Agile Software Development

Agile Software Development

Agile software development is an umbrella term for a set of frameworks and practices based on the values and principles expressed in the Manifesto for Agile Software Development and the 12 Principles behind it.

> ## The Agile Manifesto
>
> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> **Individuals and interactions** over processes and tools
>
> **Working software** over comprehensive documentation
>
> **Customer collaboration** over contract negotiation
>
> **Responding to change** over following a plan
>
> That is, while there is value in the items on the right, we value the items on the left more.

> ## 12 Principles
>
> 1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
> 2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
> 3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
> 4. Business people and developers must work together daily throughout the project.
> 5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
> 6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
> 7. Working software is the primary measure of progress.
> 8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
> 9. Continuous attention to technical excellence and good design enhances agility.
> 10. Simplicity–the art of maximizing the amount of work not done–is essential.
> 11. The best architectures, requirements, and designs emerge from self-organizing teams.
> 12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Ultimately, Agile is a mindset informed by the Agile Manifesto's values and principles. Those values and principles provide guidance on how to create and respond to change and how to deal with uncertainty. The methodologies are the set of conventions that a team agrees to follow. Each team will have its own methodology, which will be different from every other team's methodology.

For Software Development and Project Management, Agile is an iterative approach 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 vs Waterfall

in the Software Engineering material, we have read about a software development model called Waterfall where the project development only moves to next phase of development or testing if the previous step is completed successfully. This method also greatly increases the risks sustained if the project has a long development time. Agile, on the other hand, has several smaller, same iterative development processes that emphasizes incremental development & testing of the project and communication between customers, developers, managers, and testers through a continuous iteration. Agile also greatly reduces the risks as stakeholders are consulted regularly. Refer to figure 1, below.

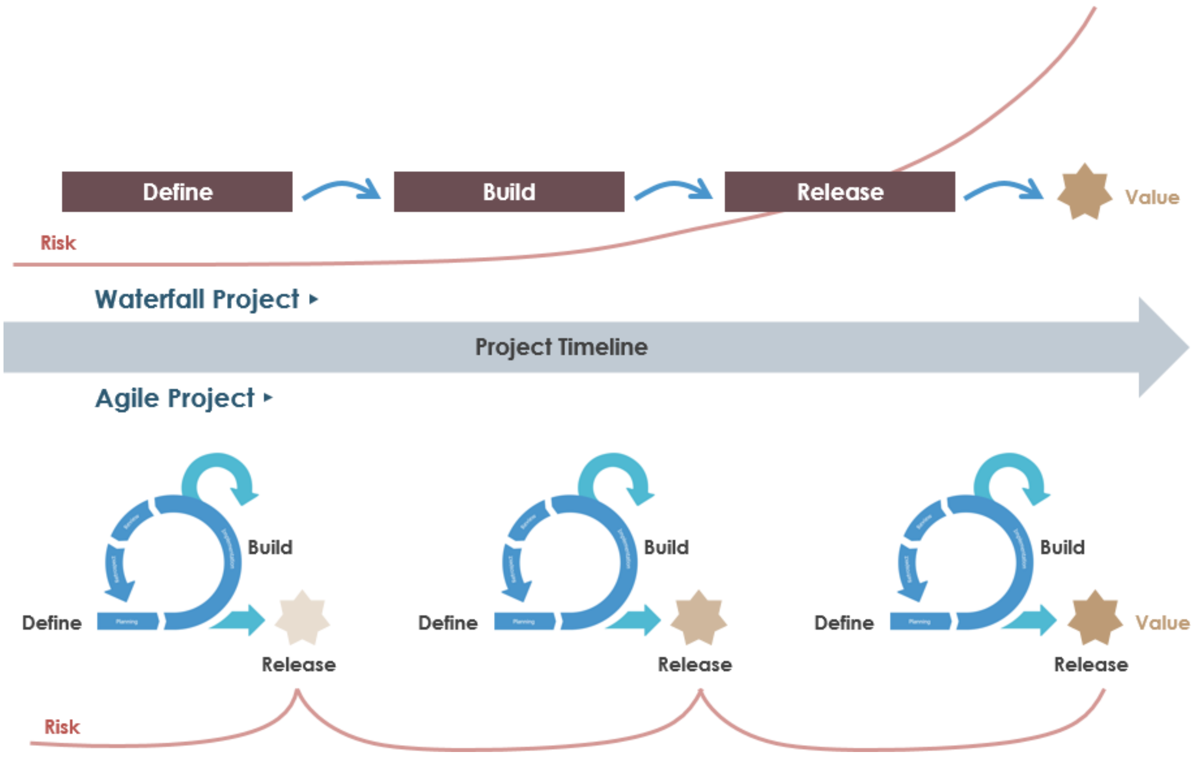


Figure 1: Agile vs Waterfall SDLC.

But what are the other differences of both models? Refer to the table on the next page.

Agile	Waterfall
It separates the project development lifecycle into sprints.	Software development process is divided into distinct phases.
It follows an incremental approach.	Waterfall methodology is a sequential design process.
Agile methodology is known for its flexibility.	Waterfall is a structured software development methodology so most times it can be quite rigid.
Agile can be considered as a collection of many different projects.	Software development will be completed as one single project.
Agile is quite a flexible method which allows changes to be made in the project development requirements even if the initial planning has been completed.	There is no scope of changing the requirements once the project development starts.
Agile methodology, follow an iterative development approach because of this planning, development, prototyping and other software development phases may appear more than once.	All the project development phases like designing, development, testing, etc. are completed once in the Waterfall model.
Test plan is reviewed after each sprint	The test plan is rarely discussed during the test phase.
Agile development is a process in which the requirements are expected to change and evolve.	The method is ideal for projects which have definite requirements and changes not at all expected.
In Agile methodology, testing is performed concurrently with software development.	In this methodology, the "Testing" phase comes after the "Build" phase
Agile introduces a product mindset where the software product satisfies needs of its end customers and changes itself as per the customer's demands.	This model shows a project mindset and places its focus completely on accomplishing the project.
Agile methodology works exceptionally well with Time & Materials or non-fixed funding. It may increase stress in fixed-price scenarios.	Reduces risk in the firm fixed price contracts by getting risk agreement at the beginning of the process.
Prefers small but dedicated teams with a high degree of coordination and synchronization.	Team coordination/synchronization is very limited.
Products owner with team prepares requirements just about every day during a project.	Business analysis prepares requirements before the beginning of the project.
Test team can take part in the requirements change without problems.	It is difficult for the test to initiate any change in requirements.

Agile	Waterfall
Description of project details can be altered anytime during the SDLC process.	Detail description needs to implement waterfall software development approach.
The Agile Team members are interchangeable, as a result, they work faster. There is also no need for project managers because the projects are managed by the entire team.	In the waterfall method, the process is always straightforward so, project manager plays an essential role during every stage of SDLC.

Despite the differences of both methods, let's state their advantages and limitations of both models. Refer to the table below and on the next page.

	Agile	Waterfall
Advantages	It is focused client process. So, it makes sure that the client is continuously involved during every stage.	It is one the easiest model to manage. Because of its nature, each phase has specific deliverables and a review process.
	Agile teams are extremely motivated and self-organized so it likely to provide a better result from the development projects.	It works well for smaller size projects where requirements are easily understandable.
	Agile software development method assures that quality of the development is maintained	Faster delivery of the project
	The process is completely based on the incremental progress. Therefore, the client and team know exactly what is complete and what is not. This reduces risk in the development process.	Process and results are well documented.
		Easily adaptable method for shifting teams
		This project management methodology is beneficial to manage dependencies.

	Agile	Waterfall
Limitations	It is not useful method for small development projects.	It is not an ideal model for a large size project
	It requires an expert to take important decisions in the meeting.	If the requirement is not clear at the beginning, it is a less effective method.
	Cost of implementing an agile method is little more compared to other development methodologies.	Very difficult to move back to makes changes in the previous phases.
	The project can easily go off track if the project manager is not clear what outcome he or she wants.	The testing process starts once development is over. Hence, it has high chances of bugs to be found later in development where they are expensive to fix.

Agile Frameworks

There are numerous methods that an organization can use when they adopt Agile for software development. Some methods focus on the practices (eg, Extreme Programming, Pair Programming, Agile Modeling), while other (frameworks) focuses on managing the flow of work (eg, Scrum, Kanban). Some support activities for requirements specification and development (eg, Feature Driven Development), while some seek to cover the full development life cycle (eg, Dynamic Systems Development Method, Rational Unified Process). Of all these frameworks, methods and practices, 2 frameworks (Scrum and Kanban) or a hybrid of them have stood out and have been widely adopted in organizations today.

Scrum

Scrum is an agile framework for developing, delivering, and sustaining complex products with an initial emphasis on software development. It is designed for teams of **ten** or **fewer members**, who break their work into goals that can be completed within time-boxed iterations, called **sprints**, no longer than one month and most commonly two weeks. The Scrum Team track progress in 15-minute time-boxed daily meetings, called **daily scrums**. At the end of the sprint, the team holds sprint review, to demonstrate the work done, and sprint retrospective to improve continuously.

Scrum Theory

Scrum is founded on empiricism and lean thinking. Empiricism asserts that knowledge comes from experience and making decisions based on what is observed. Lean thinking reduces waste and focuses on the essentials.

Scrum employs an iterative, incremental approach to optimize predictability and to control risk. Scrum engages groups of people who collectively have all the skills and expertise to do the work and share or acquire such skills as needed.

Scrum combines four formal events for inspection and adaptation within a containing event, the Sprint. These events work because they implement the empirical Scrum pillars of transparency, inspection, and adaptation.

Transparency
The emergent process and work must be visible to those performing the work as well as those receiving the work. With Scrum, important decisions are based on the perceived state of its three formal artifacts. Artifacts that have low transparency can lead to decisions that diminish value and increase risk.

Transparency enables inspection. Inspection without transparency is misleading and wasteful.

Inspection
The Scrum artifacts and the progress toward agreed goals must be inspected frequently and diligently to detect potentially undesirable variances or problems. To help with inspection, Scrum provides a sequence of five events detailed in the section *Scrum Events*.

Inspection enables adaptation. Inspection without adaptation is considered pointless. Scrum events are designed to provoke change.

Adaption
If any aspects of a process deviate outside acceptable limits or if the resulting product is unacceptable, the process being applied or the materials being produced must be adjusted. The adjustment must be made as soon as possible to minimize further deviation.

Adaptation becomes more difficult when the people involved are not empowered or self-managing. A Scrum Team is expected to adapt the moment it learns anything new through inspection.

Scrum Values

Scrum is a feedback-driven empirical approach which is, like all empirical process control, underpinned by the three pillars of transparency, inspection, and adaptation. All work within the Scrum framework should be visible to those responsible for the outcome: the process, the workflow, progress, etc. For the successful use of Scrum, the three pillars require trust and openness in the team which depends on people becoming more proficient in living the five following values:

- **Commitment** - Team members individually commit to achieving their team goals, each and every Sprint.
- **Courage** - Team members know they have the courage to work through conflict and challenges together so that they can do the right thing.
- **Focus** - Team members focus exclusively on their team goals and the sprint backlog; there should be no work done other than through their backlog.
- **Openness** - Team members and their stakeholders agree to be transparent about their work and any challenges they face.
- **Respect** - Team members respect each other to be technically capable and to work with good intent.

Scrum Roles

Scrum has three roles: **product owner**, **scrum master** and the **development team members**. These scrum roles are not job titles, they describe the key responsibilities for those on the scrum team. The three roles give a minimum definition of responsibilities and accountability to allow teams to effectively deliver work.

Product Owner
The business is represented by the product owner who tells the development team what is important to deliver. Trust between these two roles is crucial. The product owner should not only understand the customer, but also have a vision for the value the scrum team is delivering to the customer. The product owner also balances the needs of other stakeholders in the organization.

So the product owner must take all these inputs and prioritize the work. This is probably their most important responsibility because conflicting priorities and unclear directions will not only reduce the effectiveness of the team, but also could break the important trust relationship that the business has with the development team. It is therefore crucial, for scrum teams to be successful, that only one person sets priority. That person is the product owner (refer to figure 2, on the next page).

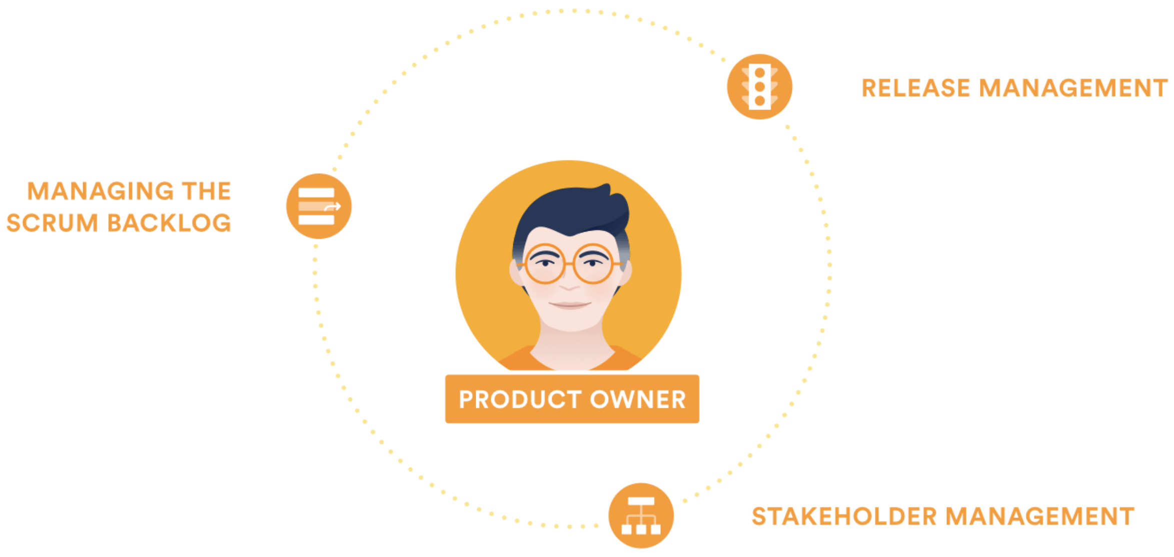


Figure 2: Scrum product owner role.

The Scrum Guide defines the product owners responsibilities as:

- **Managing the scrum backlog** - This does not mean that they are the only one putting in new product backlog Items into the backlog. But ultimately they are responsible for the backlog that the development team pulls to deliver from. That means the product owner should know about everything that is in the backlog and other people that add items to the product backlog should ensure that they communicate with the product owner.
- **Release management** - The sprint is not a release cycle, but instead a planning cycle. That means that scrum teams can deliver at any time. Ideally, they would deliver frequently throughout the sprint allowing the sprint review to review real customer usage and feedback. However continuous delivery is not always possible and other release models are required. It is important for the product owner to know when things can and should be released.
- **Stakeholder management** - Any product will have many stakeholders involved ranging from users, customers, governance and organizational leadership. The product owner will have to work with all these people to effectively ensure that the development team is delivering value. That can mean a large amount of stakeholder management and communication.

Scrum Master

The scrum master is the role responsible for gluing everything together and ensuring that scrum is being done well. In practical terms, that means they help the product owner define value, the development team deliver the value, and the scrum team to get to get better. The scrum master is a servant leader which not only describes a supportive style of leadership but describes what they do on a day-to-day basis.

They serve the product owner by helping them better understand and communicate value, to manage the backlog, help them plan the work with the team and break down that work to deliver the most effective learning. Serving the development team, the scrum master helps them self-organize, focus on outcomes, get to a “done increment,” and manage blockers. The scrum master also serves the organization at large, helping them understand what scrum is and create an environment that supports scrum (refer to figure 3, on the next page).

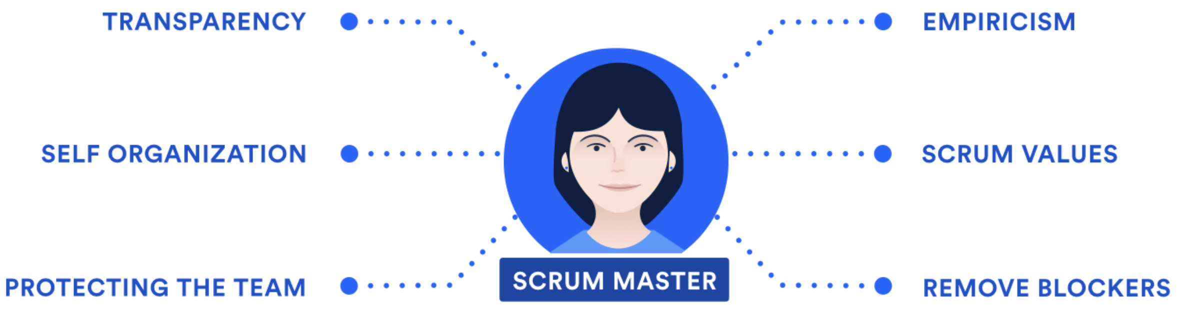


Figure 3: Scrum master role.

The scrum master focuses on:

- **Transparency** - To effectively inspect and adapt it is important that the right people can see what is going on. But this is actually much harder than it looks. The scrum master is tasked with ensuring that the scrum team works in a transparent way.
- **Empiricism** - A fundamental for scrum and agile approaches the idea that the best way of planning is to do work and learn from it. Empirical process is not easy and requires the scrum master to coach the scrum team on breaking down work, describing clear outcomes and reviewing those outcomes.
- **Self Organization** - Telling a development team they can self organize does mean that the team will self organize. In fact self organization comes over time and requires help and support.
- **Values** - Scrum defines 5 values of courage, focus, commitment, respect, and openness not because they are nice to have, but because they create an environment of physiological safety and trust. This environment is necessary for agility to thrive. Following the values is the responsibility of everyone in the scrum team, but the scrum master takes an active role in encouraging and reminding everyone of the importance of those values.

The scrum master serves the product owner in sprint planning and sprint reviews ensuring that value is clearly being described and direction set. They serve the development team in the daily scrum by ensuring that work is happening and that blockers are being removed. They also take responsibility for blockers that are outside of the the teams ability to resolve. The scrum master ensures that every opportunity to improve is made transparent to the scrum team and the retrospective has a clear set of outcomes that can be executed on.

Development Team

The development team are the people that do the work. They are not just the engineers, they can be comprised of all kinds of people including designers, writers, programmers, etc. The "developer" role in Scrum means a team member who has the right skills, as part of the team to do the work (refer to figure 4, below).

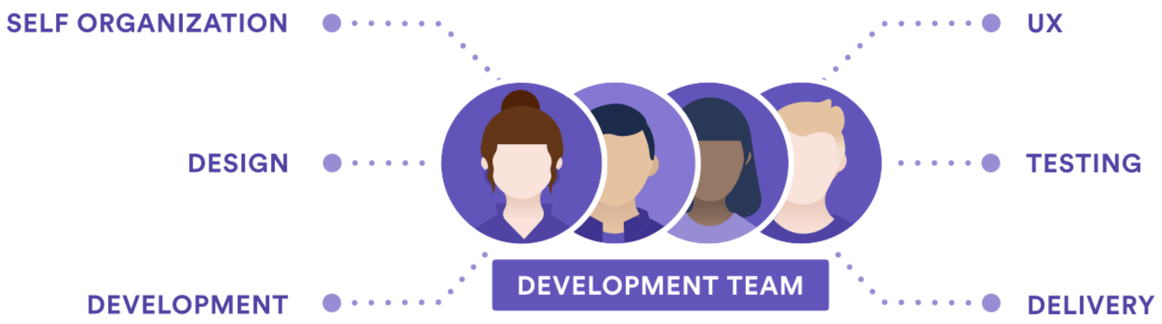


Figure 4: People comprised of a Scrum developer role.

The development team should be able to self-organize so they can make decisions to get work done. Self-organization isn't about disrespecting the organization, but rather about empowering the people closest to the work to do what's needed to solve the problem.

The development team's responsibilities include:

- Delivering the work through the sprint.
- To ensure transparency during the sprint they meet daily at the daily scrum (sometimes called a standup). The daily scrum provides transparency to the work and provides a dedicated place for team members to seek help, talk about success and highlight issues and blockers. The scrum master might facilitate the daily scrum, but ultimately it is the responsibility of the development team to run this meeting. It is their meeting to help them, as a group, to inspect and adapt the work they are doing and work in a more effective way.

Scrum Events (Workflow)

The Sprint
A sprint (also known as iteration or timebox) is the basic unit of development in Scrum. The sprint is a **timeboxed** effort; that is, the length is agreed and fixed in advance for each sprint and is normally between one week and one month, with two weeks being the most common. A new Sprint starts immediately after the conclusion of the previous Sprint.

Each sprint starts with a *sprint planning* event that establishes a sprint goal and the required product backlog items. The team accepts what they agree is ready and translate this into a sprint backlog, with a breakdown of the work required and an estimated forecast for the sprint goal.

During the Sprint:

- No changes are made that would endanger the Sprint Goal
- Quality does not decrease
- The Product Backlog is refined as needed
- Scope may be clarified and renegotiated with the Product Owner as more is learned

Each sprint ends with a *sprint review* and *sprint retrospective*, that reviews progress to show to stakeholders and identify lessons and improvements for the next sprints. Each Sprint may be considered a short project. At the end of each sprint (in the case of software), the deliverables would likely include the software has been fully integrated, tested and documented, and is potentially releasable. A Sprint could be cancelled if the Sprint Goal becomes obsolete. Only the Product Owner has the authority to cancel the Sprint.

Sprint Planning
Sprint Planning initiates the Sprint by laying out the work to be performed for the Sprint. This resulting plan is created by the collaborative work of the entire Scrum Team.

The Product Owner ensures that attendees are prepared to discuss the most important Product Backlog items and how they map to the Product Goal. The Scrum Team may also invite other people to attend Sprint Planning to provide advice.

Sprint Planning addresses the following topics:

- **Why is this Sprint valuable?** - The Product Owner proposes how the product could increase its value and utility in the current Sprint. The whole Scrum Team then collaborates to define a Sprint Goal that communicates why the Sprint is valuable to stakeholders. The Sprint Goal must be finalized prior to the end of Sprint Planning.

- **What can be Done this Sprint?** - Through discussion with the Product Owner, the Developers select items from the Product Backlog to include in the current Sprint. The Scrum Team may refine these items during this process, which increases understanding and confidence. The more experience the Developers have about their past performance, their upcoming capacity, and their Definition of Done, the more confident they will be in their Sprint forecasts.
- **How will the chosen work get done?** - For each selected Product Backlog item, the Developers plan the work necessary to create an Increment that meets the Definition of Done. This is often done by decomposing Product Backlog items into smaller work items of one day or less. How this is done is at the sole discretion of the Developers.

The Sprint Goal, the Product Backlog items selected for the Sprint, plus the plan for delivering them are together referred to as the Sprint Backlog. Sprint Planning is timeboxed to a maximum of eight hours for a one-month Sprint.

Daily Scrum
The purpose of the Daily Scrum is to inspect progress toward the Sprint Goal and adapt the Sprint Backlog as necessary, adjusting the upcoming planned work.

The Daily Scrum is a 15-minute event for the Developers of the Scrum Team. To reduce complexity, it is held at the same time and place every working day of the Sprint. If the Product Owner or Scrum Master are actively working on items in the Sprint Backlog, they participate as Developers.

The Developers can select whatever structure and techniques they want, as long as their Daily Scrum focuses on progress toward the Sprint Goal and produces an actionable plan for the next day of work. This creates focus and improves self-management.

Daily Scrums improve communications, identify impediments, promote quick decision-making, and consequently eliminate the need for other meetings.

The Daily Scrum is not the only time Developers are allowed to adjust their plan. They often meet throughout the day for more detailed discussions about adapting or re-planning the rest of the Sprint's work.

Sprint Review
The purpose of the Sprint Review is to inspect the outcome of the Sprint and determine future adaptations. The Scrum Team presents the results of their work (a demo) to key stakeholders and progress toward the Product Goal is discussed.

During the event, the Scrum Team and stakeholders review what was accomplished in the Sprint and what has changed in their environment. Based on this information, attendees collaborate on what to do next. The Product Backlog may also be adjusted to meet new opportunities. The Sprint Review is a working session and the Scrum Team should avoid limiting it to a presentation.

The Sprint Review is recommended to be timeboxed to a maximum of four hours for a one-month Sprint.

Sprint Retrospective
The purpose of the Sprint Retrospective is to plan ways to increase quality and effectiveness.

The Scrum Team inspects how the last Sprint went with regards to individuals, interactions, processes, tools, and their Definition of Done. Inspected elements often vary with the domain of work. Assumptions that led them astray are identified and their origins explored. The Scrum Team discusses what went well during the Sprint, what problems it encountered, and how those problems were (or were not) solved.

The Scrum Team identifies the most helpful changes to improve its effectiveness. The most impactful improvements are addressed as soon as possible. They may even be added to the Sprint Backlog for the next Sprint.

The Sprint Retrospective is recommended to be timeboxed to a maximum of three hours for a one-month Sprint.

Scrum Artifacts

Scrum's artifacts represent work or value. They are designed to maximize transparency of key information. Thus, everyone inspecting them has the same basis for adaptation.

Each artifact contains a commitment to ensure it provides information that enhances transparency and focus against which progress can be measured:

- For the Product Backlog it is the Product Goal.
- For the Sprint Backlog it is the Sprint Goal.
- For the Increment it is the Definition of Done.

These commitments exist to reinforce empiricism and the Scrum values for the Scrum Team and their stakeholders.

Product Backlog
The Product Backlog is an emergent, ordered list of what is needed to improve the product. It is the single source of work undertaken by the Scrum Team.

Product Backlog items that can be Done by the Scrum Team within one Sprint are deemed ready for selection in a Sprint Planning event. They usually acquire this degree of transparency after refining activities. Product Backlog refinement is the act of breaking down and further defining Product Backlog items into smaller more precise items. This is an ongoing activity to add details, such as a description, order, and size. Attributes often vary with the domain of work.

As the team works through the backlog, it must be assumed that change happens outside their environment such as learning about new market opportunities to take advantage of, competitor threats that arise, and feedback from customers that can change the way the product was meant to work. All of these new ideas tend to trigger the team to adapt the backlog to incorporate new knowledge. This is part of the fundamental mindset of an agile team as the world changes, the backlog is never finished.

The Developers who will be doing the work are responsible for the sizing. The Product Owner may influence the Developers by helping them understand and select trade-offs.

Sprint Backlog
The sprint backlog is the list of work the team must address during the next sprint. The list is derived by the scrum team progressively selecting product backlog items in priority order from the top of the product backlog until they feel they have enough work to fill the Sprint in order to achieve the Sprint Goal. Consequently, the Sprint Backlog is updated throughout the Sprint as more is learned. It should have enough detail that they can inspect their progress in the Daily Scrum.

The Sprint Goal is the single objective for the Sprint. Although the Sprint Goal is a commitment by the Developers, it provides flexibility in terms of the exact work needed to achieve it. The Sprint Goal also creates coherence and focus, encouraging the Scrum Team to work together rather than on separate initiatives.

The Sprint Goal is created during the Sprint Planning event and then added to the Sprint Backlog. As the Developers work during the Sprint, they keep the Sprint Goal in mind. If the work turns out to be different than they expected, they collaborate with the Product Owner to negotiate the scope of the Sprint Backlog within the Sprint without affecting the Sprint Goal.

Increment
An Increment is a concrete stepping stone toward the Product Goal. Each Increment is additive to all prior Increments and thoroughly verified, ensuring that all Increments work together. In order to provide value, the Increment must be usable.

Multiple Increments may be created within a Sprint. The sum of the Increments is presented at the Sprint Review thus supporting empiricism. However, an Increment may be delivered to stakeholders prior to the end of the Sprint. The Sprint Review should never be considered a gate to releasing value.

Work cannot be considered part of an Increment unless it meets the *Definition of Done*. The Definition of Done is a formal description of the state of the Increment when it meets the quality measures required for the product. The moment a Product Backlog item meets the Definition of Done, an Increment is born.

The Definition of Done creates transparency by providing everyone a shared understanding of what work was completed as part of the Increment. If a Product Backlog item does not meet the Definition of Done, it cannot be released or even presented at the Sprint Review. Instead, it returns to the Product Backlog for future consideration.

If the Definition of Done for an increment is part of the standards of the organization, all Scrum Teams must follow it as a minimum. If it is not an organizational standard, the Scrum Team must create a Definition of Done appropriate for the product.

The Developers are required to conform to the Definition of Done. If there are multiple Scrum Teams working together on a product, they must mutually define and comply with the same Definition of Done.

Limitations

The benefits of Scrum may be more difficult to achieve when:

- **Teams whose members are geographically dispersed or part-time** - In Scrum, developers should have close and ongoing interaction, ideally working together in the same space most of the time. While recent improvements in technology have reduced the impact of these barriers (eg, being able to collaborate on a digital whiteboard), the Agile manifesto asserts that the best communication is face to face.

- **Teams whose members have very specialized skills** - In Scrum, developers should have broad array of skills, allowing them to work on tasks outside of their specialization. This can be encouraged by good Scrum leadership. While team members with very specific skills can and do contribute well, they should be encouraged to learn more about and collaborate with other disciplines.
- **Products with many external dependencies** - In Scrum, dividing product development into short sprints requires careful planning; external dependencies, such as user acceptance testing or coordination with other teams, can lead to delays and the failure of individual sprints.
- **Products that are mature or legacy or with regulated quality control** - In Scrum, product increments should be fully developed and tested in a single sprint; products that need large amounts of regression testing or safety testing (eg, medical devices or vehicle control) for each release are less suited to short sprints than to longer waterfall releases.

Kanban

Kanban is a lean method (stems from the lean manufacturing method adapted from the Toyota Production System based on the principle "Just in time production") to manage and improve work across human systems. This approach aims to manage work by balancing demands with available capacity, and by improving the handling of system-level bottlenecks.

Work items are visualized to give participants a view of progress and process, from start to finish, usually via a Kanban board. Work is pulled as capacity permits, rather than work being pushed into the process when requested. Updates are released whenever they are ready, without a regular schedule or predetermined due dates.

In theory, Kanban does not prescribe a fixed time to deliver a task. If the task gets completed earlier (or later), it can be released as needed without having to wait for a release milestone like sprint review. Kanban does not have roles like in Scrum, the whole team owns the Kanban board. It’s the collective responsibility of the entire team to collaborate on and deliver the tasks on the board.

Lead time and cycle time are important metrics for Kanban teams. Lead time is calculated since the team gets a request from the client and Cycle time is calculated since the team starts working on a task. Lead time is used to understand how long a client has to wait for their product and cycle time is used to understand how fast the team produces a product. Improving cycle times indicates the success of Kanban teams.

Kanban is commonly used in software development in combination with other methods and frameworks such as Scrum.

Kanban Boards

Kanban boards, designed for the context in which they are used, vary considerably but the columns are mainly grouped into 5 components: Visual signals, columns, work-in-progress limits, a commitment point, and a delivery point. Refer to figure 5, on the next page. The aim is to make the general workflow and the progress of individual items clear to participants and stakeholders.

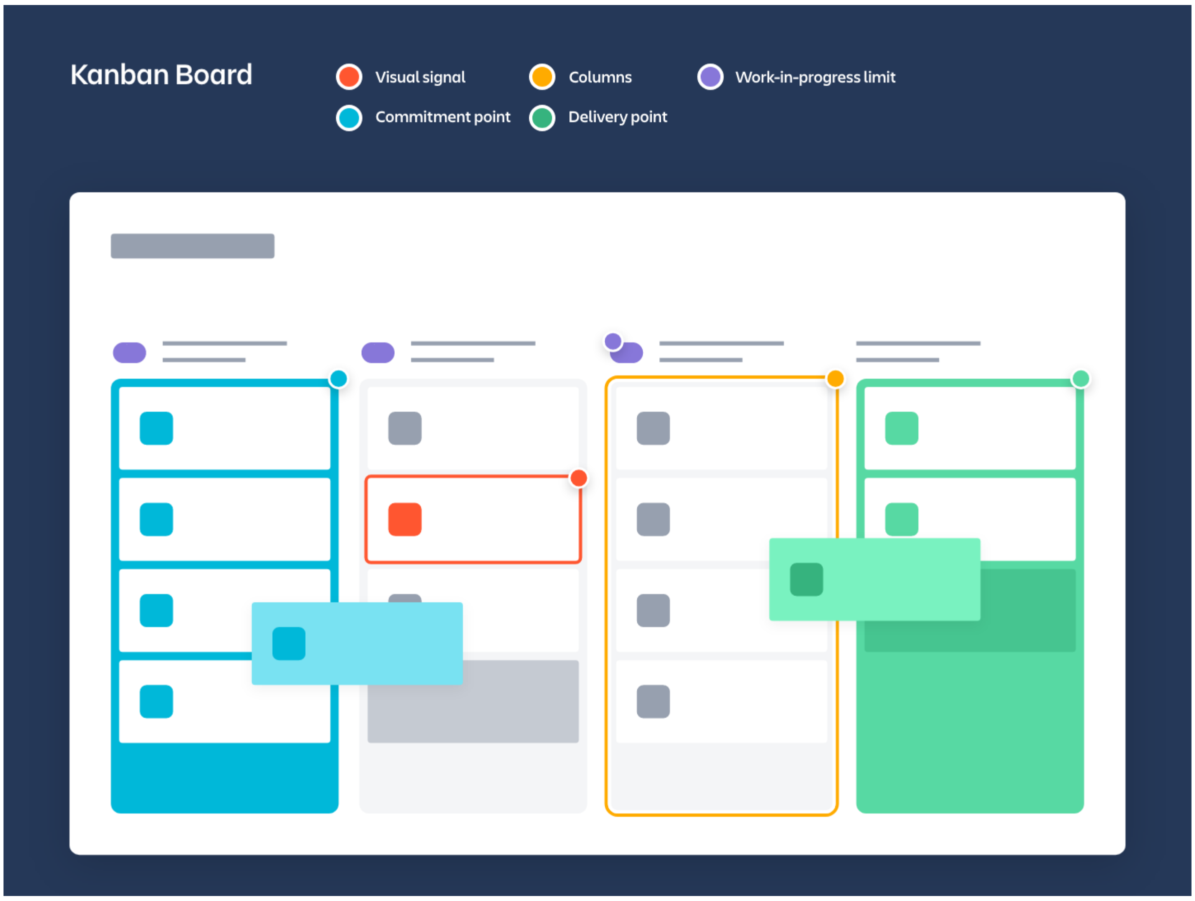


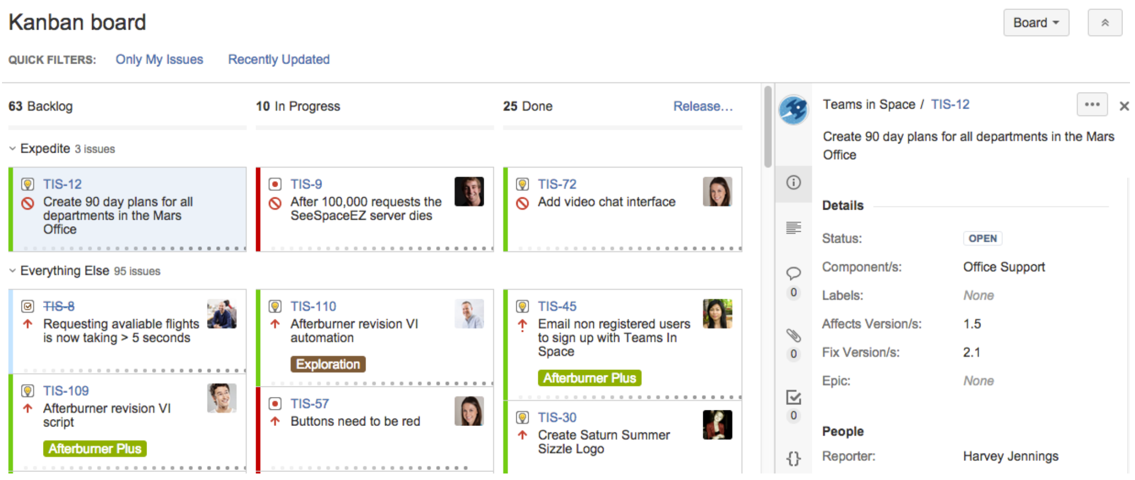
Figure 5: Kanban Board.

- **Visual Signals** - One of the first things you'll notice about a Kanban board are the visual cards (stickies, tickets, or otherwise). Kanban teams write all of their projects and work items onto cards, usually one per card. For agile teams, each card could encapsulate one user story. Once on the board, these visual signals help teammates and stakeholders quickly understand what the team is working on.
- **Columns** - Another hallmark of the Kanban board are the columns. Each column represents a specific activity that together compose a "workflow". Cards flow through the workflow until completion. Workflows can be as simple as "To Do," "In Progress," "Complete," or much more complex.
- **Work In Progress (WIP) Limits** - WIP limits are the maximum number of cards that can be in one column at any given time. A column with a WIP limit of three cannot have more than three cards in it. When the column is "maxed-out" the team needs to swarm on those cards and move them forward before new cards can move into that stage of the workflow. These WIP limits are critical for exposing bottlenecks in the workflow and maximizing flow. WIP limits give you an early warning sign that you committed to too much work.
- **Commitment point** - Kanban teams often have a backlog for their board. This is where customers and teammates put ideas for projects that the team can pick up when they are ready. The commitment point is the moment when an idea is picked up by the team and work starts on the project.
- **Delivery point** - The delivery point is the end of a Kanban team's workflow. For most teams, the delivery point is when the product or service is in the hands of the customer. The team's goal is to take cards from the commitment point to the delivery point as fast as possible. The elapsed time between the two is the called Lead Time. Kanban teams are continuously improving to decrease their lead time as much as possible.

The two primary practices of Kanban (as described in books on Kanban for software development) are to visualize your work and limit work in progress (WIP). There are 2 types pf Kanban boards: physical and digital.

- **Physical Boards** - simplest of Kanban boards. They are physical boards divided into vertical columns. Teams mark up a whiteboard or blackboard and place sticky notes onto the board. These sticky notes move through the workflow and demonstrate progress. Advantages of a physical board are: it's "always on", it's simple to set up, simple to show others, and is often times the better way to communicate with certain teams. However, physical boards are not ideal for remote teams or teams with people who have illegible handwriting.
- **Digital Boards** - allow teams that do not share a physical office space to use Kanban boards remotely and asynchronously. The advantages of a digital Kanban board like this are the speed to set it up, the ease in sharing it with others, and the ability to asynchronously track an infinite number of conversations and comments as the project progresses. No matter where or when team members check in on the Kanban board, they'll see the most up-to-date status of the project.

  Some digital Kanban boards are simple, and some are more robust and customizable. Teams that require additional functionality like WIP Limits and Control Charts should opt for a more powerful tool like Jira (refer to figure 6, below).


Kanban board

QUICK FILTERS: Only My Issues Recently Updated

63 Backlog	10 In Progress	25 Done
Expedite 3 issues		
TIS-12 Create 90 day plans for all departments in the Mars Office	TIS-9 After 100,000 requests the SeeSpaceEZ server dies	TIS-72 Add video chat interface
Everything Else 95 issues		
~~TIS-8~~ Requesting avaliable flights is now taking > 5 seconds	TIS-110 Afterburner revision VI automation Exploration	TIS-45 Email non registered users to sign up with Teams In Space Afterburner Plus
TIS-109 Afterburner revision VI script Afterburner Plus	TIS-57 Buttons need to be red	TIS-30 Create Saturn Summer Sizzle Logo

Teams in Space / TIS-12

Create 90 day plans for all departments in the Mars Office

Details

Status:	OPEN
Component/s:	Office Support
Labels:	None
Affects Version/s:	1.5
Fix Version/s:	2.1
Epic:	None

People

Reporter:	Harvey Jennings

Figure 6: Kanban Board in Jira.

The difference between Kanban and Scrum is actually quite subtle. By most interpretations, scrum teams use a Kanban board, just with scrum processes, artifacts, and roles along with it. There are, however, some key differences:

- Scrum sprints have start and stop dates whereas Kanban is an ongoing process.
- Team roles are clearly defined in scrum (product owner, dev team, and scrum master), while Kanban has no formal roles. Both teams are self-organized.
- A Kanban board is used throughout the lifecycle of a project whereas a scrum board is cleared and recycled after each sprint.
- A scrum board has a set number of tasks and strict deadline to complete them.
- Kanban boards are more flexible with regards to tasks and timing. Tasks can be reprioritized, reassigned, or updated as needed.

WIP Limits

In agile development, work in progress (WIP) limits set the maximum amount of work that can exist in each status of a workflow. Limiting the amount of work in progress makes it easier to identify inefficiency in a team's workflow by forcing the team to focus on a smaller set of tasks.

At a fundamental level, WIP limits encourage a culture of "done." More important, WIP limits make blockers and bottlenecks visible. Teams can swarm around blocking issues to get them understood, implemented, and resolved when there is a clear indicator of what existing work is causing a bottleneck. Once blockages are removed, work across the team begins to flow again.

Goals of WIP Limits:

1. **Size individual tasks consistently.** When breaking down requirements, it's important to keep individual tasks to no more than 16 hours of work. Doing so increases the team's ability to estimate confidently, and it helps prevent bottlenecks.
2. **Map WIP limits to the team’s skills.** Assuming that there is a specialist on the team, create status sets specific to the specialist's work. If bottlenecks occur in that status, use the opportunity to educate other team members to add additional capacity for the specialist's skill sets and increase flow across the entire team.
3. **Reduce idleness.** When a team member has some downtime, encourage them to help an upstream or downstream team member.
4. **Protect a sustainable engineering culture.** Work in progress limits do not mean developers need to rush through work to avoid work overload in a particular status. They are meant to support solid agile engineering practices that protect the quality of the product and health of the code base.

Scrumban

We have read about Scrum and Kanban in the above sections. Some organizations uses a combination of both Scrum and Kanban, thus the term *Scrumban*. Scrumban combines the best features of both frameworks for maintenance projects. It's raising in popularity with organizations where they have both ongoing development and maintenance projects.

In Scrum, a team member selects the work they'll be doing for the next sprint beforehand. The sprint is then locked, work is done and after a couple of weeks (the usual sprint duration) the queue is empty. Refer to figure 7, below.

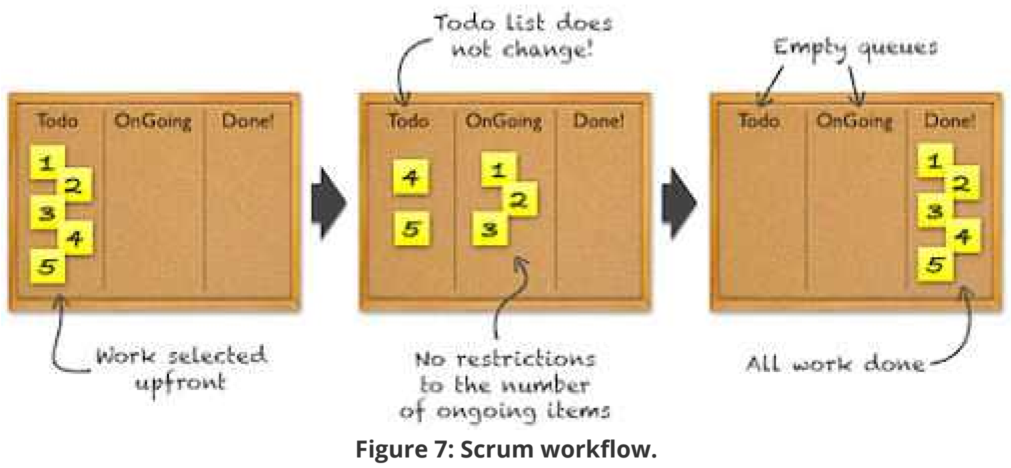

Todo list does not change!
Empty queues
Todo
OnGoing
Done!
1
2
3
4
5
Work selected upfront
No restrictions to the number of ongoing items
All work done

Figure 7: Scrum workflow.

In Kanban, all that's limited is the size of the queues, called the WIP limit. This means that a team member can change the items in the queues at any time, and that there's no "sprint end". The work just keeps flowing. Refer to figure 8, below.

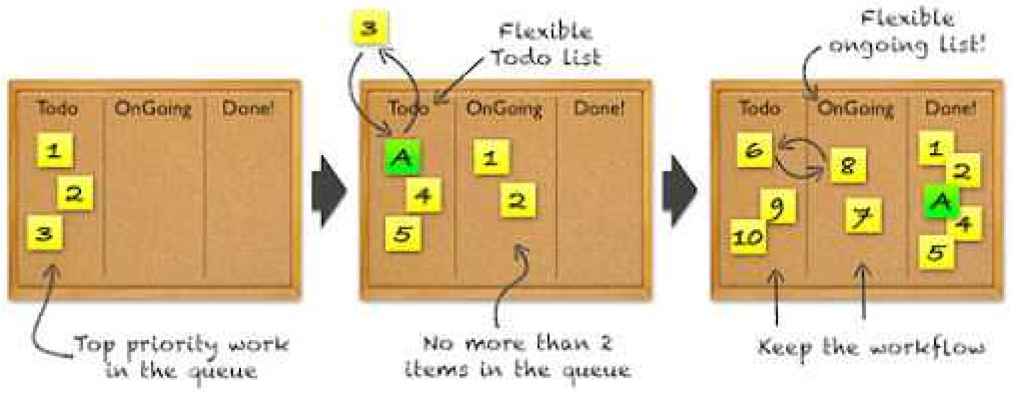


Figure 8: Kanban workflow.

In Scrumban, merges the structure and predictability of Scrum with Kanban's flexibility and continuous workflow. It involves applying Kanban principles (visualization of workflow, and flexible processes) to a team's Scrum framework. But, Scrumban removed some of the more rigid aspects of Scrum and left each team to create a custom approach to development.

A general guide to setup a Scrumban framework is as follows:

1. **Develop a Scrumban board** - A Scrumban board is similar to a Kanban board. Add as many columns to it as your team needs to mark each discrete phase of progress. But be careful not to create so many columns that the board becomes cluttered and difficult to view.
2. **Set your work-in-progress limits** - Recall that Scrum sets both time and task limits for every sprint. Kanban, by contrast, focuses on continuous workflow. Team members will need to establish a limit on how much work they can take on at any one point. For Scrumban, that limit will be the number of total cards on the board at any time. Set a realistic limit to keep the team from becoming overwhelmed and frustrated.
3. **Order the team's priorities on the board** - With Scrum, the Scrum Master will assign tasks to specific individuals within their development group for each sprint. On the other hand, Scrumban, focuses on establishing the priority order of all projects on the board. The team will decide which member will tackle which tasks.
4. **Throw out time-boxed cards** - Because each sprint has a strict time limit, and the team can work on only a predefined set of projects during any one sprint, a Scrum team needs to estimate how long each development task will take. With Scrumban, work is continuous and not time-limited, so the team would not need to estimate any time limits. The team leader will only need to focus on prioritizing the most important projects.
5. **Set your daily meetings** - Although there will not be as many meetings as typical of the Scrum framework (sprint planning, sprint review, retrospective), Scrumban meetings can include short standups for the team to discuss their plans and challenges for the upcoming day. These short meetings are also a good way to encourage team bonding and cohesion because the developers will be spending a lot of time working individually on their tasks and might not have much time for interaction otherwise.

Agile Testing

Now that we are familiar with the Agile Frameworks for software development, we will be moving on to Agile Testing. Referring back to figure 1, we can see that testing in Agile is done often and along the same small development cycle as in the software development life cycle. Therefore, Agile testing is a software testing process that follows the principles of Agile Software Development. Recall that Agile Software Development is an approach based on the iterative development of software where the requirements and solutions evolve through collaboration between cross-functional teams.

Agile testing involves all members of the above group of people forming a cross-functional team, to test the software under development. The testing process is interwoven into all the development phases (such as requirements, design, and coding). As the common objective is to achieve high-quality products at every major milestone, Agile testing is done interactively, incrementally and in parallel as the software is being developed.

Agile Testing Principles

In general, the Agile method entails giving more weight to the individuals and their interactions over processes and tools to deliver working software with the focus on customers rather than follow a plan. A valued trait of Agile teams are their responsiveness to change be it from customers feedback or from testing. Therefore, this bring us to the principles that Agile testing follows. Note that, within Agile Testing, there are between 7 to 10 principles (depending on which source the principles are referenced from) but the main principles are listed below:

- **Testing is continuous** - starts from the beginning and runs in parallel with the development process. This ensures that the product development is continuously progressing.
- **Continuous feedback** - the testing team provides early and regular feedback regarding quality. This in turn reduces the cost of fixing bugs as they are detected earlier in the development phase.
- **Tests are performed by the whole team** - entire team that is responsible for quality, includes the customer performing User Acceptance Tests to make sure that quality remains high.
- **Quick feedback** - the business team gets quick feedback and changes are made early on to the requirements. This also reduces cost when requirements are changed according to the received feedback.
- **High-level software quality** - defects are raised and fixed with each regular manual and automated tests. This ensures that the code is kept clean at any milestone of development.
- **Lightweight documentation** - reusable general test scenario are used rather than a log of incidental details. This helps to bring back the focus on the essence of testing rather than have test documentation for every set of test data regardless of testing environment.
- **Test-driven** - testing is done alongside implementation with unit testing done even before implementation rather than after implementation (as is the case with traditional testing methods).

All these principles brings business value to the project as it reduces overall cost of the project thus making the business more profitable. The whole team also takes responsibility for delivering a high-quality product that makes the customers happy.

Agile Testing Methodologies

There are many testing methods for Agile testing. Some of the most widely used are described below:

1. **Test Driven Development (TDD)**
   Test-Driven Development is a method where test cases are written firstly with a failure scenario, codes are developed so that it passes the test and the method is repeated throughout the code development. Use the following steps and figure 9, below, to help with understanding the TDD process:

   1. Write a test case to reflect the expected behaviour of the code that is to be written.
   2. Run the test. The test will fail since the code is not written yet.
   3. Develop the code base on the test case.
   4. Run the test again to see if the codes pass the test. Repeat steps 3 and 4 till the written codes pass the test.
   5. Refactor the code.
   6. Run the test on the refactored code and make sure it passes.

   The steps are repeated with each new functionality that needs to be developed. Earlier and new tests are executed every time to ensure the codes are working as expected. Automation is used to make this process fast. These test are generally at the unit, integration or system level.

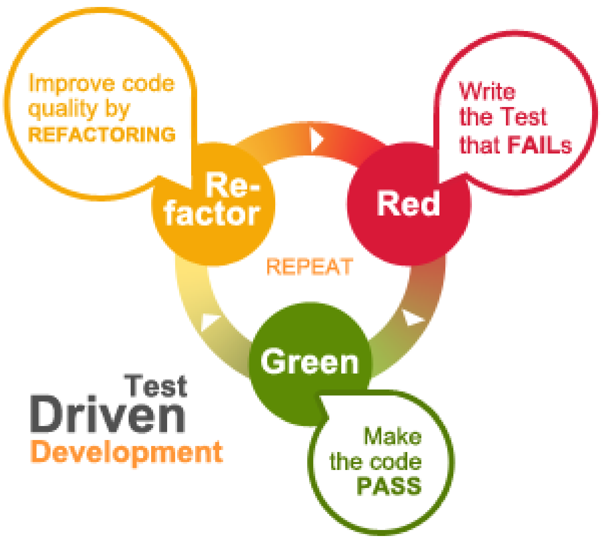


Figure 9: Test Driven Development.

2. **Behaviour Driven Development (BDD)**
   BDD is based on the same principles of Test Driven Development (TDD). In BDD, the test cases are more likely to be written from the business point of view where the feature/s functionality is tested not just the individual components (as in TDD). They contain information on how a feature behaves in different situations with varying input parameters. Figure 10, on the next page, shows the process of BDD.

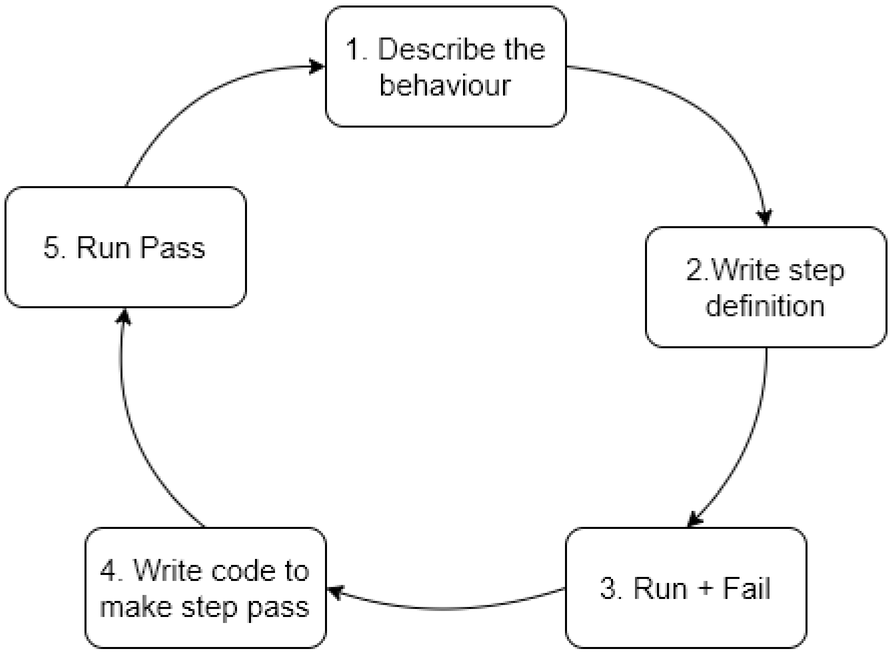


Figure 10: Behaviour Driven Development.

Best practices for testers using a BDD methodology:

- Streamline documentation to ensure the process is efficient
- Embrace a “three amigos” approach, where the developer, product owner, and tester work together to define scenarios and tests
- Use a declarative test framework, such as Cucumber, to specify criteria
- Build automated tests and reuse them across scenarios
- Have business analysts write test cases

3. **Acceptance Test Driven Development (ATDD)**
   ATDD is similar to BDD but it has one more entity, the customer in addition to the developers and the testers. The tests are written in the same way as BDD but the tests are written from the user’s perception. That means, the customer focuses on the problem, the developer pays attention to how the problem will be solved and the tester looks at what could go wrong. Customers’ feedback specifies how the system will function. These feedback are recorded and used to develop the acceptance criteria, this criterion is then translated into either manual or automated acceptance tests and codes are developed against those tests. Refer to figure 11, on the next page.

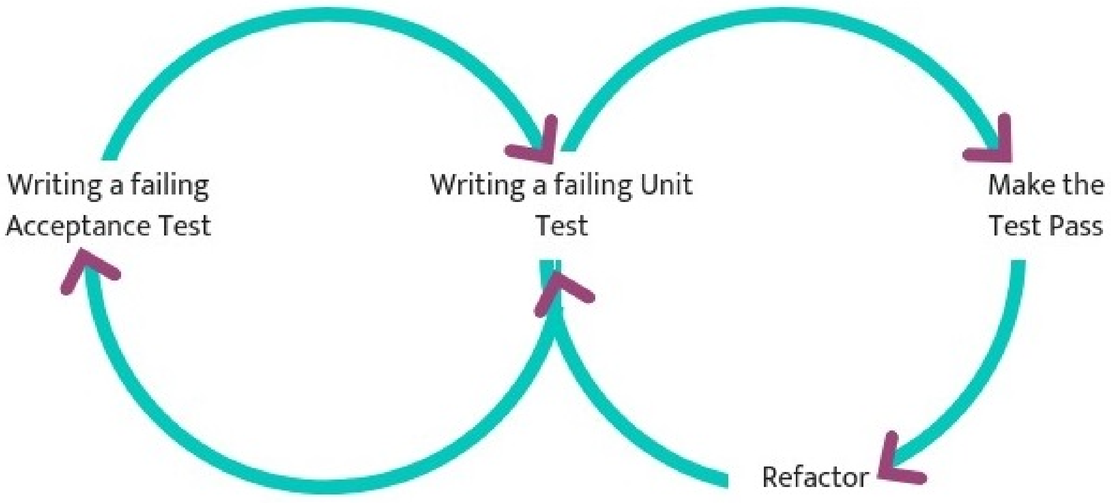

Writing a failing Acceptance Test
Writing a failing Unit Test
Make the Test Pass
Refactor

Figure 11: Acceptance Test Driven Development.

Best practices for testers using an ATDD methodology include:

- Interact directly with customers to align expectations, for example, through focus groups
- Involve customer-facing team members to understand customer needs, including customer service agents, sales representatives, and account managers
- Develop acceptance criteria according to customer expectations
- Prioritize two questions:
    1. How should we validate that the system performs a certain function?
    2. Will customers want to use the system when it has this function?

4. **Exploratory Testing**
   This test method simulates closely how clients will use the software in the real world. The testers essentially "play with" the software in a chaotic way to find ways to break the software (refer to figure 12 below). There is no detailed or structured process to do the tests but when a bug is found, it is documented as per normal bug reports.

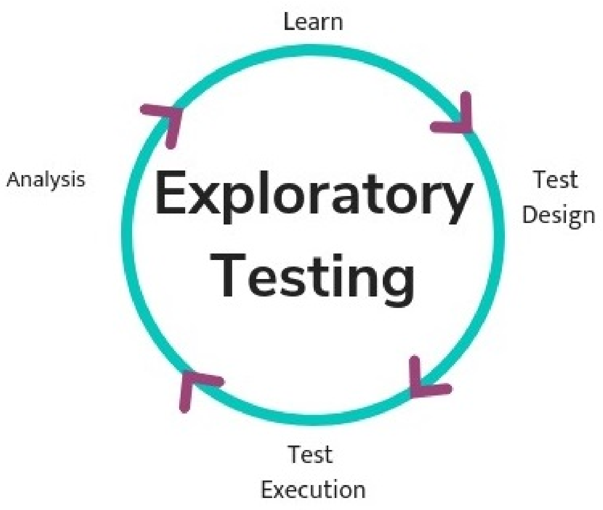


Figure 12: Exploratory Testing.

Best practices for exploratory testing:

- Organize functionality in the application, using a spreadsheet, mind map etc.
- Even though there is no detailed documentation of how tests were conducted, track which software areas were or were not covered with exploratory testing
- Focus on areas and scenarios in the software which are at high risk or have high value for users
- Ensure testers document their results so they can be accountable for areas of software they tested

5. **Session-Based Testing**
   A method similar to exploratory testing but with some order. Tests are structured into time sessions where the tests are conducted against a specific objective. The testers are still required to document their findings during each session. A debrief session is conducted between the tester/s and the manager or developers at the end of each session, covering the five **PROOF** points:

   1. What was done in the test (**P**ast)
   2. What the tester discovered or achieved (**R**esults)
   3. Any problems that got in the way (**O**bstacles)
   4. Remaining areas to be tested (**O**utlook)
   5. How the tester feels about the areas of the product they tested (**F**eelings)

   Best practices for session-based testing include:

   - Define a goal so testers are clear about priorities of testing in the current sprint
   - Develop a charter that states areas of the software to test, when the session will occur and for how long, which testers will conduct the session, etc.
   - Run uninterrupted testing sessions with a fixed, predefined length
   - Document activities, notes, and also takeaways from the face-to-face brief in a session report

Agile Testing Quadrants

With all the different Agile Principles and Testing Methodologies, one might be thinking how, when and which tests to be executed. These questions can be simplified using a system of quadrants that provide a classification for the tests (refer to figure 13 below).

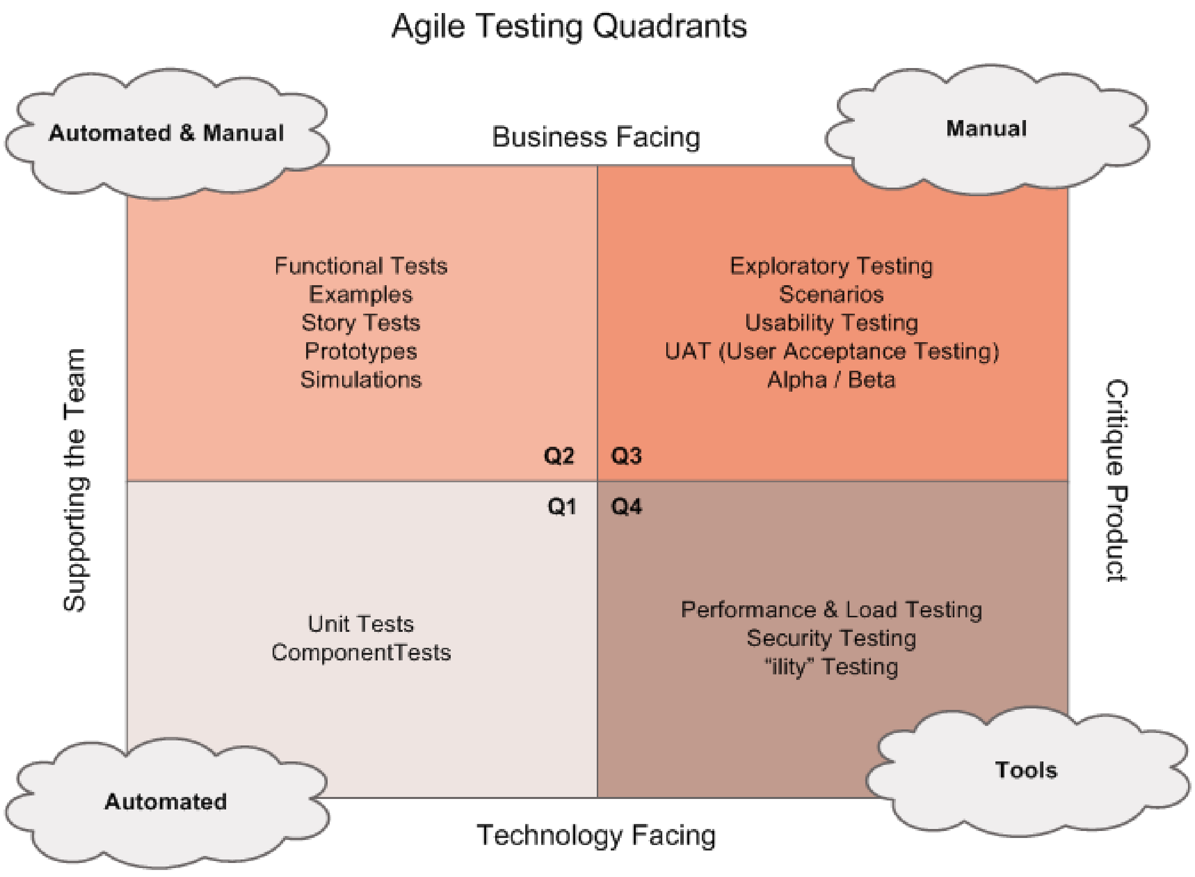


Figure 13: Agile Testing Quadrants.

From figure 13 above, the quadrants numbers on the Agile Testing Quadrants are solely for labeling and ease of description, it does not imply an execution order and they can be described in the following:

- **Quadrant 1** - Tests that relate to code quality, normally uses the Test-Driven Development (TDD) method of writing tests to help programmers design their code well.
- **Quadrant 2** - Tests that focus on the external quality and the features of the product that the customers want. These tests are normally derived from examples of product usage from the customer teams.
- **Quadrant 3** - Tests that try to emulate how a real user would use the product. Tests here provides feedback for tests in quadrants 1 and 2. Serious bugs are usually found from the results of these tests.
- **Quadrant 4** - Tests are intended to critique the product's characteristics such as security, backward compatibility, robustness, performance, etc.

Developing a testing strategy based on the four quadrants involves the whole team to identify the types of tests to use, at which point in the product roadmap to use it, having the right tools, data and people to perform the tests and talking to customers about usage and quality criteria.

Depending on the project, most would start at quadrant 2, because those are where the customer's usage examples get turn into specifications and tests that drive coding, along with prototypes and etc. However, there are also projects that starts with performance testing (which is in quadrant 4) such as testing the architecture or when customers are uncertain about their requirements, developers may suggest exploratory testing (quadrant 3).

Although quadrant 3 and quadrant 4 requires that some code has already been written and is deployable, it is still applicable as teams iterate through the quadrants rapidly by working in small increments. Test automation is also important to make sure that all the previously implemented and tested features are not broken by the implementation of new features. Depending on the project's platform, there are several test frameworks that departments can use such as Selenium for web based applications, PyTest for complex functional testing of applications and libraries or Cucumber for BDD testing.

There is no hard and fast rule in Agile to develop a test plan but there are ways to outline an Agile test strategy. The test strategy could be in the form of a document, a test matrix or even a Kanban board. No matter which Agile test methodology is chosen, the test strategy should include:

- Purpose (defined by the user perspective of a feature, also called the user story).
- Objectives (test cases).
- Scope (what needs to be tested).
- Methods (how tests will be run).

References

1. "Agile Essentials", Agile Alliance, https://www.agilealliance.org/agile-essentials/
2. I Sacolick, 2020, What is agile methodology? Modern software development explained, InfoWorld, https://www.infoworld.com/article/3237508/what-is-agile-methodology-modern-software-development-explained.html
3. "Agile Vs Waterfall: Know the Difference Between Methodologies", Guru99, https://www.guru99.com/waterfall-vs-agile.html
4. "The 2020 Scrum Guide", Scrum Guides, https://www.scrumguides.org/scrum-guide.html
5. "Agile Methodology: What is Agile Software Development Model?", Guru99, https://www.guru99.com/agile-scrum-extreme-testing.html
6. "Scrum (software development)", Wikipedia, https://en.wikipedia.org/wiki/Scrum_(software_development)
7. "Kanban (development)", Wikipedia, https://en.wikipedia.org/wiki/Kanban_(development)
8. "Kanban", Atlassian Agile Coach, https://www.atlassian.com/agile/kanban
9. "Monitoring work in a Kanban project", Jira Software Support, https://confluence.atlassian.com/jirasoftwareserver073/monitoring-work-in-a-kanban-project-861254594.html
10. "Scrumban", ProductPlan, https://www.productplan.com/glossary/scrumban/
11. "Scrumban", Wikipedia, https://en.wikipedia.org/wiki/Scrumban
12. "Agile Testing - Overview", Tutorials Point, https://www.tutorialspoint.com/agile_testing/agile_testing_overview.htm
13. "Agile Testing – Principles, methods & advantages", ReQtest, https://reqtest.com/testing-blog/agile-testing-principles-methods-advantages/
14. "Agile Testing: 8 Principles, 7 Challenges and How to Master Them", Sealights, https://www.sealights.io/software-development-metrics/agile-testing-8-principles-7-challenges-and-how-to-master-them/
15. "Agile Methodology: The Complete Guide to Understanding Agile Testing", QASymphony, https://www.qasymphony.com/blog/agile-methodology-guide-agile-testing/
16. L Crispin, "Using the Agile Testing Quadrants", Agile Testing with Lisa Crispin, https://lisacrispin.com/2011/11/08/using-the-agiletesting-quadrants/