**Notes for MCA-I (Semester- II**

**Course Name :- Software Project Management**

**(Course Code:- IT22)**

**Chapter 4] Agile Teams , Size & Schedule**

**4.1] Dynamic System Development Method :-**

DSDM is a Rapid Application Development (RAD) approach to software development and provides an agile project delivery framework. The important aspect of DSDM is that the users are required to be involved actively, and the teams are given the power to make decisions. Frequent delivery of product becomes the active focus with DSDM. The techniques used in DSDM are

1. Time Boxing
2. MoSCoW Rules
3. Prototyping

**(Time Boxing** :- is the approach for completing the project incrementally by breaking it down into splitting the project in portions, each with a fixed budget and a delivery date. For each portion a number of requirements are prioritized and selected. Because time and budget are fixed, the only remaining variables are the requirements. So if a project is running out of time or money the requirements with the lowest priority are omitted)

The DSDM project consists of 7 phases

1. Pre-project
2. Feasibility Study
3. Business Study
4. Functional Model Iteration
5. Design and build Iteration
6. Implementation
7. Post-project

The **MoSCoW method** is a prioritization technique used in management, [business analysis](https://en.wikipedia.org/wiki/Business_analysis), [project management](https://en.wikipedia.org/wiki/Project_management), and [software development](https://en.wikipedia.org/wiki/Software_development) to reach a common understanding with [stakeholders](https://en.wikipedia.org/wiki/Project_stakeholder) on the importance they place on the delivery of each [requirement](https://en.wikipedia.org/wiki/Requirements_analysis); it is also known as *MoSCoW prioritization* or *MoSCoW analysis*.

The term *MoSCoW* itself is an [acronym](https://en.wikipedia.org/wiki/Acronym) derived from the first letter of each of four prioritization categories: **M - *Must have,* S - *Should have,* C - *Could have,* W - *Won't have***

**The interstitial *O*s are added to make the word pronounceable.** While the *O*s are usually in lower-case to indicate that they do not stand for anything, the all-capitals *MOSCOW* is also used.

The **Dynamic Systems Development technique (DSDM)** is an associate degree agile code development approach that provides a framework for building and maintaining systems. The DSDM philosophy is borrowed from a modified version of the sociologist principle—80 % of An application is often delivered in twenty percent of the time it’d desire deliver the entire (100 percent) application.

DSDM is An iterative code method within which every iteration follows the 80% rule that simply enough work is needed for every increment to facilitate movement to the following increment. The remaining detail is often completed later once a lot of business necessities are noted or changes are requested and accommodated.

80% of the project comes from 20% of the system requirements, so as long as those most important 20% of requirements are implemented into the system, the system therefore meets the business needs and that no system is built perfectly in the first try.

The DSDM tool (www.dsdm.org) could be a worldwide cluster of member companies that put together tackle the role of “keeper” of the strategy. The pool has outlined An [Agile Development Model](https://www.geeksforgeeks.org/software-engineering-agile-development-models/), known as the DSDM life cycle that defines 3 different unvarying cycles, preceded by 2 further life cycle activities:

1. **Feasibility Study:**

It establishes the essential business necessities and constraints related to the applying to be designed then assesses whether or not the application could be a viable candidate for the DSDM method.

1. **Business Study:**

It establishes the use and knowledge necessities that may permit the applying to supply business value; additionally, it is the essential application design and identifies the maintainability necessities for the applying.

1. **Functional Model Iteration:**

It produces a collection of progressive prototypes that demonstrate practicality for the client.

(Note: All DSDM prototypes are supposed to evolve into the deliverable application.) The intent throughout this unvarying cycle is to collect further necessities by eliciting feedback from users as they exercise the paradigm.

1. **Design and Build Iteration:**

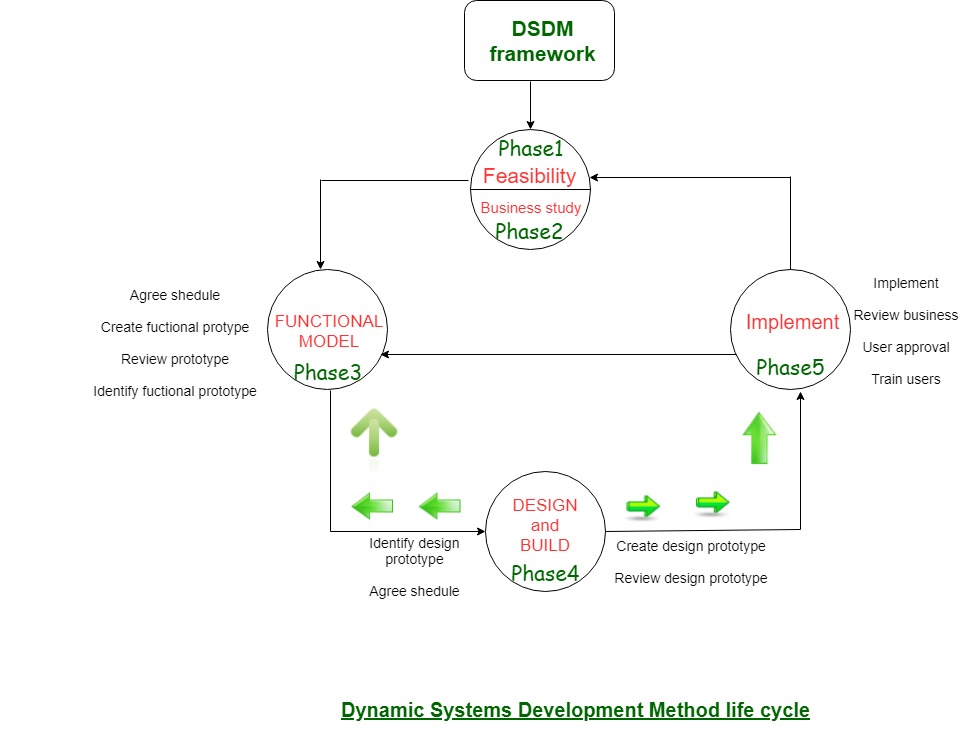
It revisits prototypes designed throughout useful model iteration to make sure that everyone has been designed during a manner that may alter it to supply operational business price for finish users. In some cases, useful model iteration and style and build iteration occur at the same time.

1. **Implementation:**  
   It places the newest code increment (an “operationalized” prototype) into the operational surroundings. It ought to be noted that:
   * **(a)** the increment might not 100% complete or,
   * **(b)** changes are also requested because the increment is placed into place. In either case, DSDM development work continues by returning to the useful model iteration activity.

**There are eight principles** underpinning DSDM . These principles direct the team in the attitude they must take and the mindset they must adopt to deliver consistently.

1. Focus on the business need
2. Deliver on time
3. Collaborate
4. Never compromise quality
5. Build incrementally from firm foundations
6. Develop iteratively
7. Communicate continuously and clearly
8. Demonstrate control

Below diagram describe the DSDM life cycle:



DSDM is often combined with XP to supply a mixed approach that defines a solid method model (the DSDM life cycle) with the barmy and bolt practices (XP) that are needed to create code increments. additionally, the ASD ideas of collaboration and self-organizing groups are often tailored to a combined method model

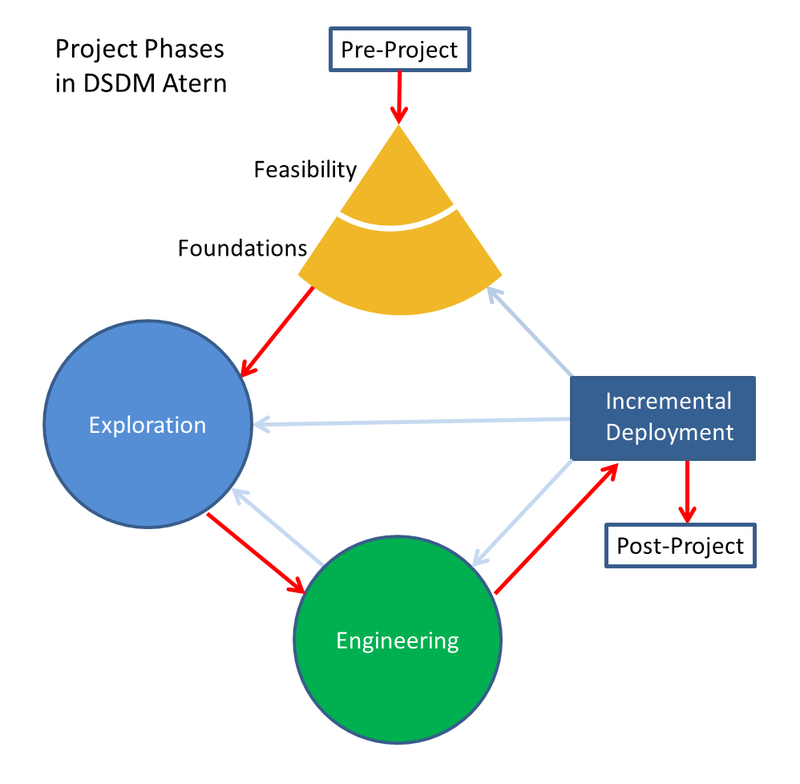


Fig. :- Model of DSDM Project Management Method

**Roles in DSDM** :-

There are some roles introduced within DSDM environment. It is important that the project members need to be appointed to different roles before they commence the project. Each role has its own responsibility. The roles are:

* **Executive Sponsor** So called the “Project Champion”. An important role from the user organisation who has the ability and responsibility to commit appropriate funds and resources. This role has an ultimate power to make decisions.
* **Visionary** The one who has the responsibility to initialise the project by ensuring that essential requirements are found early on. Visionary has the most accurate perception of the business objectives of the system and the project. Another task is to supervise and keep the development process in the right track.
* **Ambassador User** Brings the knowledge of the user community into the project, ensures that the developers receive enough user feedback during the development process.
* **Advisor User** Can be any user that represents an important viewpoint and brings daily knowledge of the project.
* **Project Manager** Can be anyone from the user community or IT staff who manages the project in general.
* **Technical Co-ordinator** Responsible in designing the system architecture and control the technical quality of the project.
* **Team Leader** Leads their team and ensures that the team works effectively as a whole.
* **Solution Developer** Interpret the system requirements and model it including developing the deliverable codes and build the prototypes.
* **Solution Tester** Checks the correctness in a technical extent by performing some testing, raise defects where necessary and retest once fixed. Tester will have to provide some comment and documentation.
* **Scribe** Responsible for gathering and recording the requirements, agreements, and decisions made in every workshop.
* **Facilitator** Responsible for managing the workshops' progress, acts as a motivator for preparation and communication.
* **Specialist Roles** Business Architect, Quality Manager, System Integrator, etc.

**Critical success factors :-**

Within DSDM a number of factors are identified as being of great importance to ensure successful projects.

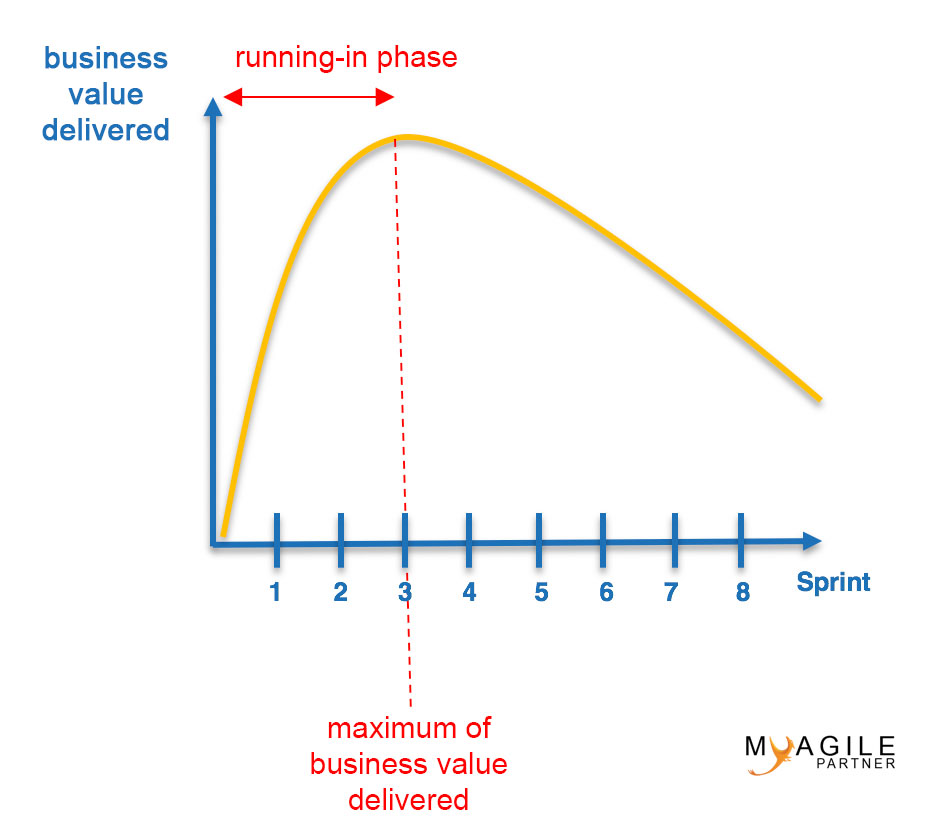
* **Factor 1**: First there is the acceptance of DSDM by senior management and other employees. This ensures that the different actors of the project are motivated from the start and remain involved throughout the project.
* **Factor 2**: Directly derived from factor 1: The commitment of the management to ensure end-user involvement. The prototyping approach requires a strong and dedicated involvement by end users to test and judge the functional prototypes.
* **Factor 3**: The project team has to be composed of skillful members that form a stable union. An important issue is the empowerment of the project team. This means that the team (or one or more of its members) has to possess the power and possibility to make important decisions regarding the project without having to write formal proposals to higher management, which can be very time-consuming. In order to enable the project team to run a successful project, they also need the appropriate technology to conduct the project. This means a development environment, project management tools, etc.
* **Factor 4**: Finally, DSDM also states that a supportive relationship between customer and vendor is required. This goes for both projects that are realized internally within companies or by external contractors.

**DSDM characteristics:**

* prioritize requirements and work though them iteratively, building a system or product in increments.
* Its tool-independent frameworks. This allows users to fill in the specific steps of the process with their own techniques and software aids of choice.
* The variables in the development are not time/resources, but the requirements. This approach ensures the main goals of DSDM, namely to stay within the deadline and the budget.
* A strong focus on communication between and the involvement of all the stakeholders in the system. Although this is addressed in other methods, DSDM strongly believes in commitment to the project to ensure a successful outcome.
* **4.2 Value-Driven Development :-**

The [**agile manifesto**](https://www.myagilepartner.com/blog/index.php/2019/02/13/agile-manifesto-4-values-12-principles/) remember us : “Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.”. And in Value Driven-Development, we try to deliver a maximum of values as soon as possible

To follow this curve where it deliver a maximum of values as soon as possible. The team will live a running-in phase at the beginning to setup environment, to create interaction between people, to create the framework.

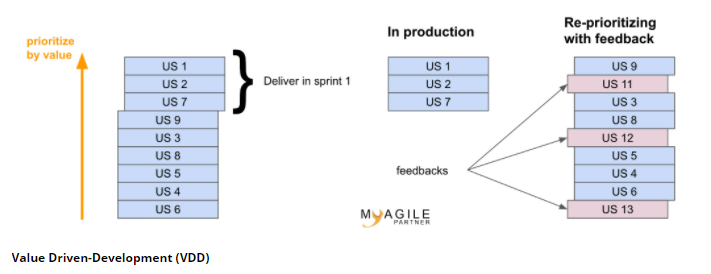


In order to provide Value-driven Delivery, it is important to meet a few requirements. First, those involved in a project should understand what adds value to customers and users, and should prioritize the high value requirements in the Prioritized Product Backlog. Second, those involved in a project should decrease uncertainty and constantly address risks that can potentially decrease value if they materialize; it is important to show product increments to project stakeholders at the end of each Sprint, enabling effective management of changes. Third, those involved in a project should create deliverables by producing potentially shippable, value-based product increments during each Sprint so that customers start realizing value early in the project.

The concept of Value-driven Delivery in Scrum makes the Scrum framework very attractive for business stakeholders and senior management. This concept is very different when compared with traditional project management models where requirements are not prioritized by business value, and value is realized only at the end of the project when the final product or service is delivered.

**Plan Driven Method is like Water fall model** where Business team will prove whole bunch of requirement , till requirement get fulfill it can’t get functional to software team to Analysis , Design, development Testing, Release and final product here customer will able to check software first time as final product so Plan Driven Method is **Rigid** , have **Low Visibility** and Its **Plan Centric**

Value Driven Method the requirement bases on Valuable requirement having high priority will be implemented in first Iteration or Sprint 1 of Development break the iteration in 3 to 4 weeks and deliver product and customer get involved in each iteration to get his feedback continuously so Value Driven Method is **Flexible** , have **High Visibility** & Its **Customer Centric**



* **4.3 Team and Roles of an Agile Team :-**
* **4.3.1 Scrum Master :-**

**Scrum is a framework** that helps agile teams to work together. Using it, the team members can deliver and sustain the complex product. It encourages the team to learn through practice, self-organize while working on the problem. Scrum is a work done through the framework and continuously shipping values to customers.

It is the most frequent software that is used by the development team. Its principle and lessons can be applied to all kinds of teamwork. Its policy and experiences is a reason of popularity of Scrum framework. The Scrum describes a set of tools, meetings, and roles that help the teams structure. It also manages the work done by the team

Scrum is Agile Development Framework for managing product development. It is flexible and holistic product development approach. In this methodology development and QA teams work as one unit.

The term is borrowed from [rugby](https://en.wikipedia.org/wiki/Rugby_football), where a [scrum](https://en.wikipedia.org/wiki/Scrum_(rugby)) is a formation of players. The term *scrum* was chosen by the paper's authors because it emphasizes teamwork



Scrum and agile are not the same thing because Scrum focused on continuous improvement, which is a core foundation of agile. Scrum framework focuses on ongoing getting work done.

**Scrum In Brief :-**

1. Scrum is a framework of agile methodology. In which **incremental builds** are delivered to end user in every two to three weeks.
2. Scrum's team is **self-organized**, cross-functional team.
3. Scrum is used in the project where the requirement **rapidly** changes.
4. A compared to agile it is more **rigid**. So that there are no chances of frequent change.
5. In **daily stand up meeting** the teamwork is achieved with a fixed role assigned to team members, scrum master, and product owner.
6. **No need to change many more** while implementing scrum process.
7. In this process, a **build is delivered** after each sprint to the client for their feedback.
8. After every sprint a demonstration of functionality is provided. So that the **regular feedback** can be taken before next sprint.
9. There is no team leader, so the **entire team handles the issues** or problems.
10. When the team completes the **current sprint activity**, then the next sprint is planned.
11. Design and execution can be **innovative and experimental**.
12. The **daily sprint meeting** is organized to review the feedback to decide the future progress of the project.
13. Working software is **not a fundamental measure**.
14. The target of the Scrum team is to deliver **maximum business value**.

The **Scrum Master** is a team leader and facility provider who helps the team member to follow agile practices, so that the team member meets their commitments and customers requirements. The scrum master plays the following responsibilities:

* They enable the close co-operation between all the roles and functions.
* They remove all the blocks which occur.
* They safeguard the team from any disturbances.
* They work with the organization to track the progress and processes of the company.
* They ensure that Agile Inspect & Adapt processes are leveraged correctly which includes
  + Planned meetings
  + Daily stand-ups
  + Demo
  + Review
  + Retrospective meetings, and
  + Facilitate team meetings and decision-making process.

**What is Scrum Master?**

Scrum master is a person who helps other people to understand Scrum and serves the project team by removing obstacles. He also helps in simplifying project complexities.

The Scrum master also needs to make sure that development team works based on the core values of Scrum. He is often considered a coach for the team, helping the team do the best work they possibly can. Moreover, he ensures that the Scrum adoption is successful in the enterprise.

Scrum master acts as a heart of Scrum project. He needs to perform responsibilities like:

* Scrum Master needs to assure that team meets its business objectives
* Scrum Master fosters collaborative environment in the team
* The Scrum master does the planning, team backlog grooming, sprint demo, sprint retrospective.
* Guides Scrum processes and helps to maintain integrity of Scrum values
* Promote improved engineering practices like TDD, automated testing and continuous integration.
* Making sure that every stakeholder should be present at the meeting at the given time
* Works together with other team members to ensure dependencies and risk are distributed across Scrum teams.
* Conducting feasibility studies, writing & validating specifications.

**What is Scrum master is not?**

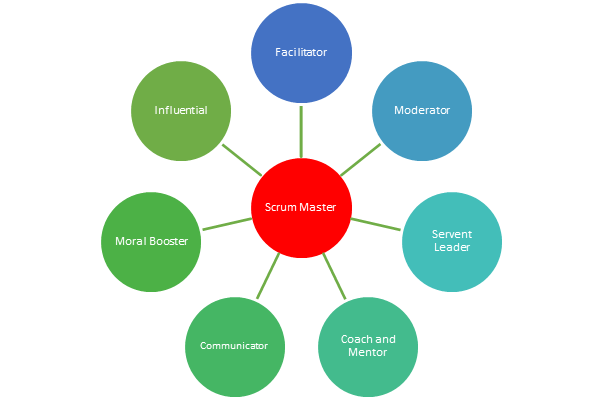
The Scrum master is not a project manager. Both roles are different in the agile process. He will not guide a team in every phase of the development. He is neither responsible for any technical decision nor for managing the business with clients. He will never lead the team or make decisions on behalf of the team.

* **Scrum Master Skills**

The Scrum master is a very important person in the agile Scrum process. He should have multiple skills including technical, scrum process and soft skills.

Scrum master must know different techniques and practices to manage self-organizing teams. He should show honesty, integrity, trust, and respect for the team. He should have expertise in Agile, IT coaching, and presentation. Moreover, Scrum master should able to schedules meetings in such a way that it doesn't affect the regular work of the team members.

**Characteristic of the Good Scrum master**

[](https://www.guru99.com/images/1/120817_0726_ScrumMaster1.png)

* **Characteristic of Scrum Master**

**Servant-Leader:** Servant leadership is a social leadership style. Scrum master act as a servant leader to attain project objectives through service to others.

**Morale booster:** The Scrum master also acts as a Morale booster by encouraging the team to do their work more effectively.

**Facilitator**: Scrum master plays a role of facilitator as he helps the team to understand their common project-related objective. He also assists the team to achieve these objectives while remaining neutral.

**Coach:** Scrum masters coaches the team to become self-organized and also guides them on the path of continuous improvement.

**Agent:** Scrum master also plays a role of an agent which helps to make the necessary changes allowing a better implementation of Scrum.

**Mentor:** The Scrum Master should share his knowledge, experiences, and even his opinions just like a mentor.

**Remover of obstacles:** Scrum master should remove all the obstacles preventing the development team to work in the best conditions.

**Manager of the framework:** The Scrum Master is also responsible for the framework within which the team works.

**Observer:** He needs to constantly observe everything even the smallest activities of the team members.

* **Daily Activities of Scrum master:-**

**Activity 1:** Capacity and velocity planning using spreadsheet and Jira.

**Activity 2:** Determine implementable stories available with points remaining from the sprint review.

**Activity 3:** Review previous sprint velocity along with the team.

**Activity 4:**Estimating teams' capacity

**Activity 5:**Calculating number of points completed by team

**Activity 6:** The team member provides all known tasks and estimates task effort within an hour. Scrum Master should ensure task given to members are never above their capacity.

**Activity 7:** Scrum master also needs to ensure that the tasks in never more than 16 hours. If at all it is bigger then that he should tell the team to make it in the break them down.

The above is just a small sample of the many activities a Scrum Master is involved in.

* **Best practices for Scrum master**
* Scrum master should listen to all the queries of developers, project manager, and owner to understand their issues.
* He should entrust his unwavering faith in his team members that they will complete the allocated work.
* Scrum master should avoid doing multitasking in his/her work
* He should give respect even to the smallest team member and listen to everyone's perspective.
* Show genuine appreciation to the team for their successful accomplishments

**Why is Scrum master's role is difficult?**

First of all, management of large size self-organizing teams is not an easy task. It needs lots of willpower. Scrum master often needs to manage conflicts between team members and project manager.

He may head the entire Scrum management project still he remains without any power. Scrum master also needs to learn about motivation, communication, behavioral economics and a whole host of other social science which surely needs lots of time outside of his job.

* **4.3.2 Product Owner**

The Product Owner is one who runs the product from a business perspective. The Product Owner plays the following responsibilities:

* He defines the requirements and prioritizes their values.
* He sets the release date and contents.
* He takes an active role in iteration and releasing planning meetings.
* He ensures that the team is working on the most valued requirement.
* He represents the voice of the customer.
* He accepts the user stories that meet the definition of done and defined acceptance criteria.

The Product Owner (PO) is a member of the Agile Team responsible for defining Stories and prioritizing the Team Backlog to streamline the execution of program priorities while maintaining the conceptual and technical integrity of the Features or components for the team.

**Roles & Responsibilities of Product Owner :-**

* 1. **Defining the vision:-**

The agile product owner is the point person on the product development team, using their high-level perspective to define goals and create a vision for development projects.

Product owners are responsible for communicating with stakeholders across the board, including customers, business managers, and the development team to make sure the goals are clear and the vision is aligned with business objectives.

* 1. **Managing the product backlog**

One of the most important responsibilities for a scrum product owner is managing the product backlog. This is the development team’s project to-do list.

The product owner’s responsibility is to create the list of backlog items and prioritize them based on the overall strategy and business objectives. Additionally, the product owner will need to map out project dependencies to inform the necessary sequence of development.

The product backlog isn’t a static to-do list though. It is a live document that should be continually updated based on evolving project needs throughout development.

Because the product backlog will change frequently, the product owner must make the list accessible and available to all stakeholders (particularly developers) to ensure optimized performance and project outcomes.

* 1. **Prioritizing needs**

Another key role of the product owner is to prioritize needs. In other words, they must juggle the triangle of scope, budget, and time, weighing priorities according to the needs and objectives of stakeholders.

For example, if the product under development needs to launch within six months, that constrains the scope of the project. As the project evolves, the product owner will have to gauge which areas have flexibility and which don’t to determine how and when each iteration and product element will be developed.

**4. Overseeing development stages**

With the vision, strategy, and product priorities set, the product owner should spend a significant amount of time overseeing the actual development of the product. They are a key player throughout each event, including planning, refinement, review, and sprint.

During the planning stages, the agile product owner works with stakeholders to identify and organize the steps required for the next iteration. They will then meet with their team to refine the process, identify areas for improvement, and support the sprint.

**5. Anticipating client needs**

The successful scrum product owner will be an expert at understanding and anticipating the client’s needs to more effectively manage the development process.

Their deep market knowledge and communication skills allow them to anticipate problems or needs and address them.

Stay one step ahead of your clients with [customer journey mapping](https://www.lucidchart.com/blog/how-to-build-customer-journey-maps). These visuals create a shared vision for the customer experience. You'll save time and effort so you can put all your energy into reading your customers’ minds and wowing your clients.

1. **Acting as primary liaison**

The product owner is also the primary communicator and link between stakeholders and teams. As such, they have to be expert communicators, making sure there’s buy-in from stakeholders on all major decisions and strategy and clear instructions and deliverables for the developers.

**7. Evaluating product progress at each iteration**

The product owner is accountable for each stage of the development process and the final product. They take a primary role in inspecting and evaluating product progress through each iteration. The product owner makes the judgment call on the performance, deciding if the team needs to go back to the drawing board or if they can move on to the next steps.

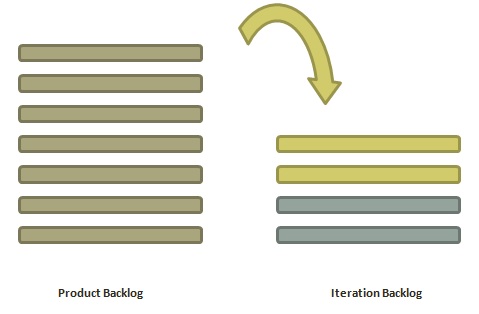
**A product backlog** is a list of items to be done. Items are ranked with feature descriptions. In an ideal scenario, items should be broken down into user stories.

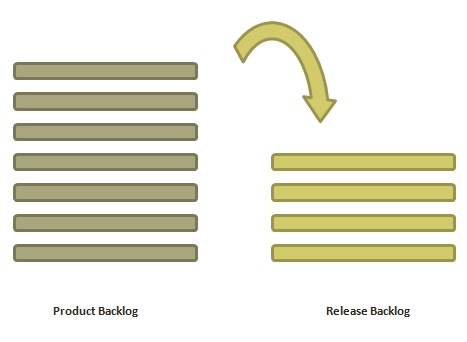
**Why Product Backlog is Important?**

* It is prepared so that estimates can be given to each and every feature.
* It helps in planning the roadmap for the product.
* It helps in re-ranking the features so that more value can be added to the product.
* It helps in determining what to prioritize first. Team ranks the item and then builds value.

**Characteristics of Product Backlog**

* Each product should have one product backlog which can have a set of large to very large features.
* Multiple teams can work on a single product backlog.
* Ranking of features is done based on business value, technical value, risk management or strategic fitness.
* Highest ranking items are decomposed into smaller stories during release planning so that they can be completed in future iterations.





* **4.3.3 Development Team:-**

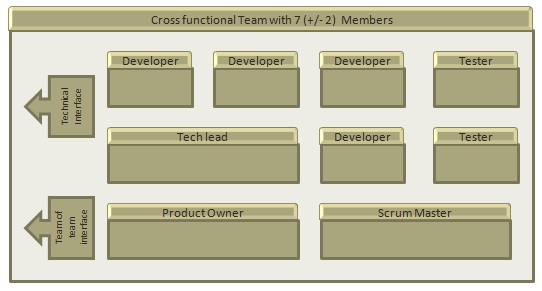
A Development Team is a group of people that work together to create software. This is complex, creative work that requires adaptability as technical challenges arise and business requirements evolve. An agile development team will seek to meet these challenges by applying the principles of cross-functionality and self-organization.

**Cross-functional**

Rather than organizing people into specialist teams such as analysts, coders, and testers; agile development teams have all of the skills to turn requirements into production-ready product. This does reduce pure efficiency and utilization – there may be times that there is nothing to do but test, so coders and analysts will have to help out. The pay-offs for this compromise include:

* Reduced dependencies and co-ordination required to get a feature into production, increasing predictability.
* Faster feedback cycles: mature teams are able to get from idea to deployed feature in a matter of weeks.
* Increased control and visibility.
* Enhanced risk management for decisions made on requirements and technology.

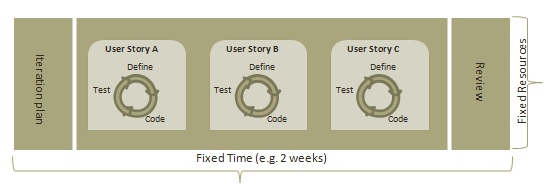
Every agile team should be a self-sufficient team with 5 to 9 team members and an average experience ranging from of 6 to 10 years. Typically, an agile team comprises of 3 to 4 developers, 1 tester, 1 technical lead, 1 product owner and 1 scrum master.



Product Owner and Scrum master are considered to be a part of Team Interface, whereas other members are part of Technical Interface.

## How an Agile Team Plans its Work?

An Agile team works in iterations to deliver user stories where each iteration is of 10 to 15 days. Each user story is planned based on its backlog prioritization and size. The team uses its capacity − how many hours are available with team to work on tasks − to decide how much scope they have to plan.



### Point

A Point defines how much a team can commit. A point usually refers to 8 hours. Each story is estimated in points.

### Capacity

Capacity defines how much an individual can commit. Capacity is estimated in hours.

**Self-organizing**

There is no project manager assigning tasks to an agile development team, nor a defacto “team lead” amongst the developers who keeps everyone busy. Self-organization is built on the premise that if no single person is in charge, then the whole team needs to stay vigilant and engaged in their shared goals. This is a radical departure from traditional development, where your responsibility as an individual technologist ends at delivering your work package and reporting issues. Now, individuals share whole-team accountability for their actions and decisions. If implemented well, this is likely to mean:

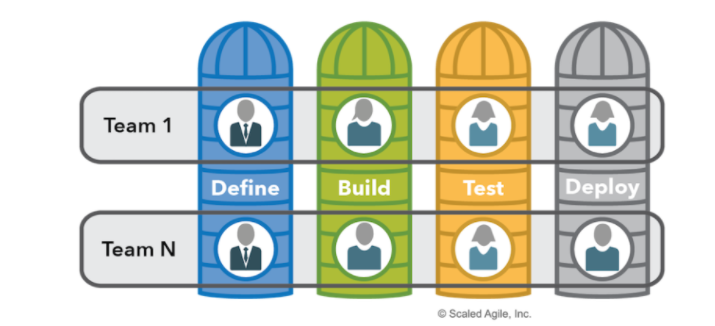
* Higher motivation arising from greater autonomy and trust.
* Increased innovation, resulting in new feature ideas and technical solutions.
* Reduced defects as the whole team takes shared accountability for overall outcomes.
* Efficient problem resolution: teams are able to act without escalating up the chain of command and waiting to be told what to do.

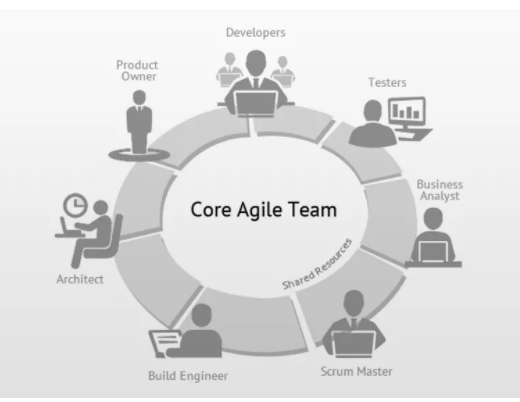
**Define –** Independently elaborate and design features and stories to accomplish their mission

**Build –** Contain all skills necessary to create the artifacts to meet their mission

**Test** – Ensure an artifact’s quality and performance

**Deploy–**Where applicable, deploy increments of value

****

****

* **4.4 Product Vision & Product Roadmap :-**

A **Company Mission** Statement defines the company’s business, its objectives and its approach to reach those objectives. A Vision Statement describes the desired future position of the company. Elements of Mission are often combined to provide a statement of the company’s purposes, goals and values.A vision will be timeless and not connected to technology or trends.

A **Product Vision** represents the core essence of its product or product line. It also sets the direction for where a product is headed, or the end state for what a product will deliver in the future.

The **product strategy** describes who your customers are, how your product fits into the current market, and how it will achieve business goals. The product strategy should describe who the product is for and why people would want to buy and to use it; what the product is and why it stands out; and what the business goals are and why it is worthwhile for your company to invest in it

The vision gives rise to specific product goals. These product goals could be things that must be accomplished over the next year — for example, growing revenue, reducing churn, and improving customer satisfaction.

**Product Strategy** is a system of achievable goals and visions that work together to align the team around desirable outcomes for both the business and your customers.

**Example for Uber** :- Decrease wait times in cities where it is over 10 minutes to less than 5 minutes by January 31, 2023" along with supporting shorter-term (a few months) goals like “onboard at least one driver for every 50 people in each city by January 31, 2022"

The **Product Roadmap** describes what products and features will be built to realize the strategy and vision, who is responsible for building those product features, and, sometimes, an estimate of when those products and features will be released.

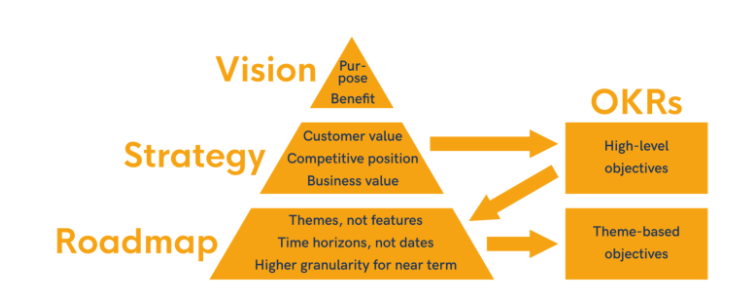
Because it’s a strategic document, what you want to convey in your roadmap is not a series of features or solutions but themes around the customer problems you intend to focus on and why.

Theme-based roadmaps start with a problem statement and move towards a solution, rather than start with features and struggle to keep up with a far more rigid roadmap.

The **vision** makes most sense when it describes customer benefits at a high level, without going into details in how they will be achieved, what the differentiation and market position is etc. This makes the vision future-proof so that it can truly serve as long-term guidance.

The **product strategy**, in my mind, needs to be explicit about “how we will win”. This means talking about how value will be delivered to the customer/user, how the product is differentiated from the competition, and what the business model is. Ideally it also describes how you will build up a sustainable competitive advantage, and **what** [**growth loops**](https://hackernoon.com/how-to-grow-a-product-a-beginners-guide-3372436cbd05) you will employ.

**product roadmap**, a theme-based roadmap should be used, with higher fidelity for themes that are currently being worked on, and much less fidelity (most likely only a description of problems/opportunities and goals) for anything further out. In the same vein as for the product strategy, goals for individual themes can be defined as part of the goal setting process.



Where **OKR** = **O**bjective **K**ey **R**esult

OKRs have two important parts: The **Objective** you want to achieve and the **Key results,** which are the way you measure achieving the objective.

“**Product Vision** specifies the What and Why of the **product**, while **Product Strategy** elaborates how to realize the **vision with a** specific approach, and provides a **roadmap** showing a timeline for executing the **strategy”**

A **product roadmap** is a plan of action for how a product or solution will evolve over time. Product owners use roadmaps to outline future product functionality and when new features will be released. When used in agile development, a roadmap provides crucial context for the team's everyday work and should be responsive to shifts in the competitive landscape

**Internal roadmap for development team:**These roadmaps can be created in several ways, depending on how your team likes to work. Some common versions include the detail about the prioritized customer value to be delivered, target release dates and milestones. Since many development teams use agile methodologies, these roadmaps are often organized by sprints and show specific pieces of work and problem areas plotted on a timeline.

**Internal roadmap for executives:**These roadmaps emphasize how teams' work supports high-level company goals and metrics. They are often organized by month or by quarter to show progress over time towards these goals, and generally include less detail about detailed development stories and tasks.

**Internal roadmap for sales:**These roadmaps focus on new features and customer benefits in order to support sales conversations. An important note: avoid including hard dates in sales roadmaps to avoid tying internal teams to potentially unrealistic dates.

**External roadmap:**These roadmaps should excite customers about what’s coming next. Make sure they are visually appealing and easy to read. They should provide a high-level, generalized view of new features and prioritized problem areas to get customers interested in the future direction of the product.

To **build a roadmap**, product owners should consider market trajectories, customer value propositions, strategic goals, and effort constraints. Once these factors are understood, the product owner can work with their team to start prioritizing initiatives and epics on the roadmap.

The content of a roadmap will depend on its audience - a roadmap for the development team may cover only one product, while a roadmap for executives can cover multiple products. Depending on the size and structure of an organization, a single roadmap may span multiple teams working on the same product. An external roadmap will often cover multiple products aligned with one point of emphasis or customer need.

The most important : create a roadmap that your audience can easily understand. Providing too much or too little detail on the roadmap can make it easy to gloss over, or worse, to too intimidating to read. A roadmap with just the right amount of detail and some visual appeal can earn the buy-in you need from key stakeholders.

Building and maintaining product roadmaps is an ongoing process to embark upon with your team. There are a few simple ways to set up for success:

* Only include as much detail as necessary for your audience
* Keep the roadmap evenly focused on short-term tactics and how these relate to long-term goals
* Review roadmaps on a regular basis and make adjustments when plans change
* Make sure everyone has access to the roadmap (and checks it on a regular basis)
* Stay connected with stakeholders at all levels to ensure alignment
* Understand your Product Vision
* Find the right amount of flexibility
* Inspire a discussion around team goals
* Communicate your agile product roadmap at the right level
* Focus on Outcomes over Feature

The **product roadmap** is a shared resource of truth. It is the outline of the vision, priorities, direction, and progress of product over time. The roadmap is the plan of action which aligns the organization around short-term and long-term goals for the product or project. The roadmap also plans how the product and project will be achieved.

The item on the roadmap should be clearly linked to your product strategy. It should also be responsive to changes in customer feedback and the competitive landscape. The product owner uses this roadmap to collaborate with their teams and build consent on how a product will grow and shift over time. This roadmap provides a team to keep everyone on the same page and gain context for their everyday work and future direction.

**Tips for presenting Product Roadmaps**

1. Set the right context
2. Consider commitments carefully
3. Make realistic plans
4. Think big, start small
5. Build a business case
6. Balance mundane with motivating
7. Roll with the punches
8. Be open and honest

* **4.5 Project Objective & Key Metrics :-**

**Objectives**: Qualitative descriptions of what you want to achieve. Objectives should be short, inspirational, and engaging. An objective should motivate and challenge the team.

For Example

* Improve customer satisfaction
* Increase recurring revenue
* Scale system performance
* Increase the number of customers served
* Reduce the number of data errors in the system

**Metrics** are nothing but standards of measurement. Agile metrics are standards that help a software team in monitoring how productive a team is across the different phases of the SDLC.

Agile metrics are an essential component of the development process. For companies or teams that work on the agile framework, agile metrics help in assessing software quality.

By measuring how productive a team is, agile metrics help keep the team performance in check. If there are any loopholes, they expose them at the initial stages. Since the data and its usage are measurable, it’s easier to work on the shortcomings with the help of these metrics. For example, velocity metrics can help you track your team’s output.

**Importance of Agile Metrics**

Now that we know what agile metrics are, let’s break down how they work. The entire concept of agile rests on [Continuous Improvement](https://www.plutora.com/blog/continuous-improvement-cicd-pipeline) (CI). But this isn’t something that you can impose on teams. It has to come from within. In short, Self-Improvement (SI) is a must. So, it’s safe to say that CI isn’t possible without SI.

Immediate delivery is an important component of agile. But one shouldn’t overlook SI in this case. Teams practicing SI give better results than those who don’t. But having a sustainable and effective SI is not child’s play. It’s a long-term process and needs a management framework. By tracking software quality and team performance, agile metrics support SI. In a way, these metrics directly impact CI.

Apart from improving continuity, delivering a high-quality product is also a vital part of agile. However, striking the balance between these two can be challenging. This gives rise to the need for metrics against which teams can measure progress. All in all, agile metrics help teams become self-managing. They also help firms in delivering value. At the same time, CI becomes a part of the workflow without much effort.

**Agile Metrics Important for Your Project**

Agile metrics measure different aspects of project development. Here are some agile metrics important for your project. They’ll help you understand the development process better. Additionally, they’ll ease the process of overall [software release](https://www.plutora.com/software-release-management/software-release).

**1. Sprint Burndown Report**

An agile framework comprises [scrum](https://www.plutora.com/blog/scrum-vs-agile) teams. They organize their processes into sprints. Since a sprint is time-bound, it’s important to track task progress frequently. A **sprint burndown report** is for tracking the completion of different tasks during a sprint. Time and work left to complete are the two main parameters of measurement in this case. **The X-axis refers to the time. The Y-axis represents the work left**. The unit of measurement is hours or story points. The team forecasts the workload at the beginning of a sprint. The target is to complete the workload by the end of the sprint.

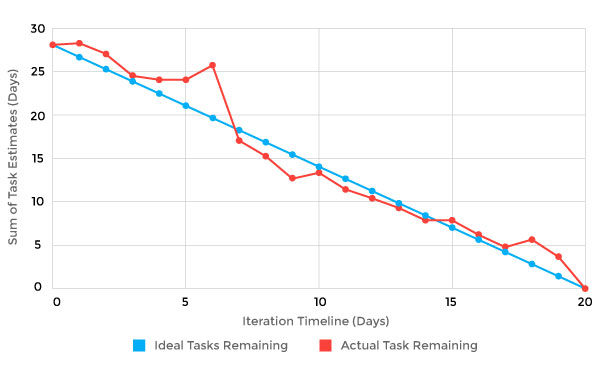


Fig:- [*Sprint Burndown Chart*](https://en.wikipedia.org/wiki/Burn_down_chart#/media/File:Burn_down_chart.png)

**2. Velocity**

Velocity measures the average work a team does during a sprint. The report, in this case, contains several iterations. The accuracy of the forecast depends on the number of iterations. The more iterations, the more precise the forecast. The unit of measurement is hours or story points. Velocity also determines the ability of a team to work through backlogs. As time passes, velocity tends to evolve. To ensure consistent performance, it’s important to track velocity. If the velocity declines, it’s a sign that the team needs to fix something.

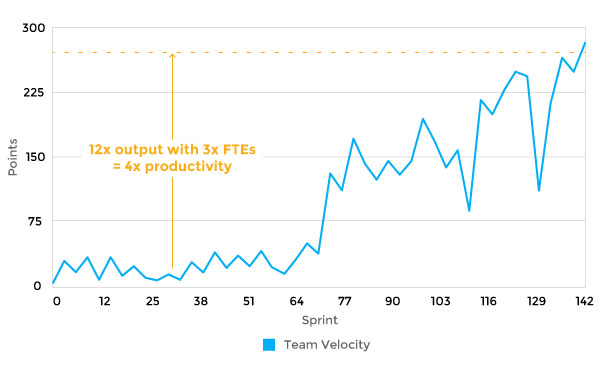


Fig :- [*Agile Velocity*](https://www.scruminc.com/velocity/)

**3. Epic and Release Burndown**

Unlike a sprint burndown, epic and release burndown focus on the bigger picture. They track progress over a large work body. There are many epics and versions of work in a sprint. So, it’s important to track their progress as well as each sprint. The entire team has to be aware of workflow in the epic and version. Epic and release burndown charts make that possible.

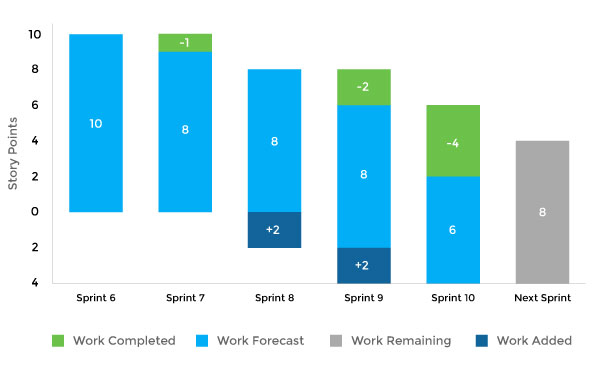


Fig :- [*Epic and Release Burndown*](https://www.researchgate.net/figure/Epic-and-release-burndown_fig2_321741375)

**4. Control Chart**

In agile, control charts focus on the time duration from the “in progress” to “complete” status of tasks. Their purpose is to check the cycle time of a single issue. Teams with consistency in cycle times have predictable deliveries. Besides this, teams with short cycle times have a high throughput. When teams measure cycle times, they improve the flexibility of their processes. For instance, in the case of changes, you can discern the results instantly. As a result, team members can make the necessary adjustments. In general, a short and consistent cycle time is the target to achieve in every sprint.

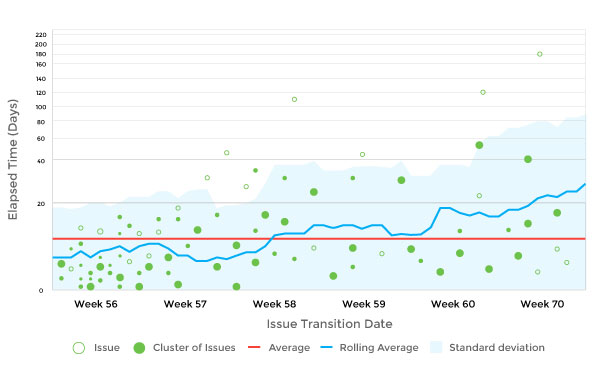


Fig:-[*Control Chart*](https://www.atlassian.com/blog/jira-software/6-key-ways-optimize-development-control-chart)

**5. Cumulative Flow Diagram**

The cumulative flow diagram (CFD) ensures consistency in workflow across the team. The X-axis represents time. The number of issues is on the Y-axis. Ideally, the diagram should be smooth from left to right. Smoothen out the color bands in case of uneven flow. The band narrowing means throughput is higher than the rate of entry. If the band widens, this means that your workflow capacity is greater than required, and it can be moved elsewhere to smoothen the flow.

The CFD measures the state of the work in progress. With that, you can take measures to speed up the workflow. The diagram provides a clear visual representation of bottlenecks. You can analyze how bottlenecks formed in the first place. After that, the team can take steps to eliminate them and make improvements.

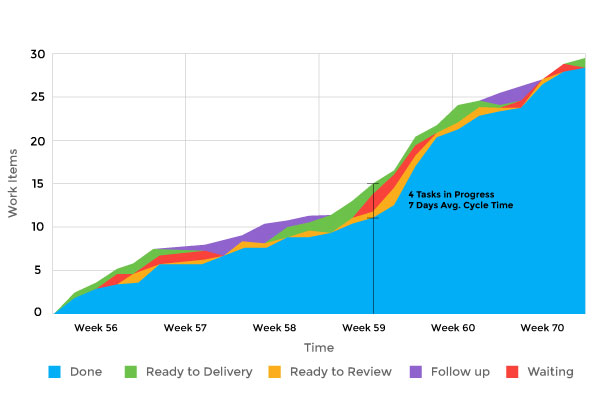


Fig:- [*Cumulative Flow Diagram*](https://kanbanize.com/kanban-resources/kanban-analytics/cumulative-flow-diagram/)

**6. Lead Time**

Lead time is the period between the moment of making a request for delivering a product and the actual delivery. All the processes to bring a product to completion come under lead time. It also includes developing a business requirement and fixing bugs. Lead time is an important metric. The reason for this is it provides the exact time calculation for every process.

**7. Value Delivered**

Here, project managers assign value to every requirement. This metric uses either dollars or a points system. Implementing features with high value should be the top priority. An upward trend in this metric shows that things are on track. On the other hand, a downward trend isn’t a good sign. It means the implementation of lower-value features is going on. If that’s the case, the team should make amends. Sometimes, you might even have to stop product development.



[Fig:- *Value Delivered*](https://www.scrum.org/resources/blog/measuring-success-measuring-value)

**8. Net Promoter Score**

Net Promoter Score measures how much the customers are willing to recommend the product or service to others. It’s an index that ranges from -100 to 100. Customer loyalty is an important factor to determine the success of a firm. You can use the Net Promoter Score as a proxy for this purpose.

**9. Work Item Age**

Work item age is the aging work in progress. This metric indicates the time that passes between the start and completion of the current task. The use of work item age is to detect the timeline for unfinished tasks. By using this metric, you’ll realize how your present tasks move forward. You can also compare your previous performance in the same context as the current scenario. The measurement tool, in this case, is the aging work in progress chart.

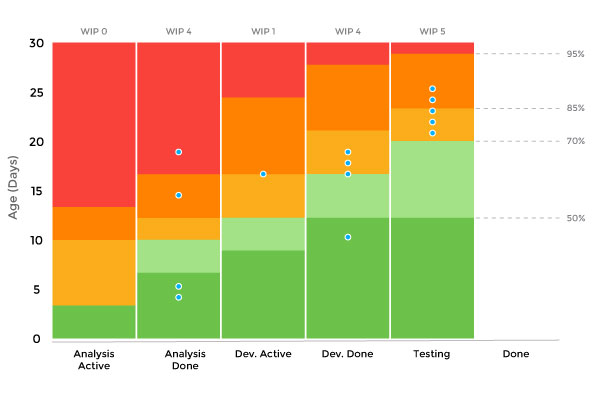


Fig:- [*Work Item Age*](https://pt.slideshare.net/rodrigosilvap/scrum-com-kanban-construindo-pontes-e-no-paredes)

**10. Throughput**

Throughput measures average tasks processed in each time unit. You can also call it a measure for story points per iteration. It represents a team’s productivity level. Throughput helps you understand the effect of workflow on business performance. You can get a better overview of the capacity of your team. However, it doesn’t show the starting point of tasks.

**11. Blocked Time**

This metric assigns a blocker sticker to a task. It means that due to some reason, the assignee can’t proceed with a particular task because of some dependency. As soon as the dependency is fulfilled, you should move the blocked card to the right on the task board. Count the number and duration of blocked cards for measuring the number of blockers. Resolving the blockers will allow you to finish your “in progress” task quickly.

**12. Escaped Defects**

When there are bugs in production, it causes a lot of unexpected damage. They pose problems, and the team needs to address them. Escaped defects metrics help in bug identification when a release enters production. You can assess the quality of the software in a raw form.

**13. Failed Deployments**

Failed deployments are a useful quality metric. It helps in assessing the number of overall deployments. Moreover, teams can determine the reliability of the testing and production environment. This metric also determines whether a sprint is ready to enter production.

**14. Code Coverage**

Code coverage measures the percentage of code unit tests cover. You can run this metric with every build. It represents the percentage of code coverage in raw form. This metric gives a decent perspective on progress. But it doesn’t cover other kinds of testing. Thus, high code coverage numbers don’t necessarily represent high quality.

**15. Quality Intelligence**

The quality intelligence metric is a must if you’re looking for clarity on software quality. It helps in identifying recent code changes. Suppose there are new codes that the team has developed but testing is yet to be done. Maybe there are instances where the quality declines in those codes. Quality intelligence helps determine the same. It makes the team aware of when they should invest more time in testing.

Agile delivers value to both employees and customers. The metrics that we discussed help to optimize the development process in an agile framework. Agile metrics are development centric. But to optimize the production processes, lean is the way to go. Both agile and lean empower teams to deliver better value. They also focus on quick delivery of value to customers. Let’s discuss lean metrics. We’ll get to know what they are and discuss some lean metrics that are musts to include in your development cycle.

* **4.6 Introduction to User Stories :-**

In [software development](https://en.wikipedia.org/wiki/Software_development) and [product management](https://en.wikipedia.org/wiki/Product_management), a **user story** is an informal, natural language description of features of a software system. They are written from the perspective of an [end user](https://en.wikipedia.org/wiki/User_(computing)#End-user) or [user of a system](https://en.wikipedia.org/wiki/User_(system)) in project management software Depending on the project, user stories may be written by different stakeholders like client, user, manager, or development team.

**User stories may follow one of several formats or templates:-**

The most common is the *Connextra template*, stated below. [Mike Cohn](https://en.wikipedia.org/wiki/Mike_Cohn) suggested the "so that" clause is optional although still often helpful.

As a <role> I can <capability>, so that <receive benefit>

Chris Matts suggested that "hunting the value" was the first step in successfully delivering software, and proposed this alternative:

In order to <receive benefit> as a <role>, I can <goal/desire>

Another template based on the [Five Ws](https://en.wikipedia.org/wiki/Five_Ws) specifies:

As <who> <when> <where>, I <want> because <why>

**Examples:-**

**# Screening Quiz**

As the HR manager, I want to create a screening quiz so that I can understand whether I want to send possible recruits to the functional manager.

**# Quiz Recall**

As a manager, I want to browse my existing quizzes so I can recall what I have in place and figure out if I can just reuse or update an existing quiz for the position I need now.

**# Limited Backup**

As a user, I can indicate folders not to backup so that my backup drive isn't filled up with things I don't need saved.

 A user story is an informal, general explanation of a software feature written from the perspective of the end user. Its purpose is to articulate how a software feature will provide value to the customer.

A key component of agile software development is putting people first, and a user story puts end users at the center of the conversation. These stories use non-technical language to provide context for the development team and their efforts. After reading a user story, the team knows why they are building, what they're building, and what value it creates.

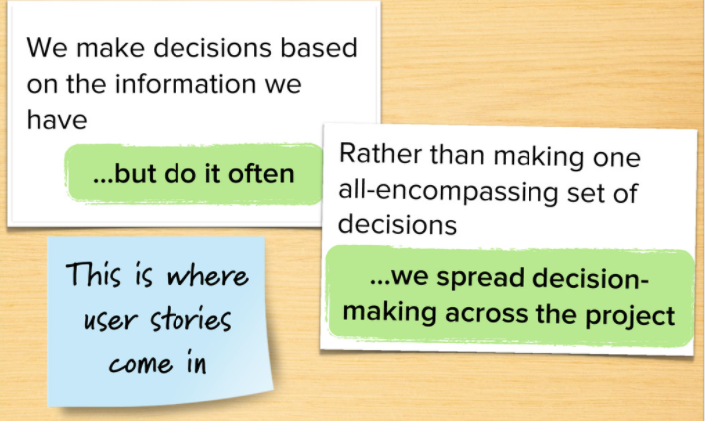
User stories are one of the core components of an agile program. They help provide a user-focused framework for daily work — which drives collaboration, creativity, and a better product overall.

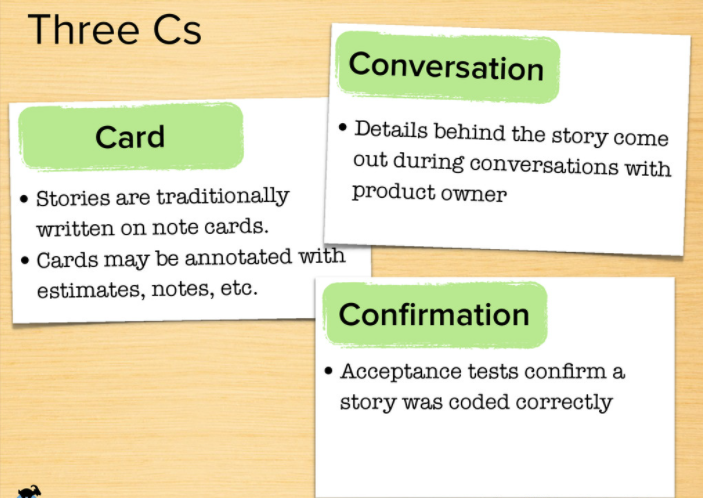
**User stories** are short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system. They typically follow a simple template:

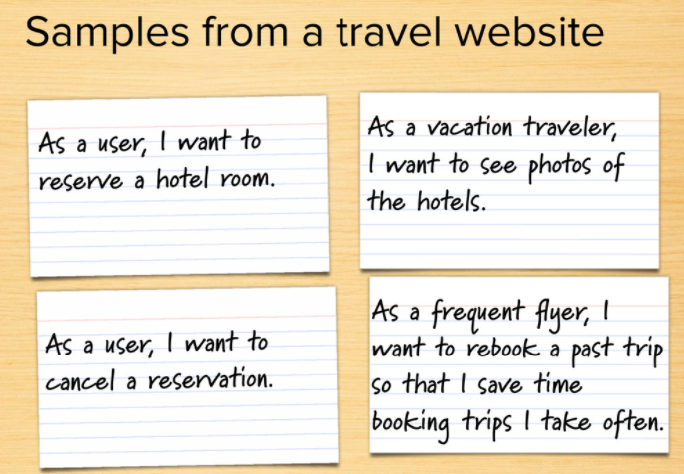
As a < type of user >, I want < some goal > so that < some reason >.

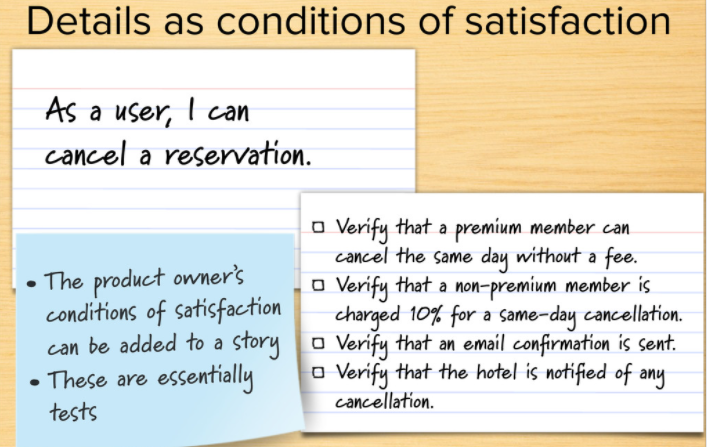
User stories are often written on index cards or sticky notes, stored in a shoe box, and arranged on walls or tables to facilitate planning and discussion. As such, they strongly shift the focus from writing about features to discussing them. In fact, these discussions are more important than whatever text is written.

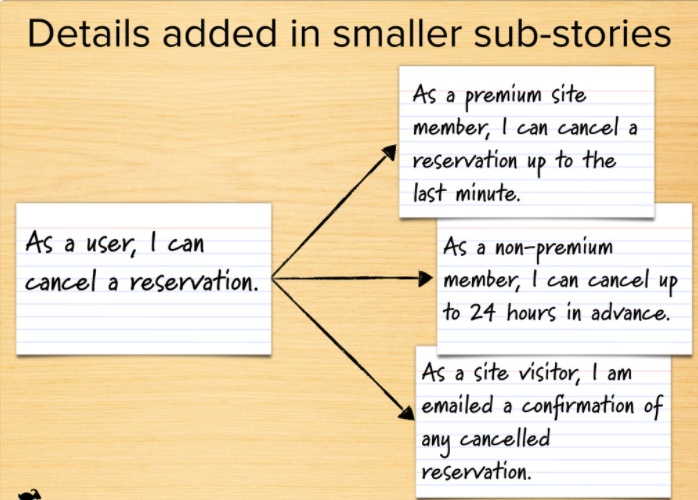
* As a customer, I want to be able to run your product on all versions of Windows from Windows 95 on.
* As the CTO, I want the system to use our existing orders database rather than create a new one, so that we don't have one more database to maintain.
* As a user, I want the site to be available 99.999 percent of the time I try to access it, so that I don't get frustrated and find another site to use.
* As someone who speaks a Latin-based language, I might want to run your software someday.
* As a user, I want the driving directions to be the best 90 percent of the time, and reasonable 99 percent of the time.

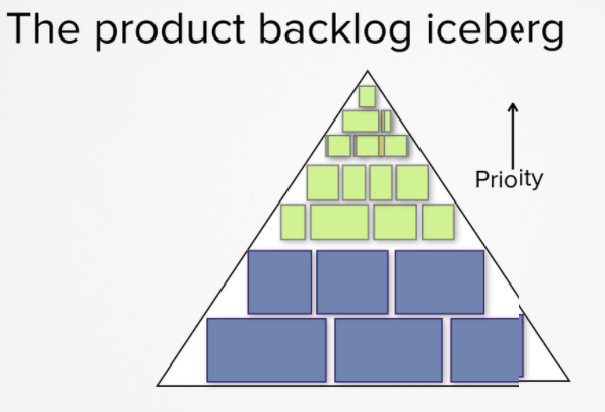


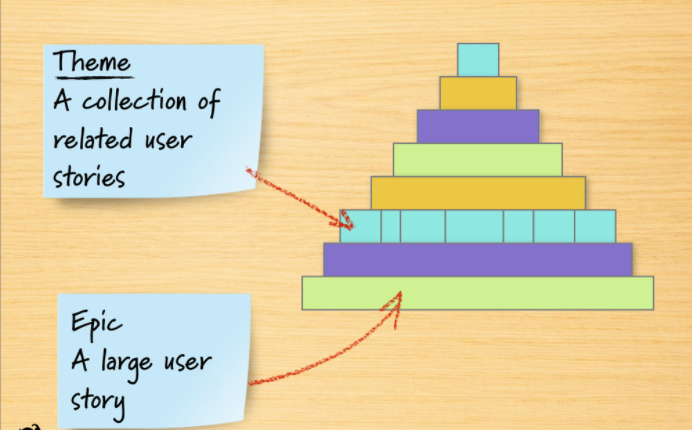


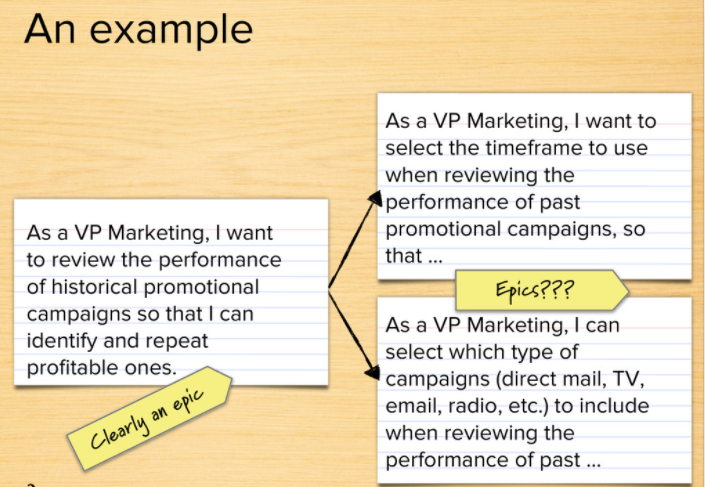












* **4.7 Estimate the Product Backlog :-**

**A product backlog** is a list of items to be done. Items are ranked with feature descriptions. In an ideal scenario, items should be broken down into user stories.

The agile product backlog in [Scrum](https://www.javatpoint.com/agile-scrum) is a list of prioritized features. It contains a short description of all the functionalities desired in the product. In usual scenario, items should be broken down into user stories. Commonly, a Scrum team and its product owner write everything that they can think for agile backlog prioritization.

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

**Why Product Backlog is Important?**

* It is prepared so that estimates can be given to each and every feature.
* It helps in planning the roadmap for the product.
* It helps in re-ranking the features so that more value can be added to the product.
* It helps in determining what to prioritize first. Team ranks the item and then builds value.

**Characteristics of Product Backlog**

* Each product should have one product backlog which can have a set of large to very large features.
* Multiple teams can work on a single product backlog.
* Ranking of features is done based on business value, technical value, risk management or strategic fitness.
* Highest ranking items are decomposed into smaller stories during release planning so that they can be completed in future iterations.

**The Product Backlog comprises the following different types of items:**

* Features
* Bugs
* Technical work
* Knowledge acquisition

There are **three main reasons to estimate a product backlog**. **First**, it allows a team and its product owner to make longer term predictions about how much can be delivered by when. It allows teams to answers questions like:

* How much can you deliver in three months?
* When can a certain set of product backlog items be delivered?

**Second**, it aides product owners in making prioritization decisions. Priorities should be set based on the expected benefits and costs of the product backlog items.

In prioritizing work, we consider the cost of developing product backlog items. For most items, the estimate of the effort involved is the biggest component of the cost.

A **third reason** to estimate items on the product backlog is that team members become more knowledgeable about the item by thinking about it enough to estimate it. This means there will be fewer surprises when the feature is being developed.

Estimate using five scales — very low, low, medium, high, and very high.

Very low means approx one day, low means 1–3 days, medium means 3–5 days, high means 4–8 days, and very high means 7–12 days.

Relative estimates are about sizing problems based on some factors like the bigness of the problem, requirement clarity and technology complexity, etc. Since the team prefers writing PBI ([Product Backlog](https://www.agilemania.com/blog/what-is-a-product-backlog/) Item) in User Story, so story point becomes the perfect choice.

If there is already an estimated PBI, we estimate the rest compared to that, but what if we don’t have any existing estimated PBI? We have to pick one PBI with moderate complexity (for example — everyone agreed that a PBI is an intermediate complex because there are 3–4 validations, 2–3 workflows, the requirement is well understood, and technology is not new).

Since we have one now, 2nd PBI can be estimated comparing with 1st PBIs. Similarly, 3rd PBIs can be estimated comparing with 1st and 2nd PBIs. We can use the Fibonacci series to estimate and can define scale: -

**1 — Very Small, 2 — Small, 3 — Medium, 5 — Large, 8 — Very Large, 13 — Very Very Large**

PERT estimation: it is also known as the 3-points estimation technique. Here are detailed steps for the same.

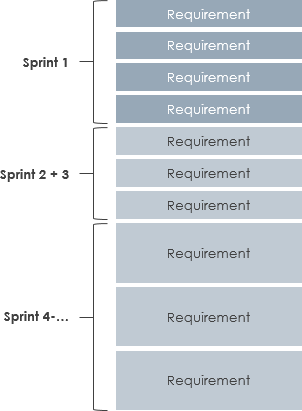
1. The team must estimate all PBI using the relative estimate.
2. Let the team decide how much work the team can commit for an iteration and choose three numbers. The highest number, called Pessimistic, the lowest is Optimistic, and the common one is called Most Likely.
3. Every team member has to give an estimate. Assume team members have decided 20, 35, 30, 25, 30, 30, 20, 25 points for 1st sprint. Then Pessimistic Number (35), Optimistic Number (20), and Most Likely (30). Now use formula *Pessimistic+Optimistic+4(Most likely)/6* = 29.16, so 29 points. It is the estimated velocity.
4. Since we know the estimated team velocity and already has an estimated product backlog, we can divide the total story point by velocity to get the number of iterations required to complete all PBIs.
5. Since Iteration duration is fixed, multiply duration with estimated iterations will give the expected duration of development.
6. Team size remains the same, so we know the estimated cost of one iteration to multiply cost with the number of expected iterations to know the expected cost of development.

The **Product Backlog** is a sorted list of all the products you need and the only source of product demand changes. The [**product owner**](https://www.visual-paradigm.com/scrum/what-is-project-owner-role-in-scrum/) is responsible for the content, availability, and priority of the product to-do list called Product Backlog.

The Product Backlog is a continuously improved list, with the initial version listing only the most preliminary and well-known requirements (no necessary well understood). Product Backlog evolved based on changes in the product and development environment. The Backlog is dynamic and it often changes to identify what is necessary to make the product reasonable, competitive, and useful. The Product Backlog exists as long as the product exists.

The Product Backlog lists all the features, use cases, user stories, improvements, and bug fixes that are made to future releases. Product to-do list entries contain descriptions, sequences, and estimated characteristics.

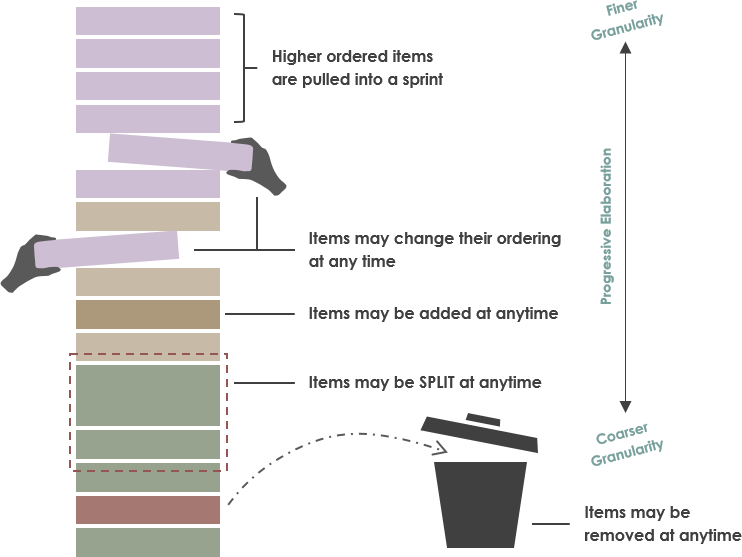
**Product Backlog Items** (PBIs) are usually sorted by value, risk, priority, and necessity. It is a sequence of highest to lowest priority, with each entry having a unique order. Product to-do list entries at the top need to be developed immediately. The higher the ranking, the more urgent the product to-do list entry is, the more you need to think carefully and the more consistent your opinion on the value.



there can only be one product backlog that describes the next product development work. Then you need to use the attributes that classify the product backlog items.

Add details, estimates, and sorts by grooming the product backlog. This is an ongoing process where the product owner and the development team collaborate to discuss the details of the product representative list entry. Entries are reviewed and modified as they are sorted out in the product backlog list. However, the product owner can update the items of the product Backlog at any time or make decisions as appropriate.

**Product Backlog Grooming** is an on-going activity in Sprint rather than a timebox event, together with product owners and development teams. Often, development teams have domain knowledge that they can refine themselves. However, when and how to complete the optimization is the decision of the [**Scrum**](https://www.visual-paradigm.com/scrum/scrum-in-3-minutes/) team. [**Product Backlog Refinement**](https://www.visual-paradigm.com/scrum/what-is-product-backlog-refinement/) usually takes no more than 10% of the development team’s time.



Product Backlog Refinement Meeting

The development team is responsible for all the estimation work. Product owners can influence their decisions by assisting the team in weighing trade-offs. However, the final estimate is determined by the person performing the work.

Product Backlog is simply a list of all backlog items (PBIs) that needs to be done within the project. It replaces the traditional requirements specification [**artifacts**](https://www.visual-paradigm.com/scrum/what-are-scrum-artifacts/). PBIs reflect the needs of customers or stakeholders. A common way to incorporate the end users’ needs is to write the **PBI in the form of a User Story**.

The [**Product Owner (PO)**](https://www.visual-paradigm.com/scrum/what-is-project-owner-role-in-scrum/) “owns” the product backlog on behalf of the stakeholders, and is primarily responsible for creating it*. It is not necessary for a PO to create the backlog personally* – he or she may instruct the development team and/or the [Scrum Master](https://www.visual-paradigm.com/scrum/what-is-scrum-master/) to help him/her in defining the backlog items and in estimating them. The PO is held responsible for the creation and upkeep of the product backlog. Therefore, the PO also oversees the [backlog refinement](https://www.visual-paradigm.com/scrum/what-is-product-backlog-refinement/) process.

The product backlog corresponds to your project plan, the roadmap for what the team plans to deliver. After the team define it, the team have a prioritized list of features and requirements to build. The product backlog also provides a repository of all the information the team need to track and share among themselves.

PBIs reflect the needs of customers or stakeholders. A common way to incorporate the end users’ needs is to write the *PBI in the form of a User Story*. However, PBIs cans range from use cases, epics, **User Stories**, or even bugs, or timeboxed research tasks that reside on the product backlog. In fact, not all items in a product backlog will be at the same level of detail at the same time as shown in the Figure below:

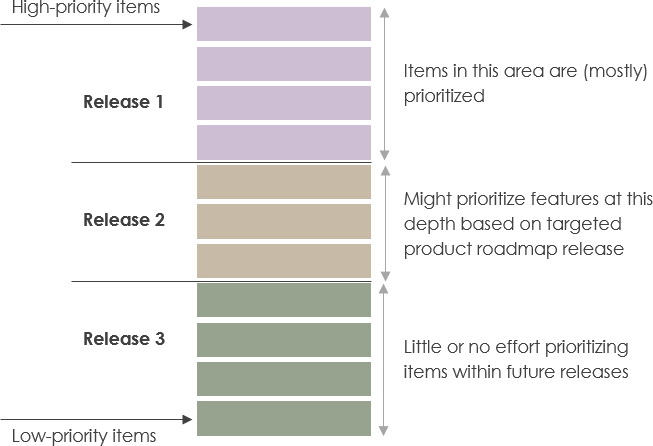


Fig:- Detailed product backlog

PBIs that we plan to work on soon should be near the top of the backlog, small in size, and very detailed so that they can be worked on in a near-term [sprint](https://www.visual-paradigm.com/scrum/what-is-sprint-in-scrum/). PBIs that we won’t work on for some time should be toward the bottom of the backlog, larger in size, and less detailed. That’s OK; we don’t plan to work on those PBIs anytime soon.

As we get closer to working on a larger PBI, such as an epic, we will break that story down into a collection of smaller, sprint-ready stories. This should happen in a just-in-time fashion. If we refine too early, we might spend a good deal of time figuring out the details, only to end up never implementing the story. If we wait too long, we will impede the flow of PBIs into the sprint and slow the team down. We need to find the proper balance of just enough and just in time.

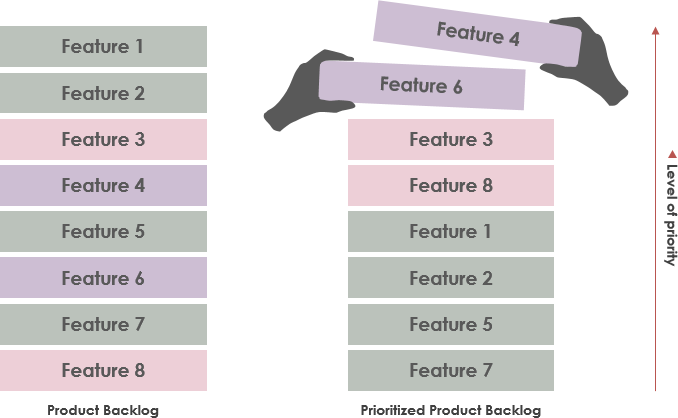


Fig:- Agile Prioritized Product Backlog

**Who is responsible for refining the PBIs?**

The Product Owner (PO) “owns” the product backlog on behalf of the stakeholders, and is primarily responsible for creating it. It is not necessary for a PO to create the backlog personally – he or she may instruct the development team and/or the Scrum Master to help him/her in defining the backlog items and in estimating them. The PO is held responsible for the creation and upkeep of the product backlog. Therefore, the PO also oversees the backlog refinement process.

So, to answer to question, the product owner who owns the product backlog, but it is not necessary for a product to create every backlog items. Typically the product owner might create large PBIs according to the high level requirement or user goals, the will then help the product owner to break down the large items into user stories as it move to the top of the backlog as some “sprintable” user stories.

* **4.8 Technique for Estimating Story Points :-**

**“**Estimation (or estimating) is the process of finding an estimate, or approximation, which is a value that is usable for some purpose even if input data may be incomplete, uncertain, or unstable. The value is nonetheless usable because it is derived from the best information available**”**

Estimating isn’t about estimating at all. Estimating is about creating a shared understanding of the requirements, and a shared understanding of the solution. When teams have problems estimating, it’s almost never an estimating problem, it’s a shared understanding problem.

**We do Estimate because**

* To get people liable for their work
* To predict the finish line
* To fill up the sprint with work
* To measure teams’ progress we do estimation

Story point estimation includes three main components:

* **Risk**: The risk of a particular project or item includes vague demands, dependence on a third party, or changes mid-task.
* **Complexity**: This component is determined by how difficult the feature is to develop.
* **Repetition:** This component is determined by how familiar the team member is with the feature and how monotonous certain tasks are within development.

### Some Estimation Technique :-

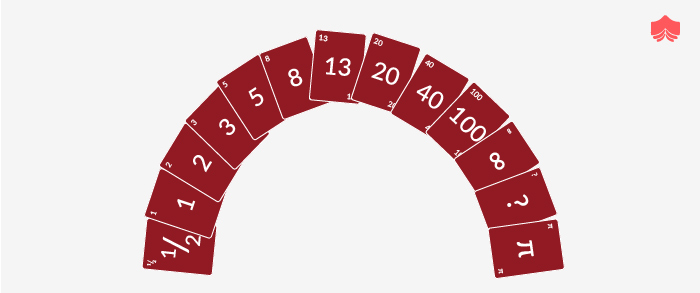
### Planning Poker

All participants use numbered playing cards and estimate the items. Voting is done anonymous and discussion is raised when there are large differences. Voting is repeated till the whole team reached consensus about the accurate estimation. Planning poker works well when you have to estimate a relative small number of items (max 10) in a small team (5-8 people). Tip: try to keep the voting between affordable numbers.  Maximize the highest card to 13 points.

Planning Poker is an Agile estimating and planning technique that is based on an agreement from the team on the points being assigned to the PBI. It makes sure that everyone participates and that every individual in the team shares his/her opinion.

To start with, each team member is given a set of cards with numbers on them. The numbers are usually in the Fibonacci sequence: 0, 1, 2, 3, 5, 8, 13, and 21. The [product owner](https://www.knowledgehut.com/blog/agile/career-path-of-a-scrum-product-owner) reads out the story, after which, everybody in the team is asked to hold up the card showing the level of effort that they believe this story holds. Once all the votes are in, the team members with the lowest and highest estimates explain why they choose their numbers. The team then re-estimates as per the new perceptions discussed. Once the agreement has been reached that score is recorded with the story to which it relates, the team is good to proceed.

Planning poker is an agile estimation technique that makes use of story points to estimate the difficulty of the task at hand. Based on the Fibonacci sequence, the story point values that can be assigned are 0, 1, 2, 3, 5, 8, 13, 21. Each of these represent a different level of complexity for the overall project.



[Planning poker](https://reqtest.com/general/the-reasons-behind-why-planning-poker-works/) starts with the team members involved in the estimation process sitting together for the session. Each member holds cards with the story point values described above. The next step is for either a leader figure or the customer to read out the ‘user story’ (which is essentially the project), and describe all the requirements and features.

The stakeholder reading out the story will engage in discussion with the team members who are estimating, who will, in turn, discuss with one another. In this phase, they can ask the customer or owner questions for clarification and express any reservations they have.

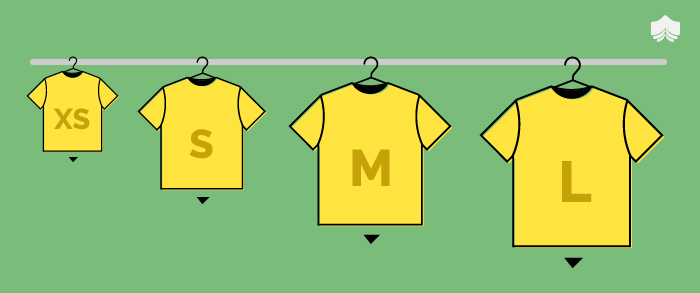
When the discussions are finished, all of the estimators will select a card with the story point they believe needs assigning to the project. If the story point estimations match up – then that will be the final estimate. However, if they do not match up, then estimators who gave the lowest and highest points can voice their reasoning, and more discussion will ensue until there is a consensus.

This technique is not good for large teams, or when there are a large number of items that need estimating. If you only have a selected number of items (between 2 and 10) and a small band of teammates, then this is a good technique to use. It’s also one of the most popular estimation techniques.

### 2. T-Shirt Sizes

If you think about T-shirts, there are multiple sizes to choose from. More specifically – there is extra-small (XS), small (S), medium (M), large (L) and extra-large (XL). This technique uses these sizes as story points for the size of the project, and it is a useful way of thinking when estimation needs to occur.

This is a useful method for being time-efficient. It can give a quick and rough estimate for how much work is expected for a project. The sizes can be converted into numbers at a later stage – when the team assigns a relative size to the project on hand. This is decided through discussion and collaborative efforts to understand everything that needs to be done.



If estimators propose sizes that do not match up, then the team voices their opinions on the topic and must eventually reach a consensus.

This is a pretty informal method that is great to use for a large number of items. Unfortunately, story points can be tricky. What might seem one size to one person could be perceived as another by someone else. Whilst this can create some confusion, this method is based on open discussion, and everyone should get the chance to have their say.

This is a seamless technique for estimating a huge backlog of relative large items. T-shirt sizing is based on the concept of binning- a technique for accurately grouping together items of similar size. The bins are typically allocated labels matching to those commonly used with T-shirt sizes: extra small, small, medium, large, extra-large, etc.

The primary advantage to t-shirt sizes is the ease of getting started. T-shirt sizes can be a great way of becoming familiar to relative estimating. So, you can start with it if your team finds that easier.

When quoting T-Shirt sizes, our colleagues will still need to know tangible cost and time this roughly translates to so estimates can be used in decision making. To do this, the following guide can be used for quoting and explaining the cost and time.



It should be understood that we are putting the two variables of time and cost into one high level measurement. There will of course be combinations that don't fit. e.g. 2 months 10 people.

**3.Dot Voting**

When there is a relatively small set of items and you don’t want any complex techniques, you can opt for Dot Voting. This method has been coined from decision making and can be used for estimating. Each person gets a small number of stickers and can choose to vote for the individual items. More the dots, bigger is the size. This method is very simple and fast, it will work effectively to assess a small number of stories (up to 8-10). This method has originated form decision making and you can use it for estimating. Each person gets a small number of small stickers and can choose to vote for the individual items.

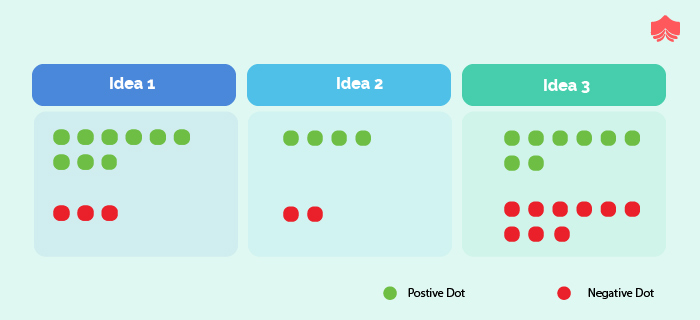
Sometimes it can be hard to order the items in the [product backlog](https://reqtest.com/agile-blog/how-to-build-your-product-backlog/). This ranking method enables you to sort these items from highest to lowest priority, so you know where to focus your efforts. To do this, you need to select the most important user stories.

Start by posting all of the stories you need to deal with on a wall somewhere (or a board, if you’re feeling more conventional). The posts should contain the story description and should look unique so that they are easily distinguishable to voters.

Team members involved in the process are all given 4 to 5 dots to dole out. The use of stickers or markers can be effective for this process. Team members place these dots on the user stories that they would prefer to start working on and distribute them throughout the options.

A leader then orders the stories from most preferred to least preferred (most preferred would be the story with the most marks or dots on it). If there is any stakeholder who is not happy with the order, then the items are divided into three groups – high, middle and low priority. Team members vote on the high priority stories again until they reach a consensus.

This is not necessarily an agile estimation technique – it’s more of a decision-making tool. But if you have a small number of items – it can be very efficient. It’s also really simple and is a good visualization tool.



### 4. The Bucket System

This method relies on placing different values on a table. We call the placements ‘buckets’, but you can just use cards. The values are generally 0,1,2,3,4,5,8,13,20,30,50,100 and 200 – although these can be expanded if necessary.

Estimators need to take stories and collectively choose which buckets each item falls into. To do this, place cards with the items written on them into the buckets. Before placing each item, it is important to have discussed the features and requirements of each with the estimator team. All items must be assigned and placed in the buckets upon consensus.

The buckets can also be changed and rearranged if the group feels it necessary to reassign an item. There is a ‘divide and conquer’ phase after assigning important items. Estimators get the remaining items and can place them in buckets that they believe the items should sit in – without consensus.

If a participant does not understand a story, it can be transferred to someone who does. If someone does not agree with a certain item placed in a specific bucket, further discussions takes place to agree on where to place the item and why. This ‘sanity check’ is critical to this process.

This method is more time-efficient and reasonable than Planning Poker. It is a great agile estimation technique to use if you have a large number of items and a big team.

Much quicker than planning poker is the Bucket System. This system is a decent substitute when estimating a large number of items with a large group of participants quickly. Different buckets are created with values: 0,1,2,3,4,5,8,13,20,30,50,100, 200. The stories need to be placed within these where the estimator finds them suitable. The group estimates the items by placing them in these “buckets”. Buckets are usually different sheets of brown paper where you can place the sticky note with the item. But you can also use actual baskets to limit discussion about already processed items.

### 5. Affinity Mapping

Firstly, silent relative sizing needs to occur. To prepare for this, place two cards on opposite sides of a wall. One should say ‘small’ and the other should say ‘large’. The leader (or product owner) needs to provide each estimator with a subset of items and should remain present during the process to clarify anything. Estimators then place the items on the wall, relative to each item’s perceived size. The size depends on the effort expected to complete them. There is no discussion at this point.

Team members can then change the location of wall items, discussing as they go. Once teams finish editing the wall, they can finalize product backlog items in their positions. However, just before this, the product owner may step in if they spot a discrepancy between what the team members have estimated in comparison to their ideas.

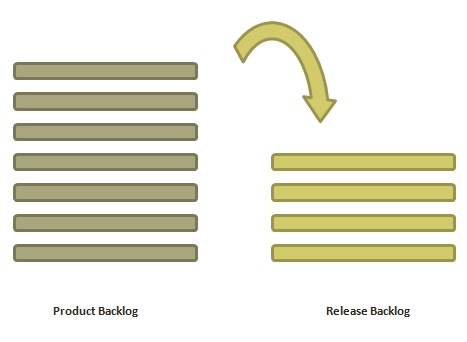
Using a project backlog management tool can help ensure that the finalized estimations are saved. This is a good technique to use for a smaller team. Also, if you have a large number of backlog items, this is a bad choice. You might find it to be time-consuming with too many items.

**6.Ordering method**

This is an exercise where you get an accurate image on the relative size of items. This works best in a small group of expert. All items are placed in random order on a scale label ranging from low to high. Every participant is being asked to move one item on the scale. Each move is just one spot lower or one spot higher or pass the turn. This continues till no team member want to move items and passes their turn. The ordering protocol is a method of getting fine grained size estimates. Works best with a relative small group of people and a large number of items.

**4.9 Plan Product Releases :-**

The purpose of release planning is to create a plan to deliver an increment to the product. It is done after every 2 to 3 months.

****

**Who is Involved?**

* **Scrum Master −** The scrum master acts as a facilitator for the agile delivery team.
* **Product Owner −** The product owner represents the general view of the product backlog.
* **Agile Team −** Agile delivery team provides insights on the technical feasibilities or any dependencies.
* **Stakeholders −** Stakeholders like customers, program managers, subject matter experts act as advisers as decisions are made around the release planning.

**Prerequisites of Planning:-**

**The prerequisites of release planning are as follows −**

* A ranked product backlog, managed by the Product Owner. Generally five to ten features are taken which the product owner feels that can be included in a release
* Team's input about capabilities, known velocity or about any technical challenge
* High-level vision
* Market and Business objective
* Acknowledgement whether new product backlog items are needed

**Materials Required**

**The list of materials required for release planning is as follows −**

* Posted agenda, purpose
* Flip charts, whiteboards, markers
* Projector, way to share computers having data/tools required during planning meeting
* Planning data

**Planning Data**

**The list of data required to do release planning is as follows −**

* Previous iterations or release planning results
* Feedback from various stakeholders on product, market conditions, and deadlines
* Action plans of previous releases / iterations
* Features or defects to be considered
* Velocity from previous releases/ estimates.
* Organizational and personal calendars
* Inputs from other teams and subject matter experts to manage any dependencies

**Output**

**The output of a release planning can be the following −**

* Release plan
* Commitment
* Issues, concerns, dependencies, and assumptions which are to be monitored
* Suggestions to improve future release plannings

**Agenda**

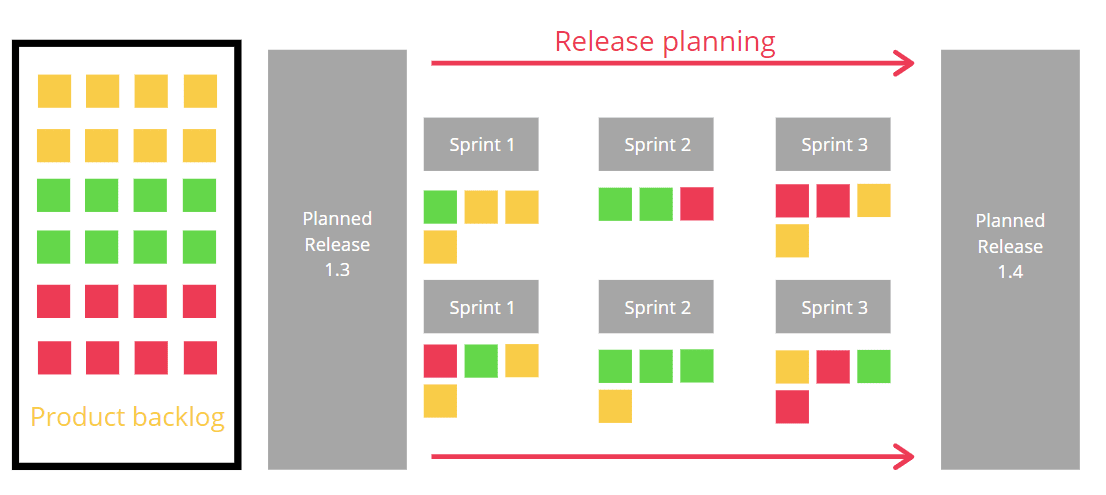
**The agenda of a release planning can be −**

* **Opening ceremony −** Welcome message, review purpose and agenda, organizing tools and introduction to business sponsors.
* **Product Vision, Roadmap −** Show the large picture of the product.
* **Review previous releases −** Discussion on any item which can impact the plan.
* **Release name / theme −** Inspect the current status of roadmap themes and do the required adjustments, if any.
* **Velocity −** Present the velocity for the current release and of previous releases.
* **Release schedule −** Review key milestones and decision on time boxes for release and iterations within release.
* **Issues and concerns −** Check any concerns or issue and record them.
* **Review and Update the Definition of Done −** Review the definition of done and make appropriate changes based on technology, skill, or changes in team members since the last iteration / release.
* **Stories and items to be considered −** Present the user stories and features from the product backlog to be considered for scheduling in the current release.
* **Determine sizing values −** If the velocity is unknown, then plan the sizing values to be used in the release planning.
* **Coarse the size of stories −** The delivery team determines the appropriate size of the stories under consideration and splits the stories into multiple iterations if a story is too large. The product owner and the subject matter experts clarify the doubts, elaborate the acceptance criteria, and make proper story splits. The scrum master facilitates the collaboration.
* **Map stories to iterations −** The delivery team and the product owner move the stories/defects in the iterations based on the size and velocity. The scrum master facilitates the collaboration.
* **New concerns or issues −** Check any new issues based on previous experience and record the same.
* **Dependencies and assumptions −** Check any dependencies/assumptions planned during the release planning.
* **Commit −** The scrum master calls for the planning. Delivery team and Product owner signal it as the best plan and then commit to move to the next level of planning, that is, iteration planning.
* **Communication and logistics planning −** Review/Update the communication and logistics planning for the release.
* **Parking lot −** Process parking lot means all items should be either resolved or set as action items.
* **Distribute Action items and action plans −** Distribute the action items among their owners, process the action plan.
* **Retrospect −** Solicit feedback from participants to make the meeting successful.
* **Close −** Celebrate the success.

Agile release planning is a product management method where you plan incremental releases of a product. It differs from traditional software planning where you focus on major releases.

In Agile release planning, you prepare for staged releases and then break those down into several different sprints or iterations.

Depending on your team structure and the size of the project, you may even have several sprints running at the same time.



A sprint ends with a new product increment, but that may not mean a product release happens.

Think about writing a book — each sprint should end with a new draft. But, just because you have a finished draft for your client (editor, publisher, etc.) to review doesn’t mean you release it to the public.

Release planning helps you plan which product increments (versions) get released to the market and when.

**What is the purpose of Agile release planning?**

The purpose of release planning within the [Agile methodology](https://monday.com/blog/project-management/agile-project-management/) is to ensure the product is always moving in the right direction and that logical releases are frequently happening.

A release plan outlines immediate future releases but doesn’t try to plan for years to come.

It goes into more detail than a product roadmap (high-level scope and timeline).

But an Agile release plan doesn’t outline the work in each release. Instead, it batches iterations or sprints together into releases.

Old-fashioned executives fear going Agile means each version is a random collection of features. A release plan ensures that you create a coherent version of your product every time.

* **4.10 Product Prioritization:-**

[Prioritization](https://university.hygger.io/en/articles/3272399-lesson-2-prioritization) means organizing things when the most important thing is done or dealt with first.

This may involve organizing a group of tasks or things that need to be completed, as well as their ranking according to different factors (for example, criticalness). Prioritization helps us to clearly define what should be focused on to accomplish more and achieve maximum productivity.

There are two common ways of defining prioritization:

* what should be done first when you have a lot of tasks to complete.
* when you prioritize throughout the day to make sure you have time for everything.

The process of prioritizing allows you to focus on tasks that are important and urgent so that you can later direct your attention to lower priority tasks.

If you ignore prioritization, you will have trouble getting things done on time and stress about how you will finish everything on your to-do list.

Everyone has many things that need to be done. Often, people track their stuff by creating to-do lists. To-do lists can be really helpful to see a birds-eye view of your needs and figure out what you need to focus on right now, work efficiently, and save time and energy. This is probably one of the most simple ways to deal with priorities.

Prioritization in literary terms means the decision of arranging things in order of their importance. Prioritization in agile is the act of deciding in what order the agile team will work on the requirements in a project. Understanding prioritization is essential for all projects, but it becomes specifically critical in agile as an agile project is time-boxed with a fixed set of resources which requires prioritization in order to accommodate the time and budget constraints. Further prioritization process helps the agile team to consider the bare minimum features necessary to create customer value. In order to process agile prioritization, it is essential to understand the factors that a product owner needs to consider before determining the priorities.

**Agile Prioritization Factors:-**

* The financial value of the requirements is a major factor to be considered in prioritizing requirements. The value could be expressed as new revenue, incremental revenue, or as operational efficiency.
* The cost of developing the requirements is another essential factor to be considered by the product owner. Value and cost together indicate the RoI for the requirements.
* The next factor to be considered in prioritization is the amount and significance of knowledge and capabilities that the team will gain while working on the requirements.
* Understanding the level of risks involved in introducing the new features is essential in the process of prioritization.
* **MoSCoW Agile Prioritization Technique :-**

MoSCoW Prioritization in Agile: In the [DSDM methodology](https://en.wikipedia.org/wiki/Dynamic_systems_development_method), the priorities are expressed as per the MoSCoW model:

* **M**ust– The must requirements is given the topmost priority

If you would have to cancel your release if you couldn’t include it, then it’s a Must Have. Must-Have user stories are those that you guarantee to deliver because you can’t deliver without, or it would be illegal or unsafe without. If there’s any way to deliver without it — a workaround, for example — then it should be downgraded to Should Have or Could Have. That doesn’t mean it won’t be delivered. It means that that delivery is not guaranteed.

* **S**hould– Next priority is given to the requirements that are highly desirable, though not mandatory

Should Have features are important, but not absolutely vital to the success of your release. They can be painful to leave out, and may have an impact on your product, but they don’t affect minimum viability of your product.

* **C**ould– The next priority is given to the requirement that is nice to have

Could Have items are those that are wanted or desirable but are less important than a Should Have item. Leaving them out will cause less pain than a Should Have item.

* **W**on't– And the final consideration is given to the requirements which will not work in the process at that point of time.

Won’t Have user stories are those in which everyone has agreed not to deliver *this time around.*You may keep it in the backlog for later, if or when it becomes necessary to do so.

* **Kano Model of Prioritization in Agile :-**

Professor [Noriaki Kano](https://en.wikipedia.org/wiki/Noriaki_Kano) propagated Kano Model of Prioritization. This prioritization technique involves three levels that include considering customer satisfaction from disappointment to not happy to immediate happiness to get delighted. Two important factors that create an impact on the satisfaction level during this prioritization are the existence of features and the degree of implementation. The level of satisfaction is achieved along with full implementation. Some features lead to a basic level of satisfaction while others create more – the higher the implementation, the greater the level of satisfaction.

The Kano model was developed in the 1980s by Professor Noriaki Kano. Under the Kano Model, features are categorized according to needs and expectations of customers. There are a variety of versions of the Kano model. The original, however, classifies items using five thresholds: Must-be, Attractive, One-Dimensional, Indifferent, and Reverse.

**Must-Be** — These are expected by your customers. They are features that will not WOW them. They *must be* included in your product, and are often taken for granted.

**Attractive** — These make users happy when they’re there, but don’t disappoint them when they’re not.

**One-Dimensional** — These are features that make users happy when they’re there, unhappy when they’re not.

**Indifferent** — These have no impact on customer satisfaction levels. For example, refactoring parts of your code so that it is easier to read and understand. There is no direct value to the customer, but it will make it easier for you to maintain in the future.

**Reverse** — These make users unhappy when they’re there, happy when they’re not. For example, you might implement high-security features requiring an extra step to login. However, if customers do not value enhanced security, they will become dissatisfied with the extra step.

* **Relative Weighting Prioritization Technique :-**

The relative weighting scheme is a simple model where prioritization is done based upon all the factors mentioned above. The major factors considered in relative weighing prioritization technique are:

* The value of a feature and the negative impact that might be caused by the absence of the feature
* Based on the expert judgment made by the product owner and supported by the agile team in ranking the score of features in the following way (a scoreboard from 1 to 9 is usually used)
  + Benefit from having the feature
  + Penalty for not having the feature
  + Cost of producing the feature
  + The risk incurred in producing the feature
* The priority and rank are then determined by dividing the value score as below:
  + (Benefit score + Penalty score) / (Cost score + Risk score)

In relative weighting prioritization, if the results come out in numerical value, it becomes easier for the [product owner](https://www.simplilearn.com/scrum-master-or-product-owner-what-suits-you-better-article) to arrive at a faster prioritizing decision. Using all these three techniques, a product owner performs the prioritization exercise towards achieving customer satisfaction and customer value. The whole process of prioritization in agile is followed in order to create customer value, which is possible with innovation, focused execution, and lean delivery.

* **Stack Ranking Technique :-**

When you stack rank, you consider each backlog item and place it in order of priority. You start with one, then two, then three, and continue to *n*, the total number of items in your backlog.

There are twothings about stack ranking:

* There can only be one number one. So, you will avoid a common (and, frankly, stressful) product pitfall where everything becomes very high priority.
* It is often more accurate, less confusing. You prioritize each item *relative* to all other items, which makes it simplifies the process and makes it clearer. This is often a more accurate and easier way to prioritize than to provide absolute values (such as “very high priority”).
* **Cost of Delay Technique :-**

To quantify the cost of delay is to answer the question: “What will be the cost per time unit if we delayed delivery?” You can then compare your answer with the estimate to deliver the feature with the highest cost of delay, and is cheapest to do, first. This is called Weighted Shortest Job First (WSJF) prioritization.

Keep in mind that cost of delay is not necessarily measured in terms of dollars. There are many ways to assess value and cost. Reputation or story points are two examples.

Quantifying cost of delay for any given feature can be challenging. Those who are good at it try to consider how cost of delay changes over time. They commonly use standards, such as the following.

**Linear** — For every day we do not deliver, we lose some money. A common example of a linear cost of delay is money lost due to competitors already having a feature that you don’t.

**Fixed date** — If we don’t deliver by a certain date, it’s too late. An example: let’s imagine you’re making New Year cards for 2019. If you don’t deliver them before the end of 2018, the cost of delay is very high. In fact, delivering afterward, in January or February, makes no difference — it is too late!

**Intangible** — We can delay for now at minimal cost, but eventually it could become expensive. A good example is the cost of delay for fixing a few bugs or refactoring your code. You can skip today, but over time it will make other improvements more expensive and can cause the cost of delay to increase exponentially.

**Expedite** — It must be done immediately or the cost of delay will grow radically. An example of an expedite feature is a severe bug that renders your product useless to all your customers.

The cost of delay categorization, when compared against cost of implementation, will often give you a good idea of what should be done first.