# What is SDLC Waterfall Model?

**What is *SDLC Waterfall Model*?**

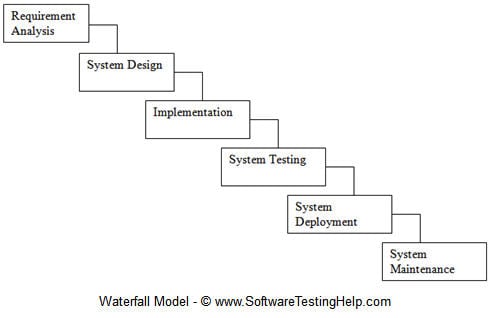
**Introduction**:

Waterfall model is an example of a Sequential model. In this model, the software development activity is divided into different phases and each phase consists of series of tasks and has different objectives.

Waterfall model is the pioneer of the [SDLC processes](http://en.wikipedia.org/wiki/Software_development_process). In fact, it was the first model which was widely used in the software industry. It is divided into phases and output of one phase becomes the input of the next phase. It is mandatory for a phase to be completed before the next phase starts. In short, there is no overlapping in Waterfall model

In waterfall, development of one phase starts only when the previous phase is complete. Because of this nature, each phase of waterfall model is quite precise well defined. Since the phases fall from higher level to lower level, like a waterfall, It’s named as waterfall model.

**Pictorial representation of waterfall model:**

****

**The activities involved in different phases are as follows:**

| **S.No** | **Phase** | **Activities Performed** |  |
| --- | --- | --- | --- |
| 1 | Requirement Analysis | 1. Capture all the requirements. 2. Do brainstorming and walkthrough to understand the requirements. 3. Do the requirements feasibility test to ensure that the requirements are testable or not. |  |
| 2 | System Design | 1. As per the requirements, create the design 2. Capture the hardware / software requirements. 3. Document the designs |  |
| 3 | Implementation | 1. As per the design create the programes / code 2. Integrate the codes for the next phase. 3. Unit testing of the code |  |
| 4 | System Testing | 1. Integrate the unit tested code and test it to make sure if it works as expected. 2. Perform all the testing activities (Functional and non functional) to make sure that the system meets the requirements. 3. In case of any anomaly, report it.  4. Track your progress on testing through tools like traceability metrics, ALM 5. Report your testing activities. |  |
| 5 | System Deployment | 1. Make sure that the environment is up 2. Make sure that there are no sev 1 defects open. 3. Make sure that the test exit criteria are met.  4. Deploy the application in the respective environment. 5. Perform a sanity check in the environment after the application is deployed to ensure the application does not break. |  |
| 6 | System maintenance | 1. Make sure that the application is up and running in the respective environment. 2. Incase user encounters and defect, make sure to note and fix the issues faced. 3. Incase any issue is fixed; the updated code is deployed in the environment.  4.The application is always enhanced to incorporate more features, update the environment with the latest features |  |

When to use SDLC Waterfall Model?

SDLC Waterfall model is used when

* Requirements are stable and not changed frequently.
* An application is small.
* There is no requirement which is not understood or not very clear.
* The environment is stable
* The tools and technology used is stable and is not dynamic
* Resources are well trained and are available.

**Pros and Cons of waterfall model:**

Advantages of using Waterfall model are as follows:

* Simple and easy to understand and use.
* For smaller projects, waterfall model works well and yield the appropriate results.
* Since the phases are rigid and precise, one phase is done one at a time, it is easy to maintain.
* The entry and exit criteria are well defined, so it easy and systematic to proceed with quality.
* Results are well documented.

**Disadvantages of using Waterfall model:**

* Cannot adopt the changes in requirements
* It becomes very difficult to move back to the phase. For example, if the application has now moved to the testing stage and there is a change in requirement, It becomes difficult to go back and change it.
* Delivery of the final product is late as there is no prototype which is demonstrated intermediately.
* For bigger and complex projects, this model is not good as a risk factor is higher.
* Not suitable for the projects where requirements are changed frequently.
* Does not work for long and ongoing projects.
* Since the testing is done at a later stage, it does not allow identifying the [challenges and risks](https://www.softwaretestinghelp.com/types-of-risks-in-software-projects/)in the earlier phase so the risk mitigation strategy is difficult to prepare.

**Conclusion**:

In the waterfall model, it is very important to take the [sign off](https://www.softwaretestinghelp.com/bug-tracking-test-metrics-and-test-sign-off-free-qa-training-day-6/) of the deliverables of each phase. As of today most of the projects are moving with [Agile](https://www.softwaretestinghelp.com/category/agile-testing/) and Prototype models, Waterfall model still holds good for smaller projects. If requirements are straightforward and testable, Waterfall model will yield the best results.

**Introduction to Agile: What is Agile Development?**

**Agile in Software Development:**

Agile is one of the most widely used and recognized software development frameworks.

The manifesto has been very carefully worded to capture the essence of agile in minimum words and it reads as below –

**“We are uncovering better ways of developing a software by doing it and helping others to do it. Through this work we have come to the below 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.”**

#### #1) Individuals and Interactions over Processes and Tools

Individuals and interactions are preferred over processes and tools because it makes the process more responsive. If the individuals are aligned and once they understand each other, then the team can resolve any issues with the tools or processes.

But if the teams insist on blindly sticking to the processes then it might cause misunderstandings among the individuals and create unexpected roadblocks thereby resulting in project delays.

That’s why it’s always preferable to have interactions and communication amongst the team members rather than blindly depending on processes to guide the way forward. One of the ways to achieve this is by having an involved product owner who works and can make decisions in collaboration with the development team.

Allowing individuals to contribute on their own also allows them to showcase freely as what they can bring to the table. When these team interactions are directed towards solving a common problem, the results can be quite powerful.

#### #2) Working Software over Comprehensive Documentation

Traditional project management involved comprehensive documentation which entailed a lag of months. This used to impact the project delivery negatively and the resulting delays were inevitable.

The kind of documentation created for these projects was very detailed and so many documents were created that many of them were not even referred to during the project progress. This was an unnecessary evil with which the project teams used to live with.

But this also exacerbated the problems in delivery. The focus was on documentation to such an extent because the teams wanted to end up with a finished product which was 100% as per the specifications. That’s why the focus was on capturing all the specifications in details.

But still, the end product used to be quite different from the expectations or would have lost relevance. That’s is why agile says that a working software is a much better option to gauge customer expectation than heaps of documentation.

This doesn’t imply that the documentation is not necessary. It just means that a working product is any day a better indicator of alignment to the customer needs and expectations than a document created months ago. It also implies that the teams are responsive and ready to adapt to change as and when required while showing the working software to the client when the sprint ends.

Failure to test the product during sprints takes manifold cost and effort in the next sprint. Once the functionality is deployed, the cost of these changes goes up further by a significant degree.

#### 3. Customer Collaboration over Contract Negotiation

Negotiation means that the details are still being captured and have not been finalized. There is still scope for renegotiation. But once the negotiation is over, there can be no discussion over it. What agile says is that instead of negotiation, go for collaboration.

Collaboration implies that there is still room for discussion and the communication is ongoing.

Not a one-time thing. What this does is, it gives a two-fold advantage – while it helps the team to do a course correction if required at an earlier stage, it helps the client to also refine their vision and redefine their requirements if required during the course of the project.

The other aspect is that while traditional software development models involve the customer before the development begins during the documentation and negotiation phase, and they are not as involved during the project development.

Once the requirements have been frozen, they get to see the product only, once the product is ready. Agile breaks through this barrier as well by allowing for customer involvement over the whole lifecycle.

This helps the agile teams align better to the customer needs. One of the ways to achieve this is through a dedicated and involved product owner who can help the team in real time for clarifications and aligning the work with the customer priorities

#### 4. Responding to Change Over Following a Plan

The standard thought process is that the changes are an expensive affair and we should avoid changes at all costs. That’s what the unnecessary focus is on documentation and elaborates plans to deliver by sticking onto the timelines and product specifications.

But as experience also teaches us, changes are mostly inevitable and instead of running from it we should try to embrace it and plan for it.

Agile allows us to do this transition. What agile thinks is that change is not an expense, it is a welcome feedback which helps to improve the project. It is not to be avoided but instead, it adds value.

With the short sprints proposed by agile, the teams can get a quick feedback and shift priorities at a short notice. New features can be added from iteration to iteration.

Why do we do this? Because most of the features developed using the waterfall approach are never used. This is because the waterfall model follows the plan whereas that is the phase when we know the least.

Agile also plans, but it also follows the just in time approach where planning is done just enough when needed. And the plans are always open to change as the sprints progress.

### The 12 Agile Principles

There are 12 agile principles which were added after the creation of the manifesto in order to help and guide teams transition into agile and check whether the practices they are following are in line with the agile culture.

**Following is the text of the original 12 principles, published in 2001 by the Agile Alliance:**

**#1)** our highest priority is to satisfy the customer through the early and continuous delivery of valuable software.

**#2)** Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.

**#3)** Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

**#4)** Business people and developers must work together daily throughout the project.

**#5)** Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

**#6)** the most efficient and effective method of conveying information to and within the development team is a face-to-face conversation.

**#7)** Working software is the primary 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 attentions to technical excellence and good design enhance agility.

**#10)** Simplicity — the art of maximizing the amount of work not done is much essential.

**#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.

These agile principles provide practical guidance for the development teams.

**Another way of organizing the 12 principles is to consider them in the following four distinct groups:**

* Customer satisfaction
* Quality
* Teamwork
* Project management

**#1)** **Our highest priority is to satisfy the customer through early and continuous delivery of a valuable software –** Customers are obviously going to be thrilled to see a working software being delivered every sprint rather than having to go through an ambiguous waiting period at the end of which only they will get to see the product.

Here the customer can be defined as the project sponsor or the person who is paying for the development. The end user of the product is also a customer but we can differentiate between the two as the end user is referred to as a user.

**#2)** **Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage –** Changes can be incorporated without much delays in the overall timelines.

Since the agile teams believe in quality above all things, they would rather incorporate changes and deliver as per the customer requirements than to avoid changes and deliver a product that does not serve the business needs.

**#3)** **Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale –** This is taken care of by the teams working in sprints. Since sprints are time-boxed iterations and deliver working software at the end of each sprint, customers regularly get an idea of the progress

**#4)** **Business people and developers must work together daily throughout the project –** Better decisions are taken when both are working together collaboratively and there is a constant feedback loop between the two for course correction and change agility. Communication among the stakeholders is always the key in agile.

**#5)** **Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done –** You have to support, trust and motivate the teams. A motivated team is more likely to be successful and will deliver a superior product than unhappy teams who are not willing to give their best.

One of the ways to do this is to empower the development team to be self-organized and take their own decisions.

**#6)** **The most efficient and effective method of conveying information to and within the development team is a face-to-face conversation –** Communication is better and more impactful if the teams are in the same location and can meet face to face for discussions. It helps to build trust and brings understanding among various stakeholders.

**#7)** **Working software is the primary measure of progress –** working software beats all the other KPIs and is the best indicator of the work done.

**#8)** **agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely –** Consistency of delivery is emphasized. The team should be able to maintain their pace over the duration of the project and not burn out after the first few sprints.

**#9)** **Continuous attentions to technical excellence and good design enhances agility –** The team should have all the skills and a good product design to handle the changes and produce a high-quality product while being able to incorporate changes

**#10)** **Simplicity —** the art of maximizing the amount of work not done is essential and is just enough to meet the definition of done.

**#11)** **the best architectures, requirements, and designs emerge from self-organizing teams** – Self-organized teams are empowered and take ownership of their work. This leads to open communication and regular sharing of ideas among the team members.

**#12)** **at regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly –** Self-improvement leads to quicker results and lesser rework.

#### C:\Users\pumadevi\Desktop\Scrum_Framework.png

#### Agile Model

Enlisted below is the list of both Pros and Cons of the Agile Model.

#### Agile Pros:

* One of the biggest advantages of the agile model is its **great adaptability**. Adaptability means ‘responding to change’. Agile is very flexible in dealing with the changes in customer needs and priorities.
* Allows constantly refining and re-prioritizing **the overall product backlog** without affecting the current iteration in which the team is focused on delivering the MVP. The changes can be planned for the next iteration, thereby offering an opportunity to bring in the changes within a few weeks only.
* Agile methodology offers a great degree of **stakeholder engagement**. The client and the project team meet before, during and after each sprint. As the customer is constantly involved throughout the project, there are more opportunities for the team to clearly understand the customer’s vision.
* The working software is delivered early and frequently. This increases the **stakeholder’s trust** in the team and encourages the team to **stay highly committed** to the project.
* This model gives **transparency**. Both the stakeholders and the team know well about what is happening. The client can see the work in progress.
* Fixed schedule sprints of one to four weeks allow for **early and predictable delivery**. New features are released quickly and frequently in a time-boxed manner.
* Agile is **customer-centric**. It does not just deliver the functionality but also focuses on delivering the feature that is of some value to the user. Each user story has a business focused acceptance criteria.
* Valuable **customer feedback** is gained **early** in the project and changes to the product can be made as required.
* **The focus is on business value**. It first delivers what is most important to the client.
* **Improves the quality of deliverables**. Unlike waterfall, testing is continuously and frequently done in an Agile project and that, in turn, helps in detecting and fixing the issues early.
* Agile **supports TDD (Test Driven Development) approach** which provides enough time to create unit tests for the features that are going to be released through MVP.
* Also, by producing **frequent builds**, any misalignment with the customer requirements can also be detected and fixed early.
* As we get **immediate user feedback** after each MVP release, the **risk of project failure is low,**when you are working in an Agile way.
* Agile **promotes teamwork**. There is a great collaboration, interaction, harmony and enthusiasm among the Agile team members.
* The cost and schedule estimates of each sprint are communicated to the client prior to the start of the sprint. This **improves decision making** as the user can understand the cost of each feature and prioritize accordingly.
* In an agile project, there is a room for **continuous improvement**. Lessons learned from the past sprints can be applied in the upcoming sprints.

#### Agile Cons:

* It is often seen that Agile teams have a **tendency to neglect documentation**. This is because the Agile manifesto focuses more on working software than the comprehensive documentation. However, the teams should maintain the right balance between the code and documentation.
* As it requires a high degree of customer involvement, it can **sometimes be problematic for customers**who do not have much time and interest to participate in the project.
* Does not works well if the team is lacking commitment and dedication. Waterfall requires involvement, however, Agile requires commitment.
* If the initial architecture and design are weak, then **frequent refactoring** is required.
* When compared to the waterfall, Agile is **difficult to practice**. The team members must be well versed with Agile concepts. It requires a lot of discipline and commitment to practice Agile.
* Due to re-prioritization, it is **less predictable** than what will be delivered at the end of the sprint.
* Due to time-boxed delivery and frequent re-prioritization, there are chances for a few features to not get delivered in the allocated timeline. This can lead to **additional sprints and additional costs**. This can also lead to the problem of **nebulous timelines**.
* Lack of structure when compared to the waterfall, it **demands self-disciplined, highly proficient and cross-skilled teams**. Without this, the project can really be a challenging one.
* Availability is **less of a blueprint of the final deliverable**.
* Sometimes the **external stakeholder can't resist following the Agile approach**. In such cases, training and education about Agile are required to a wide audience.

We had enough discussion about the differences between Agile & waterfall methodologies and the benefits & challenges of each. Let us now explore the differences between agile and waterfall testing.

From the perspective of software testing, it is important for us to have a fair idea about how Agile testing is different from Waterfall testing.

**Agile Vs Waterfall Comparison Table**

| **Waterfall** | **Agile** |
| --- | --- |
| Waterfall methodology is a model in which each stage of the product’s lifecycle occurs in a sequence.   The progress of the project flows gradually downwards through these phases resembling a waterfall. | Agile methodology is a model that  follows a sequential, linear and  Iterative approach. |
| This model believes in one-time massive whole delivery.  The product is delivered at the end of the SDLC. | This model believes in multiple  small chunks of delivery at  Defined time intervals.   A MVP (Minimum Viable Product) is  Delivered at the end of each sprint. |
| Its a traditional and old-fashioned approach. | Its a new and modern approach. |
| One single cycle and single release. | Repetitive number of iterations  and multiple releases. |
| It divides the software development lifecycle into different phases. | It divides the software development  Life cycle into sprints. |
| The process is treated as one single project which is further divided into different phases. | The process is divided into multiple  projects and each project has  an iteration of different stages. |
| Structured and rigid model. | This model is known for its flexibility. |
| Long-term planning scale. | Short term planning scale. |
| A long distance exists between the customer and the developer. | A short distance exists between  the customer and the developer. |
| Long time between specification and implementation. | Short time between specification and  Implementation. |
| Takes a long time to discover problems. | Problems are discovered quickly. |
| High project schedule risk. | Low project schedule risk. |
| Less ability to respond quickly to changes. | High ability to respond quickly to changes. |
| Testing phase occurs only after the completion of the development phase. | Testing is generally performed in parallel  with the development to ensure quality continuously. |
| The customer is involved only at the requirement gathering phase and after that there is no involvement of the customer.   He comes into the picture only at the time of delivery of the product. | The customer is involved throughout  the project and feedback is taken from  the customer from time to time to ensure  Customer satisfaction. |
| Suitable for projects which have clearly defined requirements and those which are not expecting changes. | Suitable for projects which have to  Evolve and those which involve changing requirements. |
| Stringently sequential process. | Highly collaborative software development process leads to better team efforts and  Quick problem-solving. |
| Exhibits a project mindset. | Introduces a product mindset and  Thus it is more customers focused. |
| Formal and hierarchical. The project manager is the decision maker. | It is Informal. The entire team is  Responsible for decision making. |
| This model anticipates that there will be no changes throughout the project. | This model is based on adaptation  And it embraces changes. |

#### #1) Scrum

Scrum is a frame work for developing complex products and systems.it is grounded in empirical process and control theory. Scrum employs on iterative and incremental approach to optimize predictability and control risk.

Scrum can easily be considered to be the most popular agile framework. The term ‘scrum’ is much considered synonymously to ‘agile’ by most practitioners. But that is a misconception. Scrum is just one of the frameworks by which you can implement agile.

The word scrum comes from the sports rugby. Where the players huddle together in an interlocked position pushing against the opponents. Each player has a defined role in their position and can play both offensive and defensive as per the demand of the situation.

Similarly, the scrum in IT believes in empowered self-managed development teams with three specific and clearly defined roles. These roles include – **Product Owner (PO), Scrum Master (SM) and the development team consisting of the programmers and testers**. They work together in iterative time boxed durations called sprints.

The first step is the creation of the product backlog by the PO. It’s a to-do list of stuff to be done by the scrum team. Then the scrum team selects the top priority items and tries to finish them within the time box called a sprint.

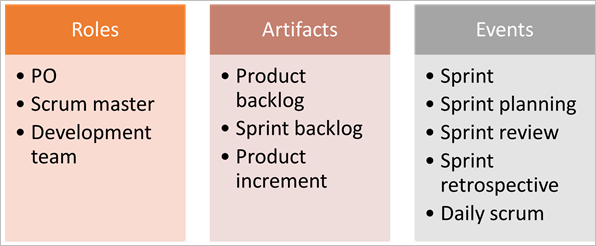
An easier way to remember all of this is to memorize the 3-3-5 framework. It means that a scrum project has 3 roles, 3 artifacts, and 5 events.

**These are** –

**Roles:**  PO, Scrum master, and development team.

**Artifacts:**  Product Backlog, Sprint Backlog and Product increment.

**Events:**  Sprint, Sprint planning, Daily Scrum, Sprint review and Sprint retrospective.



We will get to know more in detail about each of these in our subsequent tutorials.

#### #2) Kanban

Kanban is a Japanese term which means a card. These cards contain details of the work to be done on the software. The purpose is visualization. Every team member is aware of the work to be done through these visual aids.

Teams use these Kanban cards for continuous delivery. Just like Scrum, Kanban is also for helping the teams work effectively and promotes self-managed and collaborative teams.

But there are differences between these two as well – like during a scrum sprint, the items being worked upon by a team are fixed and we cannot add items to the sprint whereas, in Kanban, we can add items if there is available capacity. This is particularly useful when the requirements change frequently.

Similarly, another difference is that while the scrum has defined roles of a PO, scrum master, and development teams, there are no such pre-defined roles in Kanban.

Another difference is that while the scrum suggests prioritization of product backlogs, Kanban has no such requirement and it is totally optional. Thus Kanban requires less organization and avoids non-value adding activities and is suitable for the processes which require responsiveness towards changes.

### Important SCRUM Terminologies

**1) Scrum Team**

Scrum team is a team comprising of 7 with + or – two members. These members are a mixture of competencies and comprise of developers, testers, database people, support people etc. along with the product owner and a scrum master.

All these members work together in close collaboration for a recursive and definite interval, to develop and implement the said features. SCRUM team sitting arrangement plays a very important role in their interaction, they never sit in cubicles or cabins, but a huge table.



**2) Sprint**

Sprint is a predefined interval or time frame in which the work has to be completed and make it ready for review or ready for production deployment. This time box usually lies between 2 weeks to 1 month.

In our day to day life when we say that we follow 1-month Sprint cycle, it simply means that we work for one month on the tasks and make it ready for review by the end of that month.

**3) Product Owner**

The product owner is the key stakeholder or the lead user of the application to be developed. The product owner is the person who represents the customer side. He/she has the final authority and should always be available for the team.

He/she should be reachable when anyone has any doubts that need clarification. It is important for the product owner to understand and not to assign any new requirement in the middle of the sprint or when the sprint has already started.

**4) Scrum Master**

Scrum Master is the facilitator of the scrum team. He/she makes sure that the scrum team is productive and progressive. In case of any impediments, scrum master follows up and resolves them for the team. SCRUM Master is the mediator between the PO and the team.

He/she keeps the PO informed about the progress of the Sprint. If there are any roadblocks or concerns for the team, discusses with the PO and gets them resolved. Like the team's Daily Standup, a standup of the SCRUM Master with the PO happens every day.

**5) Business Analyst (BA)**

A Business Analyst plays a very important role in SCRUM. This person is responsible for getting the requirement finalized and drafted in the requirement docs (based on which the user stories are created).

If there are any ambiguities in the User Stories / Acceptance criteria, he/she is the one who is approached by the technical (SCRUM) team and he then takes it up to the PO or else if possible resolves on his own. In large scale projects there may be more than 1 BA but in small-scale projects, the SCRUM Master may be acting as the BA as well.

It is always a good practice to have a BA when the project kick starts.

**6) User Story**

User stories are nothing but the requirements or feature which has to be implemented.

In the scrum, we don’t have those huge requirements documents, rather the requirements are defined in a single paragraph, typically having the format as:

As a <User / type of user>  
I want to <Some achievable goal/target>  
To achieve <some result or reason for doing the thing>

**For Example:**

As an Admin I want to have a password lock in case the user enters an incorrect password for 3 consecutive times to restrict unauthorized access.

There are some characteristics of user stories which should be adhered. The user stories should be short, realistic, could be estimated, complete, negotiable and testable. A user story is never altered or changed in the middle of the Sprint.

It is the responsibility of the SCRUM Master and the BA (if applicable) to make sure that the PO has drafted the User Stories correctly with a proper set of the Acceptance Criteria”. If any changes which will impact the sprint release are to be made, then such stories are pulled out of the sprint or they are done as per the hours available.

Every user story has an acceptance criterion which should be well defined and understood by the team.

Acceptance criteria details down the user story that provides the supporting documents. It helps to further refine the user story. Anybody from the team can write down the acceptance criteria. Testing team bases their test cases/conditions on these acceptance criteria.

**7) Epics**

Epics are equivocal user stories or we can say that these are the user stories which are not defined and are kept for future sprints.

Just try to relate it with life, imagine you are going for a vacation. As you are going next week, you have everything in place like your hotel bookings, sightseeing, travelers check etc. But what about your vacation plan for next year? You only have a vague idea that you may go to XYZ place, but you have no detailed plan.

An Epic is just like you next year’s vacation plan, where you just know that you may want to go, but where, when, with whom, all these details you have no idea at this point of time.

In a similar way, there are features which are required to be implemented in future whose details are not yet known. Mostly a feature begins with an Epic and then is broken down to stories which could be implemented.

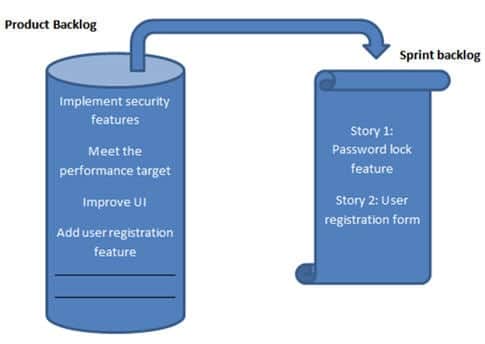
**8) Product Backlog**

The product backlog is a kind of bucket or source where all the user stories are kept. This is maintained by the Product Owner. Product backlog can be imagined as a wish list of the product owner who prioritizes it as per the business needs.

During the planning meeting (see next section), one user story is taken from the product backlog, then the team does the brainstorming, understands it and refines it and collectively decides which user stories to take, with the intervention of the product owner.

**9) Sprint Backlog**

Based on the priority, user stories are taken from the Product Backlog as one at a time. The Scrum team brainstorms on it determine the feasibility and decide on the stories to work on a particular sprint. The collective list of all the user stories which the scrum team works on a particular sprint is known as Sprint backlog.



**10) Story Points**

Story points are a quantitative indication of the complexity of a user story. Based on the story point, estimation and efforts for a story are determined.

A story point is relative and not absolute. In order to make sure that our estimate and efforts are correct, it’s important to check that the user stories are not big. The more precise and smaller is the user story, the more accurate will be the estimation.

Each and every user story is assigned to a story point based on the Fibonacci series (1, 2, 3, 5, 8, 13&21). Higher is the number, the complex is the story.

**To be precise**

* If you give 1 / 2 / 3 story point it means that the story is small and of low complexity.
* If you give points as 5 / 8, it is a medium complex and
* 13 and 21 are highly complex.

Here complexity consists of both development plus testing effort.

To decide a story point, brainstorming happens within the scrum team and the team collectively decides a story point.

It may happen that the development team gives a story point of 3 to a particular story, because for them it may be 3 lines of code change, but the testing team gives 8 story point because they feel that this code change will affect larger modules so the testing effort would be larger. Whatever story point you are giving, you have to justify it.

So in this situation, brainstorming happens and the team collectively agrees to one story point.

**Whenever you are deciding on a story point, keep the below factors in mind:**

* The dependency of the story with other application/module.
* The skill-set of the resource.
* The complexity of the story.
* Historical learning.
* Acceptance criteria of the user story.

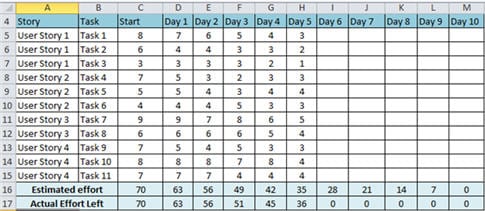
If you are not aware of a particular story, don’t size it.

Whenever a story is = or > 8 points, it is broken down into 2 or more stories.

**11) Burn down chart**

Burn down chart is a graph which shows the estimated v/s actual effort of the scrum tasks.

It is a tracking mechanism by which for a particular sprint the day to day tasks are tracked to check whether the stories are progressing towards the completion of the committed story points or not.



I have assumed:

* 2 weeks Sprint ( 10 days)
* 2 resources actual working on the sprint.

**“Story”**-> This column shows the user stories taken for the sprint.

**“Task”** -> This column shows the list of the task associated with the user story.

**“Effort”** -> This column shows the effort. Now, this measure is the total effort to complete the task. It does not depict the effort put in by any specific individual.

**“Day 1 – Day 10”** -> this column(s) shows the hours which are left to complete the story. Please see that the hour is NOT the hour which is already done BUT the hours which are still left.

**“Estimated Effort”** -> is the total of the effort. For the “Start” it is simply the sum of the entire individual task: SUM (C5: C15)

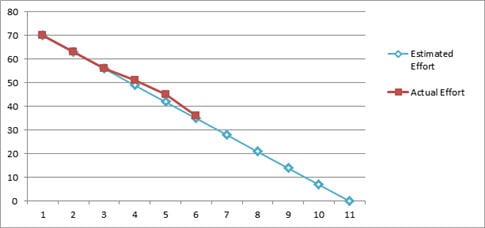
A total number of efforts that has to be completed in 1 day is 70 / 10 = 7. So at the end of day 1, the effort should reduce to 70 – 7 = 63. In a similar way, it is calculated for all the days till day 10, when the estimated effort should be 0 (Row 16)

**“Actual Effort Left”** -> As the name suggests, is the effort actually left to complete the story. It may also happen that the actual efforts increases or decreases than the estimated one.

You can use the in built functions and Chart in Excel to create this burndown chart.

**Burn down Chart steps would be:**

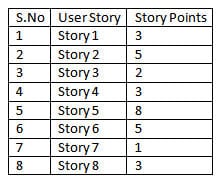
1. Enter all the stories (Column A5 – A15).
2. Enter all the Tasks (Column B5 – B15).
3. Enter the Days (Day 1 – Day 10).
4. Enter the starting efforts (Sum the tasks C5 – C15).
5. Apply the formula to calculate the “Estimated Efforts” for each day (Day 1 to Day 10). Enter the formula at D15 (C16-$C$16/10) and drag it for all the days.
6. For each day, enter the actual efforts. Enter the formula at D17 (SUM (D5:D15)) for summing up the actual efforts left, and drag it for all the other days.
7. Select it and create the chart as follows:



**12) Velocity**

The total number of story point which a scrum team archives in a sprint, is called Velocity. The Scrum team is judged or referenced by its velocity. Having said that, it should be kept in mind that the objective here is NOT achieving the maximum story points, but to have a quality deliverable, respecting the scrum team’s comfort level.

**For Example**: For a particular sprint: the total number of user stories are 8 having story points as shown below.



So here the velocity will be the sum of the story points = 30

**Definition of Done:**

A Sprint is marked as Done when all the stories are completed, all dev, research, QA tasks are marked ‘Completed', all bugs are fixed-closed else the ones that can be done later (like not completely related or are less important) are pulled out and added in the backlog, the code review and unit testing is completed, the estimated hours have met the actual hours put up in the tasks and most importantly a successful demo has been given to the PO and the stakeholders.



### Activities Done in SCRUM Methodology

#### #1) Planning Meeting

A planning meeting is the starting point of Sprint. It is the meeting where the entire scrum team gathers, the SCRUM Master selects a user story based on the priority from the product backlog and the team brainstorms on it.

Based on the discussion, the scrum team decides the complexity of the story and sizes it as per the Fibonacci series. The team identifies the tasks along with the efforts (in hours) which would be done to complete the implementation of the user story.

Many a time, the planning meeting is preceded by a “Pre-Planning meeting”. It’s just like a homework which the scrum team does before they sit for the formal planning meet. The team tries to write down the dependencies or other factors which they would like to discuss in the planning meeting.

#### #2) Execution of Sprint Tasks

As the name suggests, these are the actual work done by the scrum team to accomplish their task and take the user story into the “Done” state.

#### #3) Daily Standup

During the sprint cycle, every day the scrum team meets for, not more than 15 minutes (could be a stand-up call, recommended to have during the beginning of the day) and state 3 points:

1. What did the team member do yesterday?
2. What did the team member plan to do today?
3. Any impediments (roadblocks)?

It is the Scrum master who facilitates this meeting. In case, any team member is facing any kind of difficulties, the scrum master follows up to get it resolved. In Stand ups, the board is also reviewed and in itself shows the progress of the team.

#### #4) Review Meeting

At the end of every sprint cycle, the SCRUM team meets again and demonstrates the implemented user stories to the product owner. The product owner may cross verify the stories as per its acceptance criteria. It’s again the responsibility of the Scrum master to preside over this meeting.

Also in the SCRUM tool, the Sprint is closed and the tasks are marked done.

#### #5) Retrospective Meeting

The retrospective meeting happens after the review meeting.

The SCRUM team meets, discusses & document the following points:

* What went well during the Sprint (Best practices)?
* What did not go well in the Sprint?
* Lessons learned
* Action Items.

The Scrum team should continue to follow the best practice, ignore the “not best practices” and implement the lessons learned during the consequent sprints. The retrospective meeting helps to implement the continuous improvement of the SCRUM process.

### How the Process is done? An Example!

Having read about the technical jargons of SCRUM. let me try to demonstrate the whole process with an example.

**Example:**

**Step #1**: Let’s have a SCRUM team of 9 people comprising of 1 product owner, 1 Scrum master, 2 testers, 4 developers and 1 DBA.

**Step #2**: The Sprint is decided to follow a 4 weeks cycle. So we have 1-month Sprint starting 5th June to 4th of July.

**Step #3**: The Product owner has the prioritized list of user stories in the product backlog.

**Step #4:** The team decides to meet on 4th June for the “Pre Planning” meeting.

* The product owner takes 1 story from the product backlog, describes it and leaves it to the team to brainstorm on it.
* The entire team discusses and communicates directly to the product owner to have clearly understood the user story.
* In a similar way, various other user stories are taken. If possible, the team can go ahead and size the stories as well.

After all the discussion, Individual team members go back to their workstations and

* Identify their individual tasks for each story.
* Calculate the exact number of hours on which they will be working. Let's check how the member concludes these hours.

Total number of working hours = 9

Minus 1 hour for a break, minus 1 hour for meetings, minus 1 hour for emails, discussions, troubleshooting etc.

So the actual working hours = 6.

A total number of working days during the Sprint = 21 days.

Total number of hours available = 21\*6 = 126.

The member is on leave for 2 days = 12 hours (This varies for each member, some may take leave and some may not.)

Number of actual hours = 126 – 12 = 114 hours.

This means that the member will actually be available for 114 hours for this sprint. So he will break down his individual sprint task in such a way that a total of 114 hours is reached.

**Step #5**: On the 5th of June the entire Scrum team meets for the “Planning Meeting”.

* The final verdict of the user story from the product backlog is done and the story is moved to the Sprint Backlog.
* For each story, each team member declares their identified tasks, if required they can have a discussion on those tasks, can size or resize it (remember the Fibonacci series!!).
* The Scrum master or the team enter their individual tasks along with their hours for each story in a tool.
* After all the stories are completed, Scrum master notes the initial Velocity and formally starts the Sprint.

**Step #6**: Once the Sprint has started, based on the tasks assigned, each team member starts working on those tasks.

**Step #7**: The team meets daily for 15 minutes and discusses 3 things:

* What did they do yesterday?
* What they plan to do today?
* Any impediments (roadblocks)?

**Step #8**: The scrum master tracks the progress on a daily basis with the help of “Burn down chart”.

**Step #9**: In case of any impediments, the Scrum master follows up to resolve those.

**Step #10**: On 4th July, the team meets again for the review meeting. A member demonstrates the implemented user story to the product owner.

**Step #11**: On 5th July, the Team meets again for the Retrospective, where they discuss

* What went well?
* What did not go well?
* Action Items.

**Step #12**: On 6th July, the Team again meets for pre-planning meeting for the next sprint and the cycle continues.

