# **1) How is agile methodology different from traditional software development methodology? (Waterfall ad and dis-ad, agile ad and dis-ad, conclusion)**

Agile is an iterative approach to project management and software development that helps teams deliver value to their customers faster and with fewer headaches. Instead of betting everything on a "big bang" launch, an agile team delivers work in small, but consumable, increments. Requirements, plans, and results are evaluated continuously so teams have a natural mechanism for responding to change quickly.

Agile approaches are precise and customer focused. Customers have the opportunity to make modifications throughout project development phases. Agile proposes an incremental and iterative approach to development. Consider Agile Scrum Methodology, a Scrum Master interacts daily with the development team as well as the product owner to make sure that the product development is in sync with the customer’s expectations. During project inception, the customer splits the initial set of requirements into User Stories. The Scrum Master or Product owner organizes these User Stories and segregates them into different Sprints. In general, a sprint contains 3-4 User Stories to be delivered in 4 to 5 weeks, these are approximate figures and they will be decided based on the complexity of user stories. Once the Sprint planning is done, the selected User Stories are once again split into Tasks so that the developer can have a clear roadmap to deliver quality output. At the end of each Sprint, the customer gets a chance to review and predict the final outcome and can propose changes if any. The advantages of Agile over traditional development methodologies include:

* Requirements and solutions can be modified or changed at any time.
* User Stories can be provided periodically implying better chances for mutual understanding among developer and user.
* The solution can be determined by segregating the project into different modules and can be delivered periodically.
* Customer involvement for customer satisfaction and quality product.
* It is possible to create re-usable components.
* Less expenditure and time consumption on documentation.

Disadvantage of traditional SDM

* Customers might not be available all the time
* every team member is completely dedicated, without which weakens the principle of self-management.
* Project cost might be increased if not all requirements are finished within a fixed time-boxed.
* Co-location for efficient communication might not always possible.

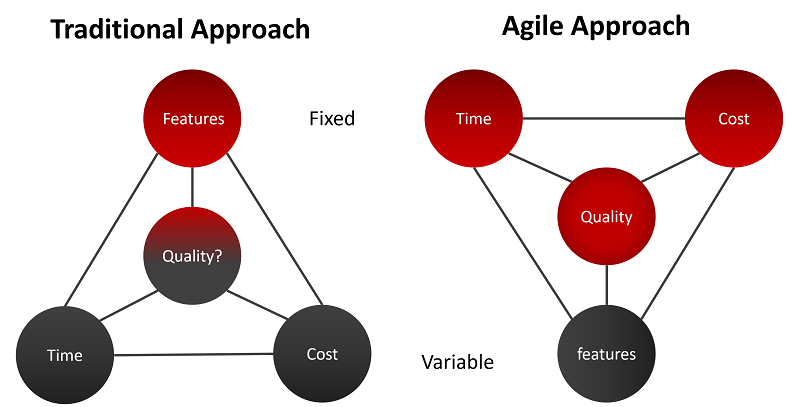
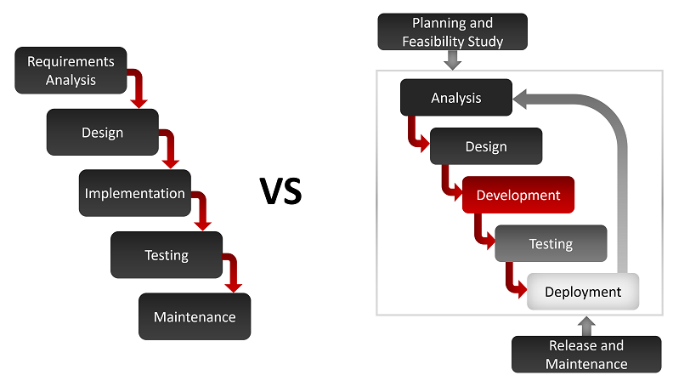
Traditional software development methodologies are based on pre-organized phases/stages. The flow of development is unidirectional, from requirements to design and then to development, then to testing and maintenance. In classical approaches like the Waterfall model, each phase has specific deliverables and detailed documentation that have undergone a thorough review process. Advantages of traditional SDM are listed below

* Simple and easy to understand and use.
* It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
* Phases are processed and completed once at a time. Phases do not overlap.

Disadvantage of traditional SDM

* Limited Customer involvement resulting in poor satisfaction
* Requirements cannot be modified
* More time consumption on documentation rather than in product development
* Only suited for the projects whose requirements and solution well-defined in advance

The main difference between traditional and agile approaches is the sequence of project are linear in traditional development methodologies where as they are iterative in agile.

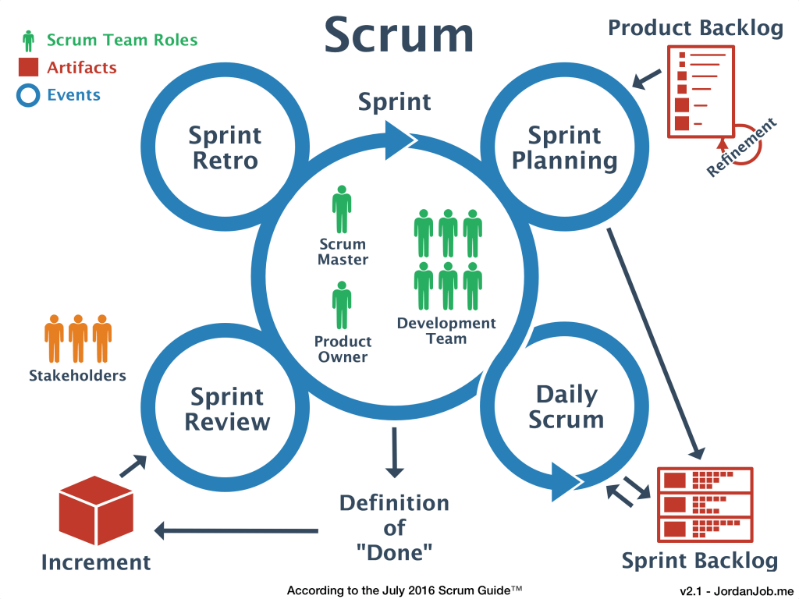


# **2) Explain the process of software development under scrum methodology?**

Agile approaches are precise and customer focused. Customers have the opportunity to make modifications throughout project development phases. Agile proposes an incremental and iterative approach to development.

Scrum Methodology is a framework in agile, scrum Master plays an important role. A Scrum Master interacts daily with the development team as well as the product owner to make sure that the product development is in sync with the customer’s expectations. Product Owner is the voice of the customer and hence is responsible for bridging the gap between the development team and customers. During project inception, the customer splits the initial set of requirements into User Stories. The Scrum Master or Product owner organizes these User Stories and segregates them into different Sprints.

In general, a sprint contains 3-4 User Stories to be delivered in 4 to 5 weeks, these are approximate figures and they will be decided based the complexity of user stories. Once the Sprint planning is done, the selected User Stories are once again split into Tasks so that the developer can have a clear roadmap to deliver quality output. At the end of each Sprint, the customer gets a chance to review and predict the final outcome and can propose changes if any.



# **3) Explain agile principles and manifesto.**

**Principles:**

1. Customer satisfaction with the early and continuous delivery of product is the highest priority.
2. Welcome changes in the requirements whenever needed.
3. Provide continuous delivery of working software within shorter timescale.
4. The developer and business people should work together daily throughout the project.
5. Build project around motivated individuals.
6. Face to face communication is the most effective method for sharing information to the development teams.
7. Working software is the primary measure of the progress.
8. Agile processes promote sustainable development.
9. Continuous excellence enhances agility.
10. Simplicity is essential.
11. Self-organizing teams generate most value.
12. Regularly reflect and adjust the way of work to boost effectiveness.

**Manifestoes:**

1. **Individuals and interactions over processes and tools:** In the past, a lot of software teams would concentrate on having the best possible tools or processes with which to build their software. [The Agile Manifesto](https://productboard.com/agile-manifesto/) suggests that while those things are important, the people behind the processes are even more so. Having the right group of individuals on your software team is vital to success. The best possible tools in the wrong hands are worthless. Perhaps even more important is how these individuals communicate with each other. The interactions between team members are what helps them to collaborate and solve any problems that arise.
2. **Working software over comprehensive documentation:** Previously, software developers would spend ages creating detailed documentation. That was before they even started writing a single line of code. And while documentation isn’t a bad thing, there comes a point when you should focus on providing your customers with working software. The Agile Manifesto places shipping software to your customers as one of the highest priorities. You can then [gather feedback to help you improve future releases](https://productboard.com/blog/3-techniques-to-quickly-discover-what-your-customer-really-needs/).
3. **Customer collaboration over contract negotiation:** Once upon a time, contracts were king. You would draw up contracts with your customers who would then detail the finished product. As a result, there was often a contrast between what the contract said, what the product did, and what the customer actually required. According to the Agile Manifesto, the focus should be on continuous development. You need to build [a feedback loop with your customers](https://productboard.com/blog/product-customer-feedback-loops/) so that you can constantly ensure that your product works for them.
4. **Responding to change over following a plan:** The Agile Manifesto suggests that a software team should have the ability to pivot and change direction whenever they need to, with a flexible roadmap that reflects that. A [dynamic roadmap](https://productboard.com/blog/4-roadmap-examples/) can change from quarter to quarter, sometimes even month to month, and agile teams are able to keep up with those changes.

# **4) List all the events of scrum. What is the purpose of using burndown chart?**

**Sprint Planning:** This is the event that starts before each Sprint and is which the Product Owner and Development team discuss which Product Backlog Items (PBI’s) that will be included in Sprint. The Development Team forecasts how many PBI’s they can deliver in the Sprint. The outcome of the sprint planning meeting is to get a sprint goal and sprint backlog that everyone agrees is realistic and achievable.

**Daily Scrum:** The Daily Scrum is time boxed to 15 minutes. Standing up is not compulsory. However, many teams find this a useful technique to keep the meeting short and to the point. The Daily Scrum is an opportunity for the Development Team to check in, assess progress towards achieving the Sprint Goal and to review and plan their activities for the next 24 hours.

**Backlog Grooming:** Backlog grooming is when the [product owner](https://www.agilealliance.org/glossary/product-owner/)and some, or all, of the rest of the team review items on the [backlog](https://www.agilealliance.org/glossary/backlog/) to ensure the backlog contains the appropriate items, that they are prioritized, and that the items at the top of the backlog are ready for delivery. This activity occurs on a regular basis and may be an officially scheduled meeting or an ongoing activity. Some of the activities that occur during this refinement of the backlog include:

* removing [user stories](https://www.agilealliance.org/glossary/user-stories/) that no longer appear relevant
* creating new user stories in response to newly discovered needs
* re-assessing the relative priority of stories
* assigning estimates to stories which have yet to receive one
* correcting estimates in light of newly discovered information
* [splitting](https://www.agilealliance.org/glossary/split/) user stories which are high priority but too coarse grained to fit in an upcoming [iteration](https://www.agilealliance.org/glossary/iteration/)

**Sprint Review:** A Sprint Review usually takes place on the last day of the Sprint and allows you the opportunity to show the “done” Increment to stakeholders (customers, management and anyone else considered relevant and interested). As well as demonstrating working features produced during the Sprint, you’re also after useful feedback that can be incorporated the Product Backlog that may help guide the work for future sprints.

**Sprint Retrospective:** This is final meeting and the Scrum team reviews what could be improved for future Sprints and how they should do it. The ethos of Scrum dictates that no matter how good the Scrum team is, there will always be opportunity to improve and the Sprint Retrospective gives the team a dedicated time in which to identify, discuss and plan this. The whole Scrum Team should take part including the Development Team, the Scrum Master and the Product Owner. The meeting should be a collaborative effort, just like the entire Scrum and Agile process.

**Burndown Chart**

A burndown chart is a tool used by [Agile teams](https://www.planview.com/resources/articles/definition-of-agile/) to gather information about both the work they have completed on a project and the work that is yet to be done within a given time period. Burndown charts allow team members to see the progress of individual sprints. This allows individuals to see how their work is progressing and if the entire team is on track or behind schedule.

# **5) Explain the roles in scrum.**

**Scrum Master:** Scrum Master is the person who is responsible for coaching the Development Team and the Product Owner to work on the day-to-day development activities. He is the one who ensures that the team understands the Scrum Values and Principles and is able to practice them. At the same time, Scrum Master also assures that the Team feels enthusiastic about Agile in order to achieve the best out of the framework.

**Product Owner:** Product Owner is the voice of the customers and hence is responsible for bridging the gap between the development team and stakeholders. Product owner manages the gap in such a way which would maximize the value of the product being built.

**Development Team:** Unlike the other Scrum Team members, the Development Teamwork on the actual implementation of the potentially deliverable software which is to be delivered at the end of each Sprint. The Development Team may consist of people having specialized skills like Front-end Developers, Backend Developers, Dev-Ops, QA Experts, Business Analyst, DBA etc., but they are all referred to as Developers; No other titles are allowed. The Development Team cannot even have sub-teams within it like the testing team, requirement specification team etc.

# **6). Explain the TDD and BDD process.**

**TDD:** TDD stands for Test Driven Development. It is Test Driven Development. This means writing a test that fails because the specified functionality doesn't exist, then writing the simplest code that can make the test pass, then refactoring to remove duplication, etc. You repeat this Red-Green-Refactor loop over and over until you have a complete feature. In this software development technique, we create the test cases first and then write the code underlying those test cases. Although TDD is a development technique, it can also be used for automation testing development. The teams that implement TDD, take more time for development however, they tend to find very few defects. TDD results in improved quality of code and the code that is more reusable and flexible. TDD also helps in achieving high test coverage of about 90-100%. The most challenging thing for developers following TDD is to write their test cases before writing the code. Tests in TDD can only be understood by people with programming knowledge. TDD methodology follows a very simple 6 step process:

1. **Write a test case:** Based on the requirements, write an automated test case.
2. **Run all the test cases:** Run these automated test cases on the currently developed code.
3. **Develop the code for that test cases:** If the test case fails, then, write the code to make that test-case work as expected.
4. **Run test cases again:** Run the test cases again and check if all the test cases developed so far are implemented.
5. **Refactor code:** This is an optional step. However, it’s important to refactor your code to make it more readable and reusable.
6. **Repeat the steps 1- 5 for new test cases:** Repeat the cycle for the other test cases until all the test cases are implemented.

**Advantages**

* **You only write code that’s needed:** Following the principles, you’ve got to prevent writing production code when all of your test’s pass. If your project needs another feature, you would like a test to drive the implementation of the feature. The code you write is the simplest code possible. So, all the code ending up within the product is really needed to implement the features.
* **More modular design:** In TDD, you consider one microfeature at a time. And as you write the test first, the code automatically becomes easy to check. Code that’s easy to check features a clear interface. This leads to a modular design for your application.
* **Easier to maintain:** Because the different parts of your application are decoupled from one another and have clear interfaces, the code becomes easier to take care of, you’ll exchange the implementation of a microfeature with a far better implementation without affecting another module. you’ll even keep the tests and rewrite the entire application. When all the tests pass, you’re done.
* **Easier to refactor:** Every feature is thoroughly tested. you do not get to be afraid to form drastic changes because if all the tests still pass, everything is ok. Now, is extremely important because you, as a developer, improve your skills each and each day. If you open the project after six months of performing on something else, most likely, you will have many ideas on the way to improve the code. But your memory about all the various parts and the way they fit together isn’t fresh anymore. So, making changes is often dangerous. With an entire test suite, you’ll easily improve the code without the fear of breaking your application.
* **High test coverage:** There’s a test for each feature. This leads to a high-test coverage It develops gain confidence in your code.
* **Tests document the code:** The test code shows you ways your code is supposed to be used. As such, it documents your code. The test code is a sample code that shows what the code does and the way the interface has got to be used.
* **Less debugging:** How often have you ever wasted each day to seek out a nasty bug? How often have you copied a mistake message from XCode and looked for it on the web?

**Disadvantages**

* **No silver bullet:** Tests help to seek out bugs, but they cannot find bugs that you simply introduce within the test code and in implementation code. If you haven’t understood the matter you would like to unravel, writing tests most likely doesn’t help.
* **slow process:** If you begin TDD, you’ll get the sensation that you simply need an extended duration of your time for straightforward implementations. you would like to believe the interfaces, write the test code, and run the tests before you’ll finally start writing the code.
* **All the members of a team got to do it:** As TDD influences the planning of code, it’s recommended that either all the members of a team use TDD or nobody in the least. additionally, to the present, it’s sometimes difficult to justify TDD to the management because they often have the sensation that the implementation of latest features takes longer if developers write code that will not find themselves within the product half the time. It helps if the entire team agrees on the importance of unit tests.
* **Tests got to be maintained when requirements change:** Probably, the strongest argument against TDD is that the tests need to be maintained because the code has got to. Whenever requirements change, you would like to vary the code and tests. But you’re working with TDD. this suggests that you simply got to change the tests first then make the tests pass. So, actually, this disadvantage is that the same as before when writing code that apparently takes an extended time takes a long time.

**BDD:** BDD stands for Behavior Driven Development. It is Behavior Driven Development. This means creating an executable specification that fails because the feature doesn't exist, then writing the simplest code that can make the spec pass. You repeat this until a release candidate is ready to ship. BDD is an extension to TDD where instead of writing the test cases, we start by writing a behavior. Later, we develop the code which is required for our application to perform the behavior. The scenario defined in the BDD approach makes it easy for the developers, testers and business users to collaborate. BDD is considered a best practice when it comes to [automated testing](https://www.softwaretestinghelp.com/automation-testing-tutorial-1/) as it focuses on the behavior of the application and not on thinking about the implementation of the code. The behavior of the application is the center of focus in BDD and it forces the developers and testers to walk-in the customer’s shoes. Tests in BDD can be understood by any person including the ones without any programming knowledge. The process involved in BDD methodology also consists of 6 steps and is very similar to that of TDD.

1. **Write the behavior of the application:** The behavior of an application is written in simple English like language by the product owner or the business analysts or QAs.
2. **Write the automated scripts:** This simple English like language is then converted into programming tests.
3. **Implement the functional code:** The functional code underlying the behavior is then implemented.
4. **Check if the behavior is successful:** Run the behavior and see if it is successful. If successful, move to the next behavior otherwise fix the errors in the functional code to achieve the application behavior.
5. **Refactor or organize code:** Refactor or organize your code to make it more readable and re-usable.
6. **Repeat the steps 1-5 for new behavior:** Repeat the steps to implement more behaviors in your application.

**Advantages**

* Greater clarity on business goals and customer requirements.
* Reaches a larger customer set as it uses non-technical languages.
* Helps in defining acceptance criteria before development.
* Focuses on the system’s behavior from the client’s and developer’s point of view.
* Helps in avoiding unnecessary features and includes important features.
* Reduces effort for post-modification and post-deployment defects.
* Avoids misinterpretations during the development process.

**Disadvantages**

* Requires continuous availability of users and developers
* Needs most requirements to be more detailed upfront than other approaches e.g., EDA
* Maintaining increasing numbers of tests
* External System setup can be hard
* Poorly written tests can lead to higher test-maintenance costs
* BDD works best in an Agile or iterative context
* BDD does not work not well in an enclosed environment

# **7). Write Short Notes.**

## **1). Planning Poker**

Planning poker is a planning and [estimation technique](https://www.wrike.com/blog/project-estimation-techniques/) used by [Agile teams](https://www.wrike.com/project-management-guide/faq/what-is-an-agile-team/) after a [product backlog](https://www.wrike.com/blog/backlog-grooming-best-practices/) has been created. The setup of this technique helps software teams accurately estimate product development time frames, improve time management and productivity. Planning poker in Agile is usually played by the Scrum master, product manager, developers, QA testers, and UX designers. The product manager begins the process by describing the user story to the estimators. The estimators can also ask questions in case a certain feature isn’t clear to them. To whom Agile poker cards with the number sequence 1, 2, 3, 5, 8, 13, 21 are given. These values represent each person’s estimation of the story points. As soon as the estimators are done assessing the user story, they reveal their cards at the same time. If the estimators choose the same number, then a consensus is reached and they can move on to the next story point. If not, the estimators discuss their estimates. Those with the highest or lowest number give reasons for their choice of number and try to get other estimators to consider their perspective. After a brief discussion, all participants repeat the Agile poker planning technique until a common number is established.  Just like an online game of poker, you can also play Agile poker online. This is helpful especially for [remote teams](https://www.wrike.com/remote-work-guide/managing-remote-workers-and-virtual-teams/) where the planning poker process remains virtually the same and is a great resource for prioritizing items in your [backlog](https://www.wrike.com/blog/backlog-grooming-best-practices/).

## **2. Velocity (Throughput in Kanban)**

Velocity ​​in agile terms means the average amount of work a team can complete in one “delivery cycle”, typically a sprint or a release for Scrum teams or a time period such as a Week or a month for Kanban teams. An increase in velocity may signal an improvement in productivity or growth in skills and understanding–but your team should be focused on delivering a consistent performance, not just bursts of productivity that can’t be maintained. For example, the backlog may have user stories worth 400 story points in the backlog. Based on past performance, if the team has demonstrated a sprint velocity of 40 story points, the team may need about 10 iterations to complete all the work in the backlog. If each iteration is two weeks, then the project will last 20 more weeks. The Sprint or Release velocity is unique to each team and the last thing you want to do is to compare or equate two teams’ velocity.  Each team is unique and so is their velocity. For example, if team A has demonstrated a sprint velocity of 40 story points over 4 sprints and team B has shown it to be 60 story points also over 4 sprints, that does not mean that team B has a higher yield. Each team has a unique estimation culture, as are their estimates and so their velocity can be different too.

## **3. User story mapping**

User Story Mapping is a technique used in product discovery outlining the features of an existing product in a sensible order. Arranging all the [user stories](https://www.digite.com/agile/user-stories/) in functional groups helps to keep eye on the big picture while also providing all the details of the whole application. The main purpose of Story Mapping is to facilitate product discovery and prioritization of development work. The Story Map always shows how each individual story fits in the whole application. And this makes it easy to spot gaps and decide how important one is over another. Advantages of user story mapping are shown below,

* Easy undersedation of the whole application.
* Can have the big picture of the application in full view. Losing the big picture is a common complaint in agile teams.
* Source of encouragement for [iterative and incremental development](https://www.digite.com/agile/iterative-and-incremental-development/).
* Having the big picture, shows you where a user story fits in the whole system in a single glance.

## **4. CICD (Continuous Integration Continuous Development)**

[**Continuous integration (CI)**](https://www.synopsys.com/software-integrity/resources/knowledge-database/continuous-integration.html) is a software engineering practice where members of a team integrate their work with increasing frequency. Teams strive to integrate at least daily and even hourly, approaching integration that occurs continuously. Historically, integration has been a costly engineering activity. So, to avoid thrash, CI emphasizes automation tools that drive build and test, ultimately focusing on achieving a software-defined life cycle. When CI is successful, build and integration effort drops, and teams can detect integration errors as quickly as practical. [**Continuous delivery (CD)**](https://www.synopsys.com/software-integrity/resources/knowledge-database/continuous-delivery.html) is to packaging and deployment what CI is to build and test. Teams practicing CD can build, configure, and package software and orchestrate its deployment in such a way that it can be released to production in a software-defined manner (low cost, high automation) at any time. High-functioning CI/CD practices directly facilitate agile development because software change reaches production more frequently. As a result, customers have more opportunities to experience and provide feedback on change.

## **5. Sprint Backlog**

A sprint backlog is the set of items that a cross-functional product team selects from its product backlog to work on during the upcoming sprint. Typically, the team will agree on these items during its sprint planning session. A clear sprint backlog prevents scope creep by clarifying exactly what your team will be doing—and not doing—during each sprint. The Sprint Backlog is a plan by and for the Developers. It is a highly visible, real-time picture of the work that the Developers plan to accomplish during the Sprint in order to achieve the Sprint Goal. This keeps information in one shared space in order to streamline communication and create one central source of sprint information. Items that are not in the backlog are not in scope. This creates a clear path, ensuring team members can focus on the task ahead to avoid scope creep. Consequently, the Sprint Backlog is updated throughout the Sprint as more is learned. From planning to organizing and managing agile projects, sprint backlogs help to collaborate project components as a team.

## **6. Product Backlog**

A product backlog is a list of the new features, changes to existing features, bug fixes, infrastructure changes or other activities that a team may deliver in order to achieve a specific outcome. The product backlog is the single authoritative source for things that a team works on. Depending on the size of the organization, it may have one central product backlog or multiple product backlogs for different teams. The product owner will refine the product backlog periodically to make sure the most important initiatives are at the top and each initiative has all of the information needed to execute against it. An effective product backlog breaks down each of the backlog items into a series of steps that help the development team. Prioritizing product backlog items, choosing which product backlog items should be eliminated and supporting product backlog refinement are key tasks in maintaining the product backlog. A product backlog can be an effective way for a team to communicate what they are working on and what they plan to work on. The product backlog can be represented both in physical form using index card or sticky notes and also in electronic form using spreadsheet or any tools like Trello.

## **7. Daily Standup**

A daily stand-up is a daily status meeting among all team members and it is held roughly for 15 minutes. Every member has to answer three important questions which are What I did yesterday? What I'll do today? Any impediment I am facing.../ I am blocked due to... Daily stand-up is for status update, not for any discussion. For discussion, team members should schedule another meeting at a different time. Participants usually stand instead of sitting so that the meeting gets over quickly. It provides visibility to the team on any delay or obstacles. The team can evaluate the progress on a daily basis and see if they can deliver as per the iteration plan. The scrum master, the product owner, and the delivery team should attend the stand-up on a daily basis. It is the scrum master's responsibility to take note of each team member's queries and the problems they are facing.

## **8. Task**

A task is a single unit of work broken down from a [user story](https://www.agile-academy.com/en/agile-dictionary/user-story/). A task is usually completed by just one person. Tasks are used in [Scrum](https://www.agile-academy.com/en/agile-dictionary/scrum/) to identify small increments of work to be completed during a [sprint](https://www.agile-academy.com/en/agile-dictionary/sprint/). The team writes the tasks on a card and put it on their [task board](https://www.agile-academy.com/en/agile-dictionary/task/). Benefits of tasks are listed below.

* Break user stories down into manageable units.
* Empower team members to complete tasks without feeling overwhelmed.
* Easy to identify on task boards.

## **9. Epic**

In agile development, an epic represents a series of user stories that share a broader strategic objective. When several epics themselves share a common goal, they are grouped together under a still-broader business objective, called a theme. Another important distinction is that a user story can be completed within the timeframe of an agile sprint. An epic will typically require development work covering several sprints. An epic sits between a [theme](https://www.productplan.com/glossary/theme/) and a [story](https://www.productplan.com/glossary/user-story/) in the agile development strategic hierarchy. A theme represents a team’s high-level strategy for its product. Let’s say the senior management of a major theater chain tasks its product team with filling empty seats in its theaters. This has become a common problem for movies that have been showing for several weeks. Here is how the product team might plan its agile development of a solution.

**Theme: -** Fill empty seats in theaters

**Epic: -** Use a mobile app to drive last-minute ticket sales

**Stories: -**

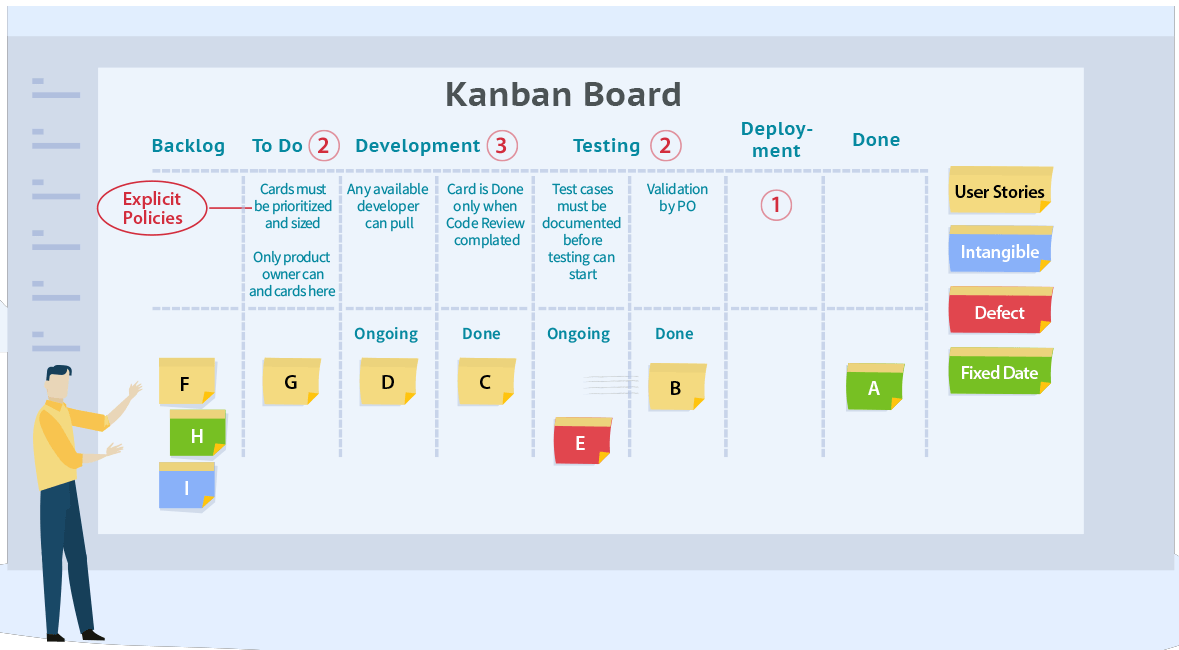
* Create and assign promotional codes for last-minute purchases
* Add text-message capability to the mobile app, to send last-minute promos and coupons
* Develop creative for promo emails and SMS texts

## **10. Integration Testing**

Integration testing is one of the very important software testing types of the software development phase where individual modules are integrated and then tested as a group to validate if they are working as per the requirements and expectations. Integration testing is an important testing technique in any project, and perhaps even more so in Agile projects and programs, because it is core to the concept of the “definition of done.”  While important to Agile, actually performing integration testing can be difficult unless you have layered the technique into the Agile framework. In agile projects, teams practice test-driven development, write small isolated unit tests, and further the small bit of code for making the unit test work. And, the later to confirm that the different layers communicate well, the agile teams efficiently write integration tests. Integration testing involves testing the integration and interaction of a number of different software modules that comprise a complete solution to ensure that they integrate successfully without error.

# **8). Many teams use Kanban board. how this can be used to increase productivity? Explain with diagrams (Kanban Board, Visibility, Simplicity, Product Increment)**

A Kanban board is an agile project management tool designed to help visualize work, limit work-in-progress, and maximize efficiency (or flow). These are a powerful tool for setting objectives, deadlines and work stages in a very visible form. [**Kanban boards are advantageous**](https://blog.proofhub.com/what-are-the-advantages-of-kanban-board-system-ec7b4c3e8807) to all kinds of teams out there. This is a methodology that promotes team collaboration. It is easy to use and is open to personalization.



**It Helps Elaborate the Workflow: -** Kanban board helps give your plan a visual interpretation. A Kanban methodology allows you to see your project, its progress, and its sub-tasks all at the same time. It is a very elaborate way of keeping everything in one place, even documents and files can be uploaded in the cards of the Kanban board.

**Kanban Is a Way to Maximize Efficiency: -** Kanban will cut your work time in half if you use it correctly. Because look, with this tool you can notify everyone about your work progress without having to go up to them or messaging them. The best part of these is that the task assignees will instantly get notified about your change in your work stage.

**Gives You a Way to Visualize Projects: -** Kanban is a great way to increase productivity because its best feature is that when you move from the various work stages, the assignees get notified. Using these the information flows effortlessly and the communication regarding the results takes place seamlessly.

**Helps to Keep Track of All Tasks: -** Kanban paves a way for you to track how much time a task would take in which stage and hence will help you manage time in a more effective way. These boards will also help you keep track of the performance of your subordinates both individually and as a team. This is why Kanban is most suitable for teams who aspire to follow the agile model.

**Gives A Chance to Break a Task into Work Stages: -** Kanban gives you a chance to break your tasks further into subtasks and also give you the liberty to assign people accordingly. Breaking tasks into stages leads to an uncluttered workflow and Kanban allows you to do just that with the littlest effort. When you break projects into tasks you can also provide the necessary links and files for your team to work with.

**Made For the Employer and The Employee: -** By now you may have seen that Kanban boards can only make tasks easier and information more accessible. Kanban boards can be used by the team to share their advancement in the tasks and to take feedback from the employer for the same. Therefore, this is something all your employees can get behind.

**Customize Your Workflow in A More Comprehensive Way: -** Name your work stages and attach labels and deadlines to tasks with Kanban boards provided by a good project management software. For example, when you have a team, the tasks will move in the workflow in stages like “ideas”, “to-do list”, “in progress”, “review”, “testing”, “launch”, etc. In addition to that, you can also prioritize and differentiate tasks with color tabs when it comes to personalizing the workflow and task lists.

**Provide Comments And Notes With The Utmost Ease: -** With a [**project management software**](https://www.proofhub.com/project-management-software) that provides Kanban boards for task assignment and work progress, you will be able to access a lot of features that will help you give your insight on the project. For example, in the task card that says “confirm client details” (for the sales industry), you can provide comments and give notes as to how the task has to be done.

**Recorded Work Progress Leads to A Sense of Accomplishment: -** Moving the task card from one task list to another will feel like a triumph and give you the strength and motivation to perform better and give you a sense of accomplishment. With data available in a neat and clean manner you get to view your advancement in the project in a visual way which gives you the motivation to go with the next stage of the task.

# **9). What is agile testing? Explain the principle of agile testing.**

**Agile** testing means testing software for defects or any other issues quickly or within the context of agile and give quick feedback for better and faster development of the project. Unlike the Waterfall method, Agile Testing can begin at the start of the project with continuous integration between development and testing. Agile Testing methodology is not sequential (in the sense it’s executed only after coding phase) but continuous. Testing and development run parallel in agile testing. The tester’s role lies not in identifying defects in the application but in behaving as a developer and bringing in improvements, suggestions, and test cases to be included in the application. In agile testing, testers follow the entire lifecycle but due to constant feedback and regular interaction between developers and customers, the application can be delivered in a short time without compromising on the quality. Agile testing requires continuous interaction between developers, customers, managers, stakeholders, and testers. Agile testing principle are listed below.

1. **Continuous testing: -** Agile teams perform tests regularly to make certain that the product is continuously progressing. Testing is done in conjunction with development.
2. **Continuous feedback: -** Testers provide continuous feedback to team members. Members regularly receive feedback regarding quality rather than [requirements](https://www.sealights.io/test-metrics/5-ways-to-do-continuous-integration-testing-wrong/).
3. **Involving the whole team**: - T[esters, developers and business analysts](https://www.sealights.io/webinars/how-to-be-an-effective-leader-with-testing-measures-and-software-quality-metrics-funnel-form/) all test the software.
4. **Quick feedback: -** The business team participates in each iteration; ongoing feedback reduces the time it takes to get feedback on development work.
5. **High-level software quality: -** Teams test the software to ensure the code is clean and tight. Through regular testing of the software, issues and vulnerabilities can be easily detected and fixed in the same iteration as they are developed.
6. **Less documentation: -** Teams use a reusable checklist. Agile development focuses on current customer needs rather than comprehensive, documented requirements and instructions.
7. **Test-driven: -** Testers evaluate the product at the time of implementation, rather than after implementation (as is the case with traditional testing methods).
8. **Customer satisfaction: -** Customers are exposed to their product during development. They can adapt and update requirements as development progresses. Tests can be modified to updated requirements.

# **10). What is meant by release candidate in agile? What is zero sprint planning in agile?**

A release candidate is commonly defined as a software version or a set of features that is functional yet not ready for being marketed, e.g., to finalize testing and receive user feedback ([source](https://www.wisegeek.com/what-is-a-release-candidate.htm)). In agile release planning, a release candidate may often refer to a feature that has been developed, is basically functional, and is in the process of being tested within an iteration. Once it has been fully tested and all errors have been fixed, it moves on to be included in the overall release. If release candidates fail in an iteration, their error fixing and finalization is often done in the subsequent iteration.

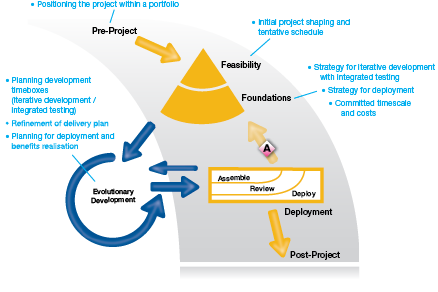
Sprint zero usually takes place before the formal start of the project and/or at a team's inception.  The goal of the Sprint is for the Development Team to come together to develop a minimal number of User Stories, project skeleton, story mapping, and develop a workable product. This Sprint should be kept lightweight and relatively high level. It is all about the origination of project exploration and gaining an understanding of where you want to head while keeping velocity low.   
  
The purpose of this Sprint, like any other Sprint, is to be as productive as possible.  It is not about intense software development, though. The Sprint should be a lightweight exercise.  By the end of this Sprint, the hope is you have done a prioritization exercise of features or a list of [User Stories](https://www.clearlyagileinc.com/agile-blog/user-story-estimation).  You may have a minimal environment set up to write your code, as well as a plan to develop the rest of the product once it is complete. From a high-level, Sprint zero has the task of trying to get ready for the subsequent Sprints to begin. A team is much more effective when they have: a defined release plan in mind, knows where the code is going to live, and how to implement that code.

The goal of this Sprint is to focus on completing the same activities as any other Sprint.  You want to work towards Agile Events, [updating the backlog](https://www.clearlyagileinc.com/agile-blog/the-ultimate-guide-to-the-sprint-backlog), taking part in the daily stand-ups, doing a retrospective, and delivering an end-product, whatever that may be in this type of set-up.  The expectation, in the end, is still to get to a [minimum viable product](https://www.clearlyagileinc.com/agile-blog/nefvmoeazgs25f8ror5ws78uknpbr0) or MVP.

Sprint zero can also work to the benefit of the Development Team to get them familiar with Scrum.  Think of it as an opportunity for the team to get a Scrum crash course, to understand the various Agile events and where they each fit.  The team can also get a rhythm, go through their forming, storming, norming, and performing phases early-on. The time can also get used to distribute the product mission and vision statement.The effect on the remaining Sprints, once the team gets going, becomes evident after this initial Sprint. The effect is that the team has a better understanding of how the Sprints will execute.  You have the framework and process in place. You get a feel for how you will interact during the various Agile Events and how to improve upon them going forward. Sprint zero should establish a solid foundation for the Scrum team to succeed.

# **11). Why DSDM is important? Explain its principles.**

The framework is based on the philosophy that every project must be clearly aligned with an organization’s strategic objectives and focused on delivering real value to the organization quickly. It covers the entire project life cycle and provides best practices for delivering projects on time, on budget, and with proven scalability to address projects of any size and in any business sector.



DSDM is designed to be easily adapted and used in conjunction with other traditional methods, those outlined in the PMBOK®, or to complement other agile approaches such as Scrum. In this sense, DSDM is an excellent reference model, both for companies that want to make a determined commitment to agility, and for all those organizations that wish to adopt dual IT models, combining traditional and agile approaches.

**Principles**

1. Focus on the business need
   1. Understand business priorities
   2. Develop a valid business case
   3. Ensure continuous business commitment
2. Delivery on time
   1. Use timeboxing techniques
   2. Focus on business priorities
   3. Predict deliveries and always hit deadlines
3. Cooperate and collaborate
   1. Involve stakeholders at the right time
   2. Develop a “one team” culture
   3. Involve business representatives
4. Never compromise quality
   1. Agree quality level at the start
   2. Ensure quality is not a variable
   3. Design, Document, and test
5. Build incrementally
   1. Deliver business benefits early where possible
   2. Reassess priorities with each increment
   3. Continuously confirm that the work is correct
6. Develop iteratively
   1. Develop using customer feedback
   2. Create enough design up front
   3. Experiment, evolve and be creative
7. Communicate continuously
   1. Mange stakeholder expectations
   2. Keep documentation simple and timely
   3. Interact through facilitated workshops
8. Demonstrate control
   1. Be proactive when monitoring progress
   2. Timebox work and review regularly
   3. Evaluate continuing project viability

# **12). What is JIT (Just in a time approach)? Explain kaizen.**

# **13). Compare and contrast agile and lean methodology.**

# **14). What is minimum viable product and where it is typically used?**

A minimum viable product, or MVP, is a product with enough features to attract early-adopter customers and validate a product idea early in the product development cycle. In industries such as software, the MVP can help the product team receive user feedback as quickly as possible to iterate and improve the product. Because the agile methodology is built on validating and iterating products based on user input, the MVP plays a central role in agile development.

Eric Ries, who introduced the concept of the minimum viable product as part of his Lean Startup methodology, describes [the purpose of an MVP](https://www.agilealliance.org/glossary/mvp/#q=~(infinite~false~filters~(tags~(~'mvp))~searchTerm~'~sort~false~sortDirection~'asc~page~1)) this way: It is the version of a new product that allows a team to collect the maximum amount of validated learning about customers with the least amount of effort. A company might choose to develop and release a minimum viable product because its product team wants to:

* Release a product to the market as quickly as possible
* Test an idea with real users before committing a large budget to the product’s full development
* Learn what resonates with the company’s target market and what doesn’t

In addition to allowing your company to validate an idea for a product without building the entire product, an MVP can also help minimize the time and resources you might otherwise commit to building a product that won’t succeed.