule backlog refinement meeting during the first and last 20% of the Sprint Planning session.
* Backlog refinement meeting should be considered as the first part of Sprint Planning.
* The backlog items’ list should be well understood by the PO, or development team member to work well in the meeting. Make sure that the set of predefined acceptance tests are present.
* Keep an eye on the meeting goals.
* Make sure to assign action items for any unknown thing.
* Do remember that the backlog items are a collaboration between the PO and the team.
* Feel free to break the product backlog items during the meeting.

After the product backlog refinement meeting, the team can update the Product Backlog items in line, based on the discussions held. Finally, you can get a potentially shippable product, ready to be deployed in the market.

## What are story points

Story points are a unit of measure for expressing an estimate of the overall effort that will be required to fully implement a product backlog item or any other piece of work.

When we estimate with story points, we assign a point value to each item. The raw values we assign are unimportant. What matters are the relative values. A story that is assigned a 2 should be twice as much as a story that is assigned a 1. It should also be two-thirds of a story that is estimated as 3 story points.

Instead of assigning 1, 2 and 3, that team could instead have assigned 100, 200 and 300. Or 1 million, 2 million and 3 million. It is the ratios that matter, not the actual numbers.

## What Goes Into a Story Point?

Because story points represent the effort to develop a story, a team’s estimate must include everything that can affect the effort. That could include:

* The amount of work to do
* The complexity of the work
* Any risk or uncertainty in doing the work

When estimating with story points, be sure to consider each of these factors. Let’s see how each impacts the effort estimate given by story points.

## The Amount of Work to Do

Certainly, if there is more to do of something, the estimate of effort should be larger. Consider the case of developing two web pages. The first page has only one field and a label asking to enter a name. The second page has 100 fields to also simply be filled with a bit of text.

The second page is no more complex. There are no interactions among the fields and each is nothing more than a bit of text. There’s no additional risk on the second page. The only difference between these two pages is that there is more to do on the second page.

The second page should be given more story points. It probably doesn’t get 100 times more points even though there are 100 times as many fields. There are, after all, economies of scale and maybe making the second page is only 2 or 3 or 10 times as much effort as the first page.

## Risk and Uncertainty

The amount of risk and uncertainty in a product backlog item should affect the story point estimate given to the item.

If a team is asked to estimate a product backlog item and the stakeholder asking for it is unclear about what will be needed, that uncertainty should be reflected in the estimate.

If implementing a feature involves changing a particular piece of old, brittle code that has no automated tests in place, that risk should be reflected in the estimate.

## Complexity

Complexity should also be considered when providing a story point estimate. Think back to the earlier example of developing a web page with 100 trivial text fields with no interactions between them.

Now think about another web page also with 100 fields. But some are date fields with calendar widgets that pop up. Some are formatted text fields like phone numbers or Social Security numbers. Other fields do checksum validations as with credit card numbers.

This screen also requires interactions between fields. If the user enters a Visa card, a three-digit CVV field is shown. But if the user enters an American Express card, a four-digit CVV field is shown.

Even though there are still 100 fields on this screen, these fields are harder to implement. They’re more complex. They’ll take more time. There’s more chance the developer makes a mistake and has to back up and correct it.

This additional complexity should be reflected in the estimate provided.

## Consider All Factors: Amount of Work, Risk and Uncertainty, and Complexity

It may seem impossible to combine three factors into one number and provide that as an estimate. It’s possible, though, because effort is the unifying factor. Estimators consider how much effort will be required to do the amount of work described by a product backlog item.

Estimators then consider how much effort to include for dealing with the risk and uncertainty inherent in the product backlog item. Usually this is done by considering the risk of a problem occurring and the impact if the risk does occur. So, for example, more will be included in the estimate for a time-consuming risk that is likely to occur than for a minor and unlikely risk.

Estimators also consider the complexity of the work to be done. Work that is complex will require more thinking, may require more trial-and-error experimentation, perhaps more back-and-forth with a customer, may take longer to validate and may need more time to correct mistakes.

All three factors must be combined.

## Consider Everything in the Definition of Done

A story point estimate must include everything involved in getting a product backlog item all the way to done. If a team’s definition of done includes creating automated tests to validate the story (and that would be a good idea), the effort to create those tests should be included in the story point estimate.

Story points can be a hard concept to grasp. But the effort to fully understand that points represent effort as impacted by the amount of work, the complexity of the work and any risk or uncertainty in the work will be worth it.

## Story points and estimation

**Good estimation helps product owners optimize for efficiency and impact. That's why it's so important.**

Estimation is hard. For software developers, it's among the most difficult — if not the most difficult — aspects of the job. It must take into account a slew of factors that help product owners make decisions that affect the entire team — and the business. With all that at stake, it's no wonder everyone from developers to upper management is prone to getting their undies in a bunch about it. But that's a mistake. Agile estimation is just that: an estimate. Not a blood-oath.

There's no requirement to work weekends in order to compensate for under-estimating a piece of work. That said, let's look at some ways to make agile estimates as accurate as possible.

## Collaborating with the product owner

In agile development, the [product owner](https://www.atlassian.com/agile/product-management) is tasked with prioritizing the [backlog](https://www.atlassian.com/agile/scrum/backlogs) — the ordered list of work that contains short descriptions of all desired features and fixes for a product. Product owners capture [requirements](https://www.atlassian.com/agile/product-management/requirements) from the business, but they don’t always understand the details of implementation. So good estimation can give the product owner new insight into the level of effort for each work item, which then feeds back into their assessment of each item's relative priority.

When the engineering team begins its estimation process, questions usually arise about requirements and user stories. And that's good: those questions help the entire team understand the work more fully. For product owners specifically, breaking down work items into granular pieces and estimates via story points helps them prioritize all (and potentially hidden!) areas of work. And once they have estimates from the dev team, it's not uncommon for a product owner to reorder items on the backlog.

## Agile estimation is a team sport

Involving everyone (developers, designers, testers, deployers... everyone) on the team is key. Each team member brings a different perspective on the product and the work required to deliver a user story. For example, if product management wants to do something that seems simple, like support a new web browser, development and QA need to weigh in because their experience has taught them what dragons may be lurking beneath the surface.

Likewise, design changes require not only the design team's input, but that of development and QA as well. Leaving part of the broader product team out of the estimation process creates lower quality estimates, lowers morale because key contributors don't feel included, and compromises the quality of the software.

So don’t let your team fall victim to estimates made in a vacuum. It’s a fast track to failure!

## Story points vs. hours

Traditional software teams give estimates in a time format: days, weeks, months. Many agile teams, however, have transitioned to story points. Story points are units of measure for expressing an estimate of the overall effort required to fully implement a product backlog item or any other piece of work. Teams assign story points relative to work complexity, the amount of work, and risk or uncertainty. Values are assigned to more effectively break down work into smaller pieces, so they can address uncertainty. Over time, this helps teams understand how much they can achieve in a period of time and builds consensus and commitment to the solution.  It may sound counter-intuitive, but this abstraction is actually helpful because it pushes the team to make tougher decisions around the difficulty of work. Here are few reasons to use story points:

* Dates don’t account for the non-project related work that inevitably creeps into our days: emails, meetings, and interviews that a team member may be involved in.
* Dates have an emotional attachment to them. Relative estimation removes the emotional attachment.
* Each team will estimate work on a slightly different scale, which means their velocity (measured in points) will naturally be different. This, in turn, makes it impossible to play politics using velocity as a weapon.
* Once you agree on the relative effort of each story point value, you can assign points quickly without much debate.
* Story points reward team members for solving problems based on difficulty, not time spent. This keeps team members focused on shipping value, not spending time.

## Story points methodology

## Why the Fibonacci series is used in Agile

The Fibonacci sequence is found in many different disciplines and in nature. For example, it has been used to describe plant life growth, estimate population increases over a specified timeframe, model virus breakouts, and predict the behavior of financial markets.

But what does this long string of numbers have to do with Agile planning?

Essentially, the Agile Fibonacci scale gives teams a more realistic way to approach estimates using story points. Story points are used to represent the size, complexity, and effort needed for completing or implementing a user story. Each story point is assigned a number from the Fibonacci scale. The higher the number, the more complex the story point, and presumably, the amount of effort it will take to complete.

As discussed previously, estimates can often be inaccurate—and that happens because people tend to be overly optimistic.

For example, instead of making an estimate based on a similar project we completed in the past, we believe we can get it done faster because we are more experienced and we are sure that this time there won’t be any problems that cause delays.

Because the Agile Fibonacci Scale is exponential rather than linear, it helps teams to be more realistic when looking at larger, more complex tasks.

To help you understand why the exponential nature of the Fibonacci series is helpful, we’ll paraphrase an analogy used by Mike Cohn, one of the founders of the Scrum Alliance:

Imagine holding a one-pound weight in one hand and a two-pound weight in the other. Without looking, you are likely able to determine which is which because the 2-pound weight is noticeably heavier—the two-pound weight is 100% heavier than the 1-pound weight).

If you were to hold a 20-pound weight and a 21-pound weight, it is harder to determine which is heavier because the difference between the weights is not that significant—the 21-pound weight is only 5% heavier.

Each number in the Fibonacci scale is exponentially larger (by about 60%) than the previous number. Teams can more easily recognize the differences and define the complexity of each story point.

## How to use the Fibonacci scale in Agile

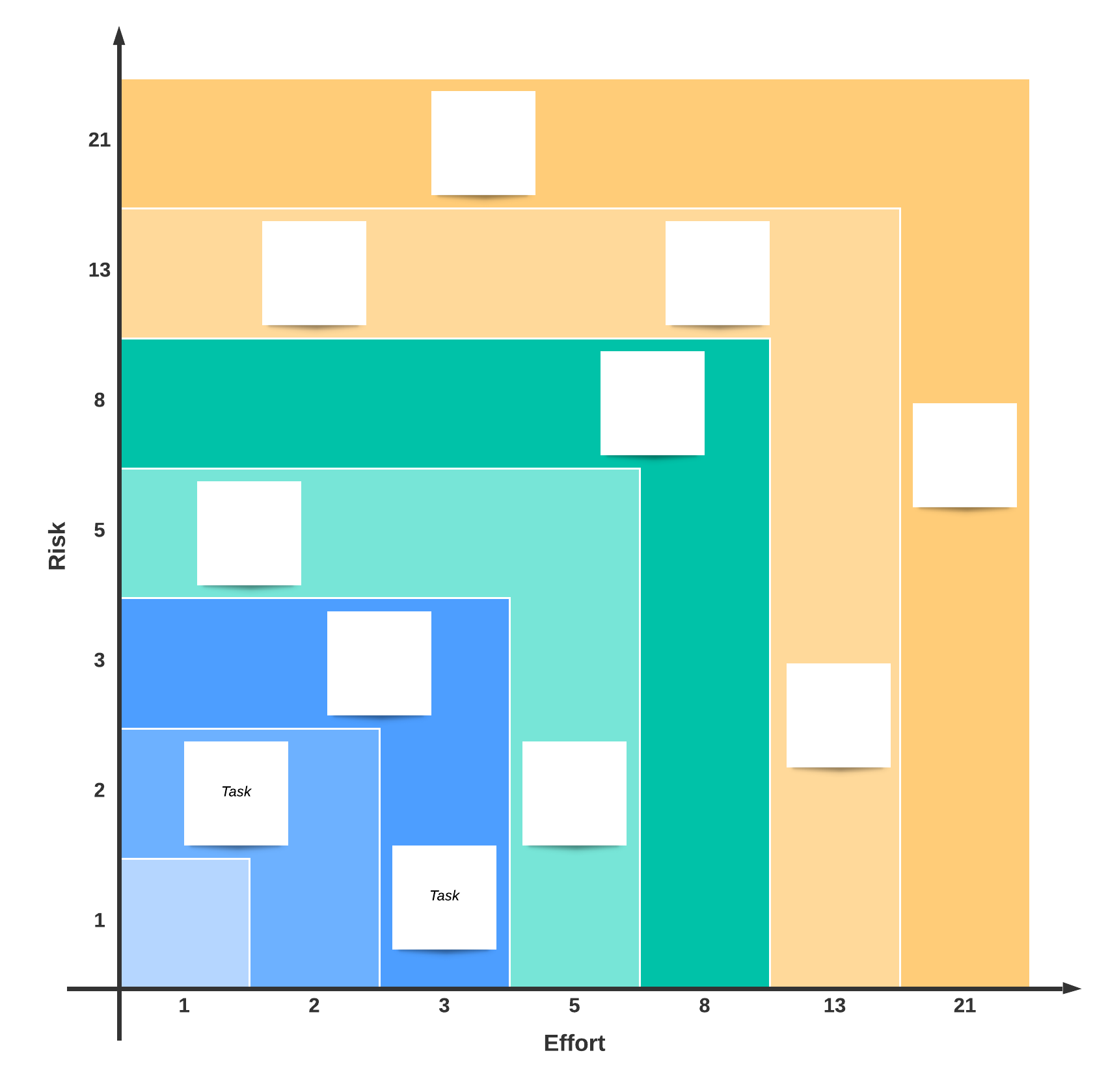
Many Agile teams use the planning poker technique for estimating story points.

This technique involves card decks with Fibonacci numbers starting with zero (0, 1, 2, 3, 5, 8, 13, 21, etc.) or a modified version of the sequence (0, .05, 1, 2, 3, 5, 8, 13, 20, 40, 100). Your team can use any number sequence with a fixed ratio (e.g., 1, 2, 4, 8, etc.) as long as the team understands and agrees with what the ratios mean.

Each member of the estimation team, aside from the moderator, needs their own deck of cards, and the planning poker technique proceeds with these steps:

1. The product owner gives the team an overview of a user story.
2. The team is given time to discuss and ask questions to get a better understanding of risks and assumptions. Team members should not mention any estimation numbers so that estimates are not biased. The moderator should record a summary of the discussion and can use a timer to limit the discussion time.
3. Each team member chooses a card to represent their estimate of the story and places it facedown on the table.
4. Team members turn over their cards simultaneously.
   * If everybody selects the same number, you’re done. That number is used for the estimate, and you can move on to the next story.
   * Individuals who have selected numbers that are significantly higher or lower than other numbers are given time to justify their estimates. For example, the task may be simple for a developer to complete (such as adding a field to a form), but that simple addition could turn out to be more complex for testers later.
5. The process repeats from step 3 until the team reaches a consensus.
6. The next user story is introduced for estimation and the entire process repeats.

If your team doesn’t have physical card decks, you can run through your user story estimation process using the template below. An online Fibonacci scale is a great solution for distributed teams who can’t physically meet in the same location.

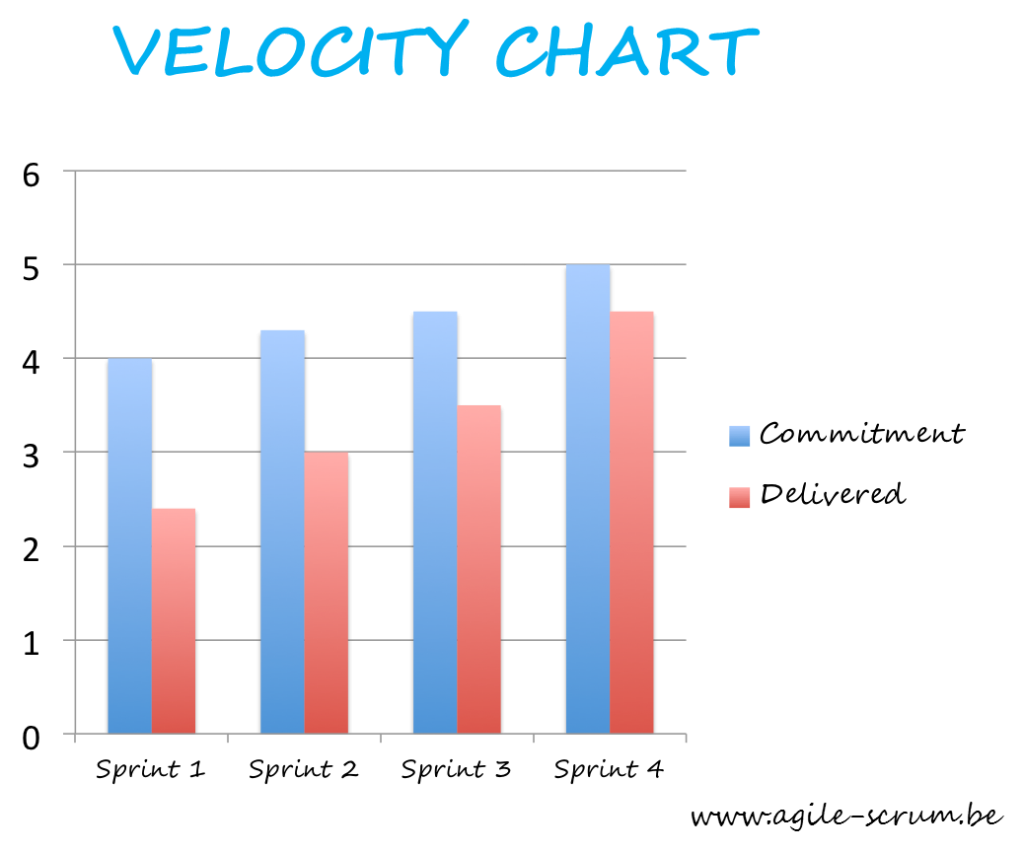
[](https://www.lucidchart.com/documents/editNewOrRegister/f8c09563-8974-464e-bf08-4d9dac3c5624?anonId=0.49e1e17e1830cbb050e&sessionDate=2022-09-05T08:28:09.263Z&sessionId=0.4e0c9a2d1830cbb0517)

# Velocity Chart

The concept of velocity can be easily explained. The main purpose of the velocity chart is to overview how much work has been delivered for each sprint. It will help you to have a clear view on future perspectives and on the workload of your team.

To estimate this metric, to calculate it, you need to define the units of work for each task and how long is each interval (the time). During the sprints every member of the team should communicate on the tasks that have been finished or completed. And at the end of the interval that we want to analyze the completed number of units of work are counted.

If we apply this formula, we will have one of the most important metrics for Scrum methodology, velocity of work flow.



## What Is a Burndown Chart?

A burndown chart is a [project management chart](https://www.projectmanager.com/blog/3-best-project-management-charts) that shows how quickly a team is working through a customer’s user stories. This agile tool captures the description of a feature from an end-user perspective and shows the total effort against the amount of work for each iteration or agile sprint.

The quantity of work remaining appears on a vertical axis while the time that’s passed since the beginning of the project is placed horizontally on the chart, showing the past and the future. The burndown chart is displayed so everyone on the [agile project management](https://www.projectmanager.com/training/agile-project-management-in-3-minutes) team can see it and is updated regularly for accuracy

### ypes of Burndown Chart

There are two burndown chart variants: a sprint burndown and a product burndown. A sprint burndown is used for work remaining in the iteration while a product burndown illustrates the work remaining for the entire project.

## Components of a Burndown Chart

Although the specifics can vary, it’s common to see the below sections of a burndown chart.

### Axes

A burndown chart has two axes, x and y. The horizontal axis represents time while the vertical axis displays user story points. The rightmost point of the chart indicates the start of a project or [agile sprint](https://www.projectmanager.com/guides/agile-sprints) while the leftmost point shows its end.

### Ideal Work Remaining Line

As its name suggests, the ideal work remaining line indicates the remaining work that a team has at a specific point of the project or sprint under ideal conditions. Managers use past data to estimate this baseline and draft a straight line across the burndown chart. The ideal work remaining line should always have a negative slope.

### Actual Work Remaining Line

The actual work remaining line indicates the remaining work a team has at any point of the project or sprint. Unlike the ideal work remaining line, this is not an estimate, but rather a realistic depiction of the team’s performance. The line is drawn as the team progresses and completes user stories. Actual work remaining lines are usually not straight as teams work at different paces as projects are completed.

## What is velocity in Agile?

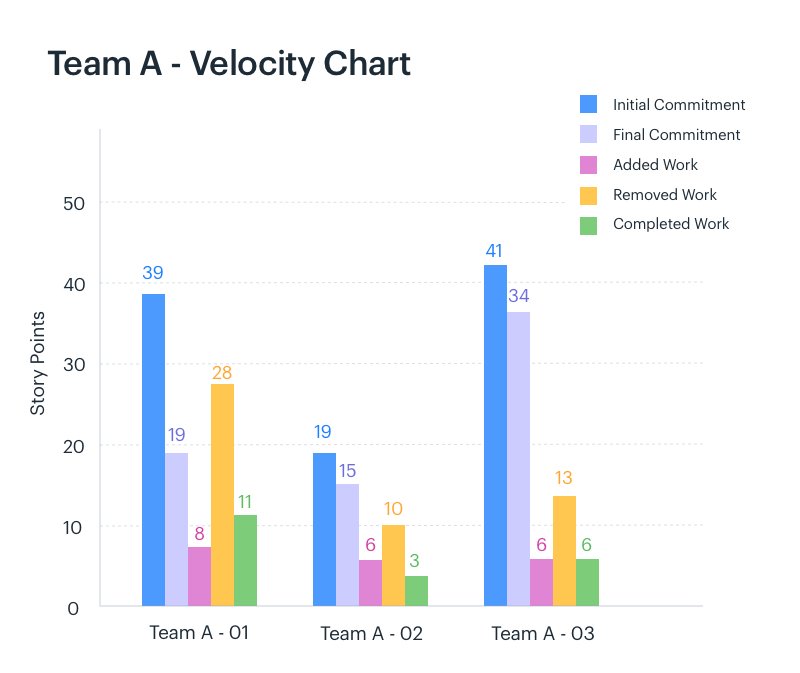
Velocity in agile is a measure of how much work an [agile team](https://kissflow.com/project/agile/agile-team/) can deliver on average in a sprint.

**It takes a view at measuring:**

* How much work an agile team has delivered in the past sprints
* How long it took the team to get the work done

With this in mind, the agile team and its [stakeholders](https://kissflow.com/project/project-stakeholder-management/) can accurately estimate the agile team’s capacity, i.e. how much work they can achieve in a specific duration, i.e. in a sprint.

In essence, velocity in agile offers agile teams a way of calculating how fast they’re getting work done, and as a result, how many sprints will be required to get [a project](https://kissflow.com/project/what-is-a-project/) to a certain degree, and overall, create significant change.



### Key factors to consider when measuring agile velocity for your projects

Like we’ve established, the need for agile velocity arises from wanting to know how fast an agile team can iterate and deliver features that incrementally develop the project. There are several factors that come into play when determining how fast an agile team can deliver.

#### 1. Sprint workload

How much work are we delivering in this iteration? Specifically, how many [backlog items](https://kissflow.com/project/agile/role-of-sprint-backlogs/), user stories are we moving from to-do to done [within this sprint](https://kissflow.com/project/agile/sprint-retrospectives/)?

##### **Tired of using Asana?**

###### 

#### 2. Sprint length

How long is this sprint going to last for? How long do we have to deliver our outlined workload? Is it feasible based on previous sprints?

#### 3. Burndown chart

The [burndown chart](https://kissflow.com/project/agile/benefits-of-burndown-charts/) shows the [action items](https://kissflow.com/project/agile/how-to-define-action-items/) the team has to execute, plotted against the amount of time remaining in a sprint.

## How to calculate velocity

Velocity in agile is a fairly simple concept to calculate. All you have to do is measure the total amount of backlog items that were delivered per sprint.

As the team runs through more iterations, you can determine an average amount of backlog items, or ideally, a slim range of backlog items or features the agile team can [deliver per sprint](https://kissflow.com/project/agile/scrum-sprint-planning/).

Velocity makes it easy for agile teams to estimate how much work they can achieve per sprint and how long it’ll take to get a project to a certain level of growth.

Agile velocity is measured in either days, ideal days, or the number of hours it takes the team to deliver a number of backlog items, story points, etc.