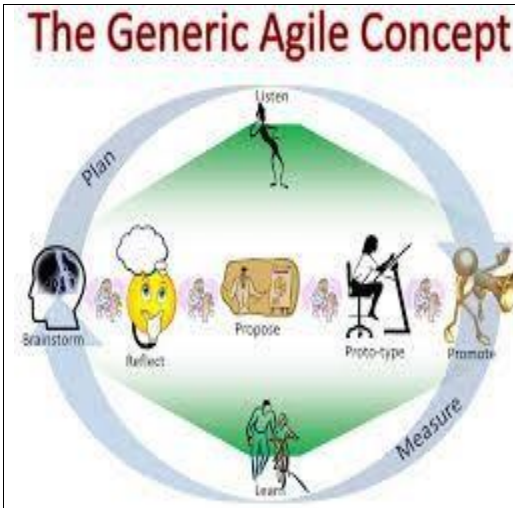
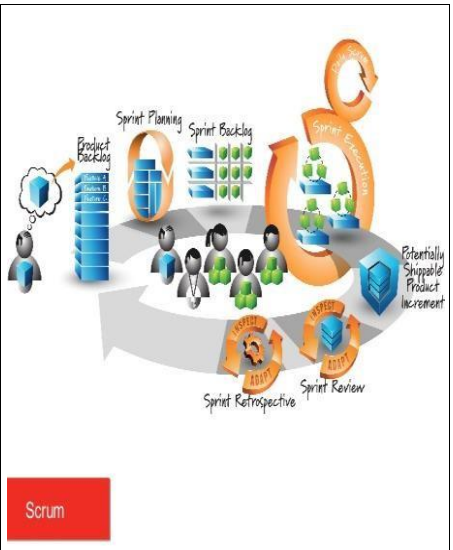
Department of Computer Sci.&Engineering

LAB MANUAL

B.TECH(CSE)/M.TECH(S.E)

AGILE DEVELOPMENT PROCESS





VIT AP
AMARAVATI

List of Practical's

1. **Understand the background and driving forces for taking an Agile Approach to Software Development.**
2. **Understand the business value of adopting agile approach.**
3. **Understand agile development practices**
4. **Propose a sample project with its Objectives, Vision statement, use cases and UML diagrams.**
5. **Compile Project release map, user stories for proposed project.**
6. **Identify the story-boarding tasks and detailed release plan for the project.**
7. **Design Product road-map for the proposed project.**
8. **Design Story-mapping for the proposed project.**
9. **Compile the Product-backlog for the Proposed project.**
10. **Compile the Sprint-Backlog for the Proposed Project.**
11. **Demonstration of the project with its Sprint-1, Sprint-2 releases...**
12. **Final Demonstration of the entire project as per proposal.**

Practical No. 1

Understand the background and driving forces for taking an Agile Approach to Software Development

Objective

1. Difference between agile software development model and waterfall model.
2. Why Agile is better?
3. Understanding the Agile Manifesto
4. Discussing Important Characteristics that make agile approach best suited for Software Development.

Theory

Agile software development is a group of software development methods in which requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. It promotes adaptive planning, evolutionary development, early delivery, continuous improvement, and encourages rapid and flexible response to change.

The *Manifesto for Agile Software Development*, also known as the *Agile Manifesto*, first introduced the term *agile* in the context of software development in 2001.

Manifesto for Agile Software Development

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.

Agile principles

The Agile Manifesto is based on 12 principles:

1. Customer satisfaction by rapid delivery of useful software
2. Welcome changing requirements, even late in development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the principal measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential

11. Self-organizing teams
12. Regular adaptation to changing circumstance

What's wrong with Traditional Approaches?

In 1970, Dr. Winston Royce presented a paper entitled "Managing the Development of Large Software Systems," which criticized sequential development. He asserted that software should not be developed like an automobile on an assembly line, in which each piece is added in sequential phases, each phase depending on the previous. Dr. Royce recommended against the phase based approach in which developers first gather all of a project's requirements, then complete all of its architecture and design, then write all of the code, and so on. Royce specifically objected to the lack of communication between the specialized groups that complete each phase of work.

It's easy to see the problems with the waterfall method. It assumes that every requirement can be identified before any design or coding occurs. Could you tell a team of developers everything that needed to be in a software product before any of it was up and running? Or would it be easier to describe your vision to the team if you could react to functional software? Many software developers have learned the answer to that question the hard way: At the end of a project, a team might have built the software it was asked to build, but, in the time it took to create, business realities have changed so dramatically that the product is irrelevant. Your company has spent time and money to create software that no one wants. Couldn't it have been possible to ensure the end product would still be relevant before it was actually finished?

Today very few organizations openly admit to doing waterfall or traditional command and control. But those habits persist.

Why Agile?

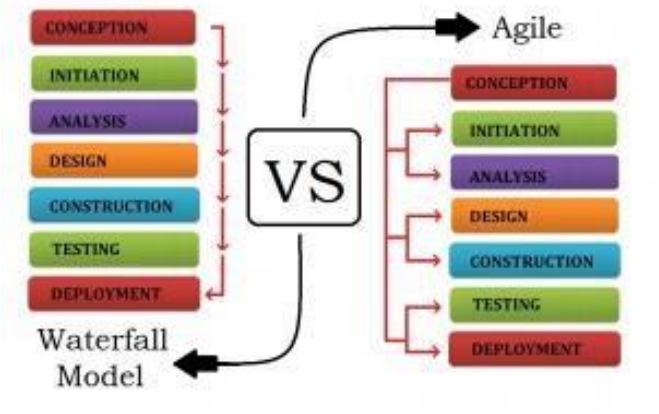
Agile development provides opportunities to assess the direction throughout the development lifecycle. This is achieved through regular cadences of work, known as Sprints or iterations, at the end of which teams must present a potentially shippable product increment. By focusing on the repetition of abbreviated work cycles as well as the functional product they yield, agile methodology is described as "iterative" and "incremental." In waterfall, development teams only have one chance to get each aspect of a project right. In an agile paradigm, every aspect of development — requirements, design, etc. — is continually revisited. When a team stops and re- evaluates the direction of a project every two weeks, there's time to steer it in another direction.

This "inspect-and-adapt" approach to development reduces development costs and time to market. Because teams can develop software at the same time they're gathering requirements, "analysis paralysis" is less likely to impede a team from making progress. And because a team's work cycle is limited to two weeks, stakeholders have recurring opportunities to calibrate releases for success in the real world. Agile development helps companies build the right product. Instead of committing to market a piece of software that hasn't been written yet, agile empowers teams to continuously replan their release to optimize its value throughout development, allowing them to be as competitive as possible in the marketplace. Agile development preserves a product's critical market relevance and ensures a team's work doesn't wind up on a shelf, never released.

Difference between agile software development model and waterfall model

It is worth mentioning here that the Waterfall model is the primitive model type and has been implemented in the development phase time after time. Hence in the due course if time developers found many drawbacks in this model which were later

rectified to form various other development models.



Advantages of the Agile Methodology

1. The Agile methodology allows for changes to be made after the initial planning. Re-writes to the program, as the client decides to make changes, are expected.

2. Because the Agile methodology allows you to make changes, it's easier to add features that will keep you up to date with the latest developments in your industry.

3. At the end of each sprint, project priorities are evaluated. This allows clients to add their feedback so that they ultimately get the product they desire.

4. The testing at the end of each sprint ensures that the bugs are caught and taken care of in the development cycle. They won't be found at the end.

5. Because the products are tested so thoroughly with Agile, the product could be launched at the end of any cycle. As a result, it's more likely to reach its launch date.

Conclusion

Both the Agile and waterfall methodologies have their strengths and weaknesses. The key to deciding which is right for you comes down to the context of the project. Is it going to be changing rapidly? If so, choose Agile. Do you know exactly what you need? Good. Then maybe waterfall is the better option. Or better yet? Consider taking aspects of both methodologies and combining them in order to make the best possible software development process for your project.

Practical No. 2

Understand the business value of adopting Agile approaches

Objective

1. Understanding the business values
2. Adopting Agile Software Development: Issues and Challenges
3. Overview of Scrum, Extreme Programming, Feature Driven development, Lean Software Development, Agile project management

Theory

When it comes to creating custom applications, too many of us live in denial. We want to believe that it's possible to predict accurately how long a group of developers will take to build software that meets our requirements. We also want to believe that we can know what those requirements are up front, before we've seen any part of the application, and that the requirements won't change during development. Sadly, none of these things are true for most projects. We can't predict how long development will take, largely because we can't get the requirements right up front and we can't stop them from changing. Because we deny these realities, many organizations still use software development processes that don't work well. Fortunately, this is changing. Agile development processes get more popular every day, primarily because they're rooted in reality: They're designed to accommodate change. Doing software development in this way can be scary at first, especially for the business leaders who are footing the bill. This needn't be the case, however. The truth is that agile processes are usually better both for development teams and for the business people who pay them. To understand why this is true, we need to start by understanding what an agile process really is. What Agile Development Means The challenge is always the same: We need to create software in the face of uncertainty. At the start of a development project, we don't know whether we've defined the project's requirements correctly. We also don't know how those requirements will change while we're building the software. A traditional development process does its best to pretend this uncertainty doesn't exist. In the classic waterfall approach, for example, an organization creates detailed plans and precise schedules before writing any code. But real development projects rarely comply with these plans and schedules—they're notoriously unruly. The core reason for this is that even though we use the term "software engineering", writing code isn't like other kinds of engineering. In traditional engineering projects—building a bridge, say, or constructing a factory—it's usually possible to define stable requirements up front. Once you've done this, creating plans and schedules based on previous experience is straightforward. Software development just isn't like this1 . Creating stable requirements up front is usually impossible, in part because people don't know what they want until they see it. And since every development project involves some innovation—if it doesn't, you should be buying rather than building the software—uncertainty is unavoidable. Traditional development processes work against these realities. Agile processes, however, are designed for this situation. Because requirements change, an agile process provides a way to add, remove, and modify requirements during the development process. Rather than resisting change, an agile process embraces it. Just as important, the process recognizes that short-term plans are more stable than long-term plans. You might not know exactly what you want to be doing three months from now, but you probably do

know what you want to do in the next three weeks. To accomplish this, an agile development process relies on iteration. Rather than the traditional approach of defining all of the requirements, writing all of the code, then testing that code, an agile process does these things over and over in smaller iterations. Each iteration creates more of the finished product, with the requirements updated at the start of each one.

Challenges in adopting agile Methodology

Challenge 1: Missing the Agile Master Role Agile master or Agile coach is an essential role during Agile adopting process in any organization. Agile coach is considered a consultant for the team in every step of a project using any Agile method, such as Scrum, that is responsible of providing guidance and helps to succeed in adopting Agile. Entity"S" management recognized the need to hire a contractor as an agile master. However the position was not filled due to financial constraints.

Challenge 2: The overzealous teams after attending a course on Agile methods, many of entity "S" teams wanted to adopt Agile methods as soon as possible hoping it will solve all their previous development challenges known for traditional methods. This overzealous team fast adoption of Agile resulted in a decrease in productivity because the development cycle took longer time due to many mistakes in implementation. This decrease in productivity led many team members to be less optimistic and started to lose interest in agile methods.

Challenge 3: The absent of a Pilot Project Another challenge is the absent of a pilot project in the transition from the previous traditional method to the scrum method. Conducting a pilot project was a recommended step in the adoption of agile development for the first time. As a part of the plan to adopt agile method, the pilot project is essential to evaluate how"S" environment will be able to move from the previous heavy-weight method to a new light method. Many organizations went through the same experience of running a pilot project especially those companies that have large projects such as Amazon, Yahoo, Microsoft and Intel. After investing the needed time and resources they have reached to a successful adoption of Agile.

Challenge 4: Scrum Implementation International Journal of Managing Value and Supply Chains (IJMVSC) Vol. 2, No. 3, September 2011 7 Although the employees in "S" were very experienced but yet none of them had any previous experience with agile development methods or Scrum implementation in particular. This is in addition of the absence of the agile master. For the team members, scrum implementation was not easy as it appeared to be during the training session. The team members find themselves, suddenly, in a completely new set up. The experience of traditional methods is completely different than committing to daily meeting, working with time boxes, finishing tasks in small period iteration and documenting the stories (or backlogs) in a different way.

Challenge 5: Current Work Pressure Although"S" software development team serves a very large organization of over 30 departments and developed numerous projects through the years, the development projects require continues maintenance and support. In addition, the team was working on a new project with firm deadlines. The work environment was very demanding and team worked under pressure to produce products according to the planned schedule. Scrum adoption process started while every member of the team was engaged in his/her everyday tasks. With such work pressure the daily Scrum meetings were considered waste of time and added extra pressure to the employees. They used to meet weekly and later twice a month and then only when required and usually after working hours. As teams started to skip

daily meetings it also affected the learning process of scrum between the team members. That eventually leads to the failure of learning and implementing agile method correctly.

Challenge 6: Upper Management Concerns The upper management of"S" had many concerns about the effectiveness and success of the transition to a new method. They were not easily convinced to invest in a new method.

Challenge 7: Governmental bureaucratic System The traditional method currently in "S" was customized to comply with the governmental system of other department. The new Agile method being introduced, Scrum, is developed in such highly bureaucratic environment. The Agile team has to secure approvals and signatures before moving from one step to another. This was perceived by the team members as unnecessary and more time was taken into account to develop a new project. The scrum method requires much less correspondence, less time in communication between the customer and the team and requires significantly less paper work and approvals as the customer is supposed to be involved in every step.

Challenge 8: Documentation requirements after years and years of extensive documentation of every step in the traditional method, moving to a new method with minimum documentation requirements was one of the greatest challenges. Every project used to end up with dozens of document such as project charter, project plan, testing plan, SRS, STS, technical documents, user manual, etc. Each of these document contained large number of pages written by every member of the team and consumed hours of the valuable development time. The documentation requirements were driven basically from the previous challenge (the governmental system), upper management, ISO certificate requirements and the traditional development method that is currently used. Although agile development promises sufficient documentation of the projects, it didn't seem very convincing to the upper management when they end up receiving few documents in comparison with the previous model of documentation. Many attempts were made to try to balance between the upper management requirement regarding documentation and between adopting Scrum method. Agile teams started to increase the number of documents required for documentation and started to customize Scrum as much as possible to conform to all the upper management requirements of documentation norms. This did not work very well and it created extra burden on the agile teams

Agile Process Examples

1. **SCRUM**
2. **FDD**
3. **Lean software development**
4. **XP**

SCRUM

Scrum is an iterative and incremental agile software development methodology for managing product development. It defines "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal", challenges assumptions of the "traditional, sequential approach" to product development, and enables teams to self-organize by encouraging physical co-location or close online collaboration of all team members, as well as daily face-to-face communication among all team members and disciplines in the project.

FDD

Feature-driven development (**FDD**) is an iterative and incremental software development process. It is one of a number of lightweight or agile methods for developing software. FDD blends a number of industry-recognized best practices into a cohesive whole. These practices are all driven from a client-valued functionality (feature) perspective. Its main purpose is to deliver tangible, working software repeatedly in a timely manner.

Lean software development

Lean software development (**LSD**) is a translation of lean manufacturing and lean IT principles and practices to the software development domain. Adapted from the Toyota Production System,[1] a pro-lean subculture is emerging from within the Agile community. Lean is most popular with startups that want to penetrate the market, or test their idea and see if it would make a viable business.

XP

Extreme programming (**XP**) is a software development methodology which is intended to improve software quality and responsiveness to changing customer requirements. As a type of agile software development, it advocates frequent "releases" in short development cycles, which is intended to improve productivity and introduce checkpoints at which new customer requirements can be adopted.

Other elements of extreme programming include: programming in pairs or doing extensive code review, unit testing of all code, avoiding programming of features until they are actually needed, a flat management structure, simplicity and clarity in code, expecting changes in the customer's requirements as time passes and the problem is better understood, and frequent communication with the customer and among programmers.The methodology takes its name from the idea that the beneficial elements of traditional software engineering practices are taken to "extreme" levels. As an example, code reviews are considered a beneficial practice; taken to the extreme, code can be reviewed *continuously*, i.e. the practice of pair programming.

Critics have noted several potential drawbacks,[5] including problems with unstable requirements, no documented compromises of user conflicts, and a lack of an overall design specification or document.

Conclusion

Each agile methodology has a slightly different approach for implementing the core values from the Agile Manifesto, just as many computer languages manifest the core features of object-oriented programming in different ways. A recent survey shows that about 50 percent of agile practitioners say that their team is doing Scrum. Another 20 percent say that they are doing Scrum with XP components. An additional 12 percent say that they are doing XP alone. Because more than 80 percent of agile implementations worldwide are Scrum or XP, MSF for Agile Software Development v5.0 focuses on the core processes and practices of Scrum and XP

Practical No. 3

Understand the Agile development practices

Objective

Understanding SCRUM

Theory

Scrum is an iterative and incremental agile software development methodology for managing product development. It defines "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal", challenges assumptions of the "traditional, sequential approach" to product development, and enables teams to self-organize by encouraging physical co-location or close online collaboration of all team members, as well as daily face-to-face communication among all team members and disciplines in the project.

A key principle of scrum is its recognition that during a project the customers can change their minds about what they want and need (often called "requirements churn"), and that unpredicted challenges cannot be easily addressed in a traditional predictive or planned manner. As such, scrum adopts an empirical approach— accepting that the problem cannot be fully understood or defined, focusing instead on maximizing the team's ability to deliver quickly and respond to emerging requirements.

History

Scrum was first defined as "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal" as opposed to a "traditional, sequential approach" in 1986 by Hirotaka Takeuchi and Ikujiro Nonaka in the *New New Product Development Game*. Takeuchi and Nonaka later argued in *The Knowledge Creating Company* that it is a form of "organizational knowledge creation, [...] especially good at bringing about innovation continuously, incrementally and spirally".

The authors described a new approach to commercial product development that would increase speed and flexibility, based on case studies from manufacturing firms in the automotive, photocopier and printer industries. They called this the *holistic* or *rugby approach*, as the whole process is performed by one cross-functional team across multiple overlapping phases, where the team "tries to go the distance as a unit, passing the ball back and forth". (In rugby football, a scrum refers to a tight-packed formation of players with their heads down who attempt to gain possession of the ball.)

In the early 1990s, Ken Schwaber used what would become scrum at his company, Advanced Development Methods, and Jeff Sutherland, with John Scumniotales and Jeff McKenna, developed a similar approach at Easel Corporation, and were the first to refer to it using the single word *scrum*. In 1995, Sutherland and Schwaber jointly presented a paper describing the *scrum methodology* at the Business Object Design and Implementation Workshop held as part of Object-Oriented Programming, Systems, Languages & Applications '95 (OOPSLA '95) in Austin, Texas, its first public presentation. Schwaber and Sutherland collaborated during the following years to merge the above writings, their experiences, and industry best practices into what is now known as scrum.

In 2001, Schwaber worked with Mike Beedle to describe the method in the book *Agile Software Development with Scrum*.Its approach to planning and managing projects is to bring decision-making authority to the level of operation properties and certainties. Although the word is not an acronym, some companies implementing the process have been known to spell it with capital letters as SCRUM. This may be due to one of Ken Schwaber's early papers, which capitalized SCRUM in the title.

Later, Schwaber with others founded the Scrum Alliance and created the Certified Scrum Master programs and its derivatives. Schwaber left the Scrum Alliance in the fall of 2009, and founded Scrum.org to further improve the quality and effectiveness of scrum.

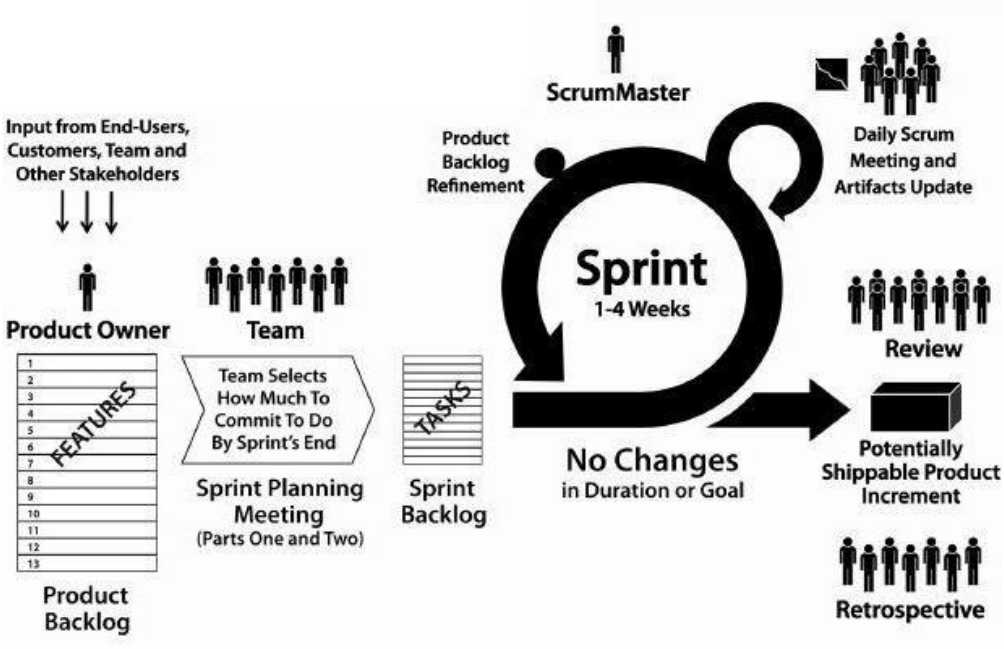


Figure: **SCRUM Process**

How Does Scrum Fit With Agile?

The Agile Manifesto doesn't provide concrete steps. Organizations usually seek more specific methods within the Agile movement. These include Crystal Clear, Extreme Programming, Feature Driven Development, Dynamic Systems Development Method (DSDM), Scrum, and others. While I like all the Agile approaches, for my own team Scrum was the one that enabled our initial breakthroughs. Scrum's simple definitions gave our team the autonomy we needed to do our best work while helping our boss (who became our *Product Owner*) get the business results he wanted. Scrum opened our door to other useful Agile practices such as test-driven development (TDD). Since then we've helped businesses around the world use Scrum to become more agile. A truly agile enterprise would not have a "business side" and a "technical side." It would have teams working directly on delivering business value. We get the best results when we involve the whole business in this, so those are the types of engagements I'm personally the most interested in.

What's The Philosophy behind Scrum?

Scrum's early advocates were inspired by empirical *inspect and adapt* feedback loops to cope with complexity and risk. Scrum emphasizes decision making from real-world results rather than speculation. Time is divided into short work cadences, known as sprints, typically one week or two weeks long. The product is kept in a potentially shippable (properly integrated and tested) state at all times. At the end of each sprint, stakeholders and team members meet to see a demonstrated potentially shippable product increment and plan its next steps.

Scrum is a simple set of roles, responsibilities, and meetings that never change. By removing unnecessary unpredictability, we're better able to cope with the necessary unpredictability of continuous discovery and learning.

Scrum Roles

- **Product Owner:** The Product Owner should be a person with vision, authority, and availability. The Product Owner is responsible for continuously communicating the vision and priorities to the development team.

  It's sometimes hard for Product Owners to strike the right balance of involvement. Because Scrum values self-organization among teams, a Product Owner must fight the urge to micro-manage. At the same time, Product Owners must be available to answer questions from the team.

- **Scrum Master:** The Scrum Master acts as a facilitator for the Product Owner and the team. The Scrum Master does not manage the team. The Scrum Master works to remove any impediments that are obstructing the team from achieving its sprint goals. This helps the team remain creative and productive while making sure its successes are visible to the Product Owner. The Scrum Master also works to advise the Product Owner about how to maximize ROI for the team.
- **Team:** According to Scrum's founder, "the team is utterly self managing." The development team is responsible for self organizing to complete work. A Scrum development team contains about seven fully dedicated members (officially 3-9), ideally in one team room protected from outside distractions. For software projects, a typical team includes a mix of software engineers, architects, programmers, analysts, QA experts, testers, and UI designers. Each sprint, the team is responsible for determining how it will accomplish the work to be completed. The team has autonomy and responsibility to meet the goals of the sprint.

Scrum Meetings

There are number of regular meetings take place among Agile Team members. Let's go through different type of meetings.

1. Sprint Planning Meeting

During a sprint team works on selected features. Those features are planned and selected in sprint planning meeting which holds before every sprint. Product owner and scrum team participate this planning session which usually last for 4 hours.

Product owner come up with a set of features to discuss with intent to add into sprint backlog so team could work on it during the sprint. Usually those features have a specific theme attach to them, that theme is also called sprint goal.

Following are a few example of the sprint goal.

- Improve the performance of the current system.
- Build the UI of certain component.
- Create API of the product so other application could access some specific information.

Based on input from the team, features are finalized for the sprint. Team input helps to decide if proposed features can be completed in the sprint or there are any impediments involve in completing those features. Features story points should be fall within sprint velocity. Once features are decided scrum team breaks those features into tasks. Team may assign hours or story point to each task. The output of the meeting is a sprint backlog, sprint theme/goal and tasks related to each feature.

Timings: Before every Sprint

Duration: 4-8 hours

Participants: Scrum Owner, Scrum master, Scrum Team

Artifacts: Sprint Tag/Goal, sprint backlog, tasks

2. Release Planning Meeting

Purpose of the meeting

An Agile project may consist of number of iterations (sprints). A release consists of a sub set of such iterations. The length of release could be between 2 to 6 months. Release planning meeting is held at the beginning of each release. The purpose of the release planning meeting is to go through the backlog and give an estimate the number and set the priority of features that can be completed in a release. At the start of the meeting Product owner usually come up with the backlog with selected features he wants to get completed during the release time frame. Scrum team gives its opinion regarding the features and based on the discussion certain features are added or removed from the release, or their priority and time to complete may change. The technical knowledge of the team gets very helpful in this planning session. The scrum team may prefer to complete certain issues before than others because of architecture design or some other technical issues that product owner does not have much knowledge about. An important measure which plays a vital role in release planning and estimation meeting is Velocity. If it's not the first release, velocity of previous releases can be helpful in finding out number of sprint may take to complete a release. If the velocity is not known then team have to estimate the work that can be completed in a release.

Initial estimate of the team can go wrong. In those situations team can hold another planning meeting to adjust the estimates based on real progress. With time velocity gets more predictable the release planning meeting start giving more accurate estimates based on velocity.

Timings: Beginning of each release or revised during the release

Duration: 4 – 16 hours

Participants: product owner, scrum master, scrum team

Best Practices:

- Project owner come with prioritized and estimated features, so that team could create a schedule for the release.
- Use the velocity to determine the number of sprints.
- Don't dictate technical team with your technical choices.
- Don't try to push features against the team recommendations.
- If teams are very big than one person representing each area should attend the meeting.

Artifacts: Release backlog.

3. Review Meetings

a. Sprint Review (Demo) Meeting

Purpose of the meeting

Once sprint is completed Sprint team present a demonstration of the features completed during the sprint. The Product owner and customer review the features and give their comments on the completed features. Product owner may accept or reject the feature or point out the deficiency or bug in the work completed during the sprint. Features that fulfill the definition of done, provided by the product owner, are accepted. The fate of those features that are not completed or partially completed decided during next sprint planning meeting.

Timings: After sprint completion.

Duration: 2 hours

Participants: Product owner, customer, scrum team

Artifacts: completed features.

b. Sprint Retrospective Meeting

Purpose of the meeting

The purpose of Retrospective meeting is to discuss with team what processes or practices went well during the sprint? and what process and practices need to be improved? In this meeting only scrum team and scrum team participate. This is the last meeting of the sprint.

Following are some of the examples.

- Smaller task work better than larger ones
- Definition of done need to be more clear
- Information radiators need to be placed where everyone could see them
- Daily meeting should not exceed 15 minutes.

Timings: After Sprint Review meeting

Duration: 2-4 hours

Participants: Scrum Team, Scrum Master

Best Practices: Following are some of the best practices we can follow.

- Use time line of the sprint to get feed back
- Make team comfortable and relax
- Try to let team to reach a consensus without scrum master interuptions
- Don't try to cover events outside of the sprint

Artifacts: better processes and practices.

4. Daily Stand-up Meeting

Purpose of the meeting

The Daily stand-up or daily scrum meeting is the most important scrum meeting. The purpose of the meeting is to understand what team members are working on and propose them a solution for any impediment they may be facing. Each scrum team member answer following three questions in each of the meeting.

What you have done since last meeting?

What, if anything, is preventing you to perform your task?

What you will do between now and next meeting?

Timings: Daily

Duration: 15 minutes

Participants: Scrum master, scrum team (other can participate as a silent participants)

Best Practices: Following are some of the best practices we can follow.

- Don't exceed more than 15 minutes.
- Only scrum team speaks, product owner can attend the meeting as silence participant
- Fix the time for the meeting.
- Don't discuss anything out of three questions mentioned above.

Artifacts

Product backlog

The *product backlog* comprises an ordered list of *requirements* that a scrum team maintains for a product. It consists of features, bug fixes, non-functional requirements, etc.—whatever needs doing in order to successfully deliver a viable product. The product owner orders the *product backlog items* (PBIs) based on considerations such as risk, business value, dependencies, and date needed.

Items added to a backlog are commonly written in story format. The product backlog is *what* will be delivered, ordered into the sequence in which it should be delivered. It is open and editable by anyone, but the product owner is ultimately responsible for ordering the items on the backlog for the development team to choose.

The product backlog contains the product owner's assessment of business value and the development team's assessment of development effort, which are often, but not always, stated in story points using a rounded Fibonacci sequence. These estimates help the product owner to gauge the timeline and may influence ordering of backlog items; for example, if the "add spellcheck" and "add table support" features have the same business value, the product owner may schedule earlier delivery of the one with the lower development effort (because the return on investment is higher) or the one with higher development effort (because it is more complex or riskier, and they want to retire that risk earlier).

The product backlog and the business value of each backlog item is the responsibility of the product owner. The size (i.e. estimated complexity or effort) of each backlog item is, however, determined by the development team, who contributes by sizing items, either in story points or in estimated hours.

There is a common misunderstanding that only user stories are allowed in a product backlog. By contrast, scrum is neutral on requirement techniques. As the *Scrum Primer* states,

Product Backlog items are articulated in any way that is clear and sustainable. Contrary to popular misunderstanding, the Product Backlog does not contain "user stories"; it simply contains items. Those items can be expressed as user stories, use cases, or any other requirements approach that the group finds useful. But whatever the approach, most items should focus on delivering value to customers.

Scrum advocates that the role of product owner be assigned. The product owner is responsible for maximizing the value of the product. The product owner gathers input and takes feedback from, and is lobbied by, many people, but ultimately makes the call on what gets built.

The product backlog is used to:

- capture requests for modifying a product. This can include adding new features, replacing old features, removing features and fixing issues
- ensure the development team is given work which maximizes the business benefit to the owner of the product

Typically, the product owner and the scrum team come together and write down everything that needs to be prioritized and this becomes content for the first sprint, which is a block of time meant for focused work on selected items that can be accommodated within a timeframe. The product backlog can evolve as new information surfaces about the product and about its customers, and so later sprints may address new work.

The following items typically comprise a scrum backlog: features, bugs, technical work, and knowledge acquisition. Web development can entail confusion as to the difference between a feature and a bug: technically a feature is "wanted", while a bug is a feature that is "unintended" or "unwanted" (but may not be necessarily a defective thing). An example of technical work would be: "run virus check on all developers' workstations". An example of knowledge acquisition could be a scrum backlog item about researching Wordpress plugin libraries and making a selection.

Managing the product backlog between product owner and scrum team

A backlog, in its simplest form, is merely a list of items to be worked on. Having well-established rules about how work is added, removed and ordered helps the whole team make better decisions about how to change the product.

The product owner prioritizes which of the items in the product backlog are most needed. The team then chooses which items they can complete in the coming sprint. On the scrum board, the team moves items from the product backlog to the sprint backlog, which is the list of items they will build. Conceptually, it is ideal for the team to only select what they think they can accomplish from the top of the list, but it is not unusual to see in practice that teams are able to take lower-priority items from the list along with the top ones selected. This normally happens because there is time left within the sprint to accommodate more work. Items at the top of the backlog, the items that are going to be worked on first, should be broken down into stories that are suitable for the development team to work on. The further down the backlog goes, the less refined the items should be. As Schwaber and Beedle put it "The lower the priority, the less detail, until you can barely make out the backlog item.

As the team works through the backlog, it needs to be assumed that "changes in the world can happen"—the team can learn 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. The world changes, the backlog is never finished.[12]

Sprint backlog

The *sprint backlog* is the list of work the development team must address during the next sprint. The list is derived by selecting product backlog items from the top of the product backlog until the development team feels it has enough work to fill the sprint. This is done by the development team asking "Can we also do this?" and adding product backlog items to the sprint backlog. The development team should keep in mind its past performance assessing its capacity for the new sprint, and use this as a guide line of how much "effort" they can complete.

The product backlog items are broken down into tasks by the development team. Tasks on the sprint backlog are never assigned; rather, tasks are signed up for by the team members as needed according to the set priority and the development team member skills. This promotes self-organization of the development team, and developer buy-in.

The sprint backlog is the property of the development team, and all included estimates are provided by the development team. Often an accompanying *task board* is used to see and change the state of the tasks of the current sprint, like "to do", "in progress" and "done".

Once a sprint backlog is committed, no additional functionality can be added to the sprint backlog except by the team. Once a sprint has been delivered, the product backlog is analyzed and reprioritized if necessary, and the next set of functionality is selected for the next sprint.

Product increment

The *increment* (or *potentially shippable increment*, PSI) is the sum of all the product backlog items completed during a sprint and all previous sprints. At the end of a sprint, the increment must be complete, according to the scrum team's Definition of Done (DoD), and in a usable condition regardless of whether the product owner decides to actually release it.

Sprint burn-down chart

A sample burn down chart for a completed iteration, showing remaining effort and tasks for each of the 21 work days of the 1-month iteration

The sprint burndown chart is a public displayed chart showing remaining work in the sprint backlog. Updated every day, it gives a simple view of the sprint progress. It also provides quick visualizations for reference. During sprint planning the ideal burndown chart is plotted. Then, during the sprint, each member picks up tasks from the sprint backlog and works on them. At the end of the day, they update the remaining hours for tasks to be completed. In such way the actual burndown chart is updated day by day.

The following terms are often used in a scrum process.

Sprint burn-down chart
Daily progress for a sprint over the sprint's length.

Release burn-down chart
Feature level progress of completed product backlog items in the product backlog.

Product backlog (PBL) list
A prioritized list of high-level requirements.

Sprint backlog (SBL) list
A prioritized list of tasks to be completed during the sprint.

Sprint
A time period (typically 1–4 weeks) in which development occurs on a set of backlog items that the team has committed to. Also commonly referred to as a time-box or iteration.

Spike
A time boxed period used to research a concept and/or create a simple prototype. Spikes can either be planned to take place in between sprints or, for larger teams, a spike might be accepted as one of many sprint delivery objectives. Spikes are often introduced before the delivery of large or complex product backlog items in order to secure budget, expand knowledge, and/or produce a proof of concept. The duration and objective(s) of a spike will be agreed between the product owner and development team before the start. Unlike sprint commitments, spikes may or may not deliver tangible, shippable, valuable functionality. For example, the objective of a spike might be to successfully reach a decision on a course of action. The spike is over when the time is up, not necessarily when the objective has been delivered.

Tasks
Work items added to the sprint backlog at the beginning of a sprint and broken down into hours. Each task should not exceed 12 hours (or two days), but it's common for teams to insist that a task take no more than a day to finish.

Definition of Done (DoD)

The exit-criteria to determine whether a product backlog item is complete. In many cases the DoD requires that all regression tests should be successful. The definition of "done" may vary from one scrum team to another, but must be consistent within one team.

Velocity

The total effort a team is capable of in a sprint. The number is derived by evaluating the work (typically in user story points) completed from the last sprint's backlog items. The collection of historical velocity data is a guideline for assisting the team in understanding how much work they can do in a future sprint.

Impediment

Anything that prevents a team member from performing work as efficiently as possible.

Conclusion

Scrum is an agile process most commonly used for product development, especially software development. Scrum is a project management framework that is applicable to any project with aggressive deadlines, complex requirements and a degree of uniqueness. In Scrum, projects move forward via a series of iterations called sprints. Each sprint is typically two to four weeks long.