Software Development Life Cycle (SDLC)

Revision Book

Jack Moon

Jack.Moon@ada.ac.uk

**Software Development Life Cycle Revision Contents:**

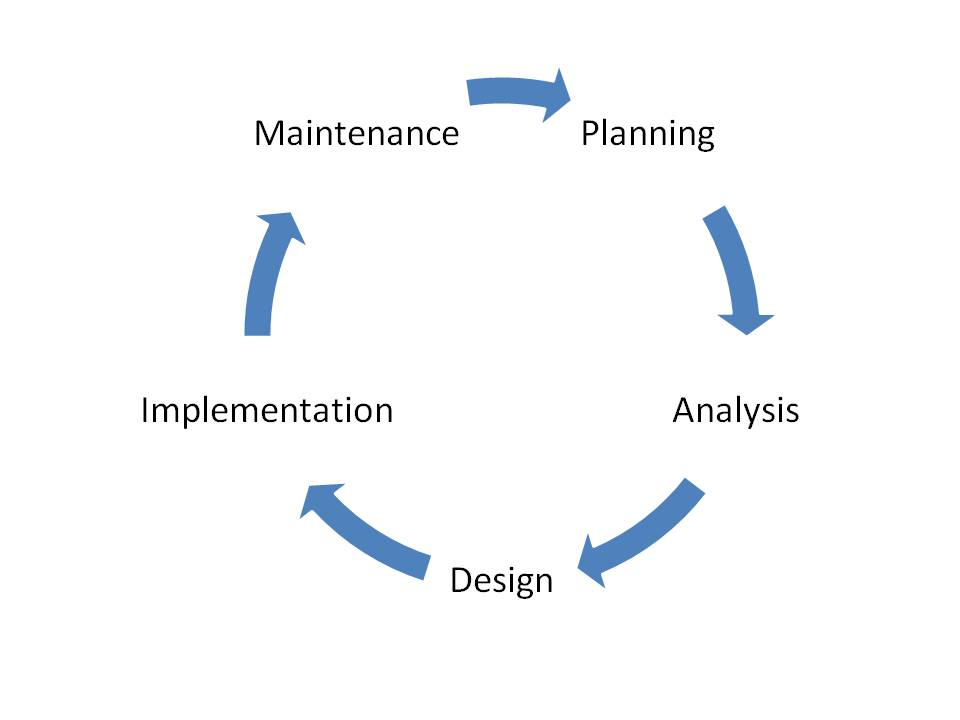
* Basic definition of SDLC
* The Phases of the SDLC
* Top 5 Software Development Life Cycle models
* Waterfall vs Agile
* The 7 stages of the SDLC
* Prototype model and phases
* Extreme programming and extreme prototyping
* Agile Manifesto
* Software testing
* Scrum
* User centred design

**Basic definition of SDLC**

SDLC is a process followed for a software project, within a software organisation. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The cycle defines a methodology for improving the quality of software and the overall development process.

SDLC serves as a guide to the project and provides a flexible and consistent medium to accommodate changes, and perform the project to meet the client’s objectives. SDLC phases define key schedule and delivery points which ensure timely and correct delivery to the client within budget and other constraints and project requirements. SDLC co-operates project control and management activities as they must be introduced within each phase of the SDLC.

**The phases of the SDLC**



The first phase is the planning phase and the objective of this phase is to develop a project charter (for above threshold projects), project management plan, and a project budget plan, where appropriate, that will lead to a successful project. The next phase of this cycle is analysis, the objective of this phase is to define in more detail the system inputs, processes, outputs and interfaces.

At the end of this phase the systems processes will be defined at the functional level, meaning the functions to be performed will be known, but not necessarily how they will be performed. Unless specifically constrained by the project Charter, Requirements analysis should not consider the computer programs, files and data streams.

Requirements analysis will identify and consider the risks related to how the technology will be integrated into the standard operating procedures. Requirements analysis will collect the functional and system requirements of the business process, the user requirements and the operational requirements.

The next phase is the design phase which is to transform business requirements identified during previous phases, into a detailed system architecture which is feasible, robust and brings value to the organisation. After this phase, the implementation phase occurs which objective is to install the system in the production environment and to bring it into operation; and second, to ensure that the system, as developed:

* Satisfies the functional requirements.
* Satisfies the business needs.
* Adheres to all mandates, physical constraints and service level agreements.
* Operates as described in the user and operator manuals.

The final phase of this model would be the maintenance phase and the objective of this phase would be to facilitate configuration, administration, and troubleshooting of common issues.

**Top 5 Software Development Life Cycle models**

An SDLC model defines implementation of an approach to the project. It defines the various processes, and phases that would be carried out throughout the project to produce the desired output. There are a variety of SDLC models that exist catering to different needs and characteristics of a project. Some are of iterative nature (Prototyping), whereas some are sequential (Waterfall). The top 5 models are: Waterfall model, Iterative model, Spiral model, V-Model, RAD model.

The waterfall Model illustrates the software development process in a linear sequential flow. This means that any phase in the development process begins only if the previous phase is complete. In this waterfall model, the phases do not overlap.

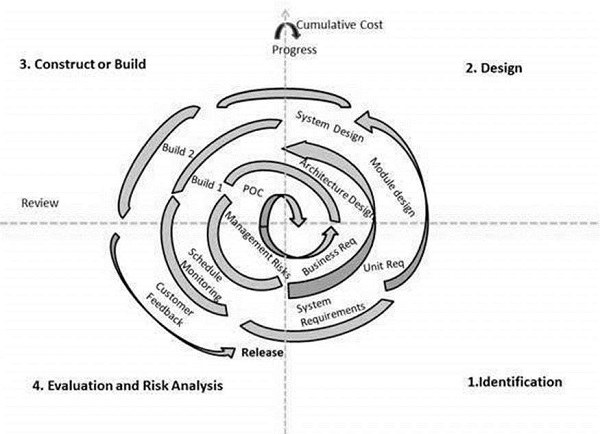


In the Iterative model, iterative process starts with a simple implementation of a small set of the software requirements and iteratively enhances the evolving versions until the complete system is implemented and ready to be deployed.

An iterative life cycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which is then reviewed to identify further requirements. This process is then repeated, producing a new version of the software at the end of each iteration of the model.

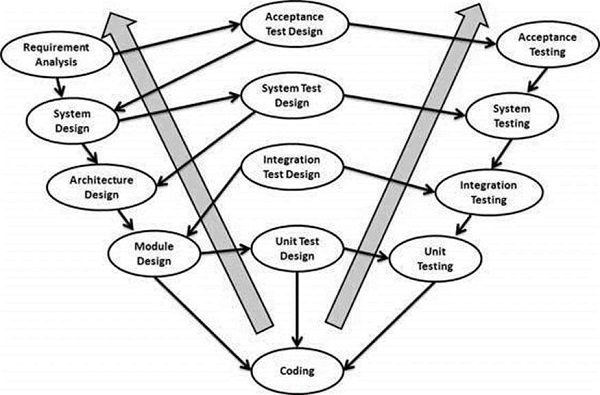


The spiral model combines the idea of iterative development with the systematic, controlled aspects of the waterfall model. This Spiral model is a combination of iterative development process model and sequential linear development model i.e. the waterfall model with a very high emphasis on risk analysis. It allows incremental releases of the product or incremental refinement through each iteration around the spiral.



The V-model is an SDLC model where execution of processes happens in a sequential manner in a V-shape. It is also known as Verification and Validation model.

The V-Model is an extension of the waterfall model and is based on the association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle, there is a directly associated testing phase. This is a highly-disciplined model and the next phase starts only after completion of the previous phase.



The RAD (Rapid Application Development) model is based on prototyping and iterative development with no specific planning involved. The process of writing the software itself involves the planning required for developing the product.

Rapid Application Development focuses on gathering customer requirements through workshops or focus groups, early testing of the prototypes by the customer using iterative concept, reuse of the existing prototypes (components), continuous integration and rapid delivery.



**Waterfall VS Agile**

Waterfall positives:

Developers and customers agree on what will be delivered early in the development lifecycle. This makes planning and designing more straightforward.

Progress is more easily measured, as the full scope of the work is known in advance.

Throughout the development effort, it’s possible for various members of the team to be involved or to continue with other work, depending on the active phase of the project. For example, business analysts can learn about and document what needs to be done, while the developers are working on other projects. Testers can prepare test scripts from requirements documentation while coding is underway.

Except for reviews, approvals, status meetings, etc., a customer presence is not strictly required after the requirements phase.

Because design is completed early in the development lifecycle, this approach lends itself to projects where multiple software components must be designed (sometimes in parallel) for integration with external systems.

Finally, the software can be designed completely and more carefully, based upon a more complete understanding of all software deliverables. This provides a better software design with less likelihood of the “piecemeal effect,” a development phenomenon that can occur as pieces of code are defined and subsequently added to an application where they may or may not fit well.

Waterfall negatives:

One area which almost always falls short is the effectiveness of requirements. Gathering and documenting requirements in a way that is meaningful to a customer is often the most difficult part of software development, in my opinion. Customers are sometimes intimidated by details, and specific details, provided early in the project, are required with this approach. In addition, customers are not always able to visualize an application from a requirements document. Wireframes and mock-ups can help, but there’s no question that most end users have some difficulty putting these elements together with written requirements to arrive at a good picture of what they will be getting.

Another potential drawback of pure Waterfall development is the possibility that the customer will be dissatisfied with their delivered software product. As all deliverables are based upon documented requirements, a customer may not see what will be delivered until it’s almost finished. By that time, changes can be difficult (and costly) to implement.

Agile positives:

The customer has frequent and early opportunities to see the work being delivered, and to make decisions and changes throughout the development project.

The customer gains a strong sense of ownership by working extensively and directly with the project team throughout the project.

If time to market for a specific application is a greater concern than releasing a full feature set at initial launch, Agile can more quickly produce a basic version of working software which can be built upon in successive iterations.

Development is often more user-focused, likely a result of more and frequent direction from the customer.

For more Agile Development benefits, please see 8 Benefits of Agile Software Development

Agile negatives:

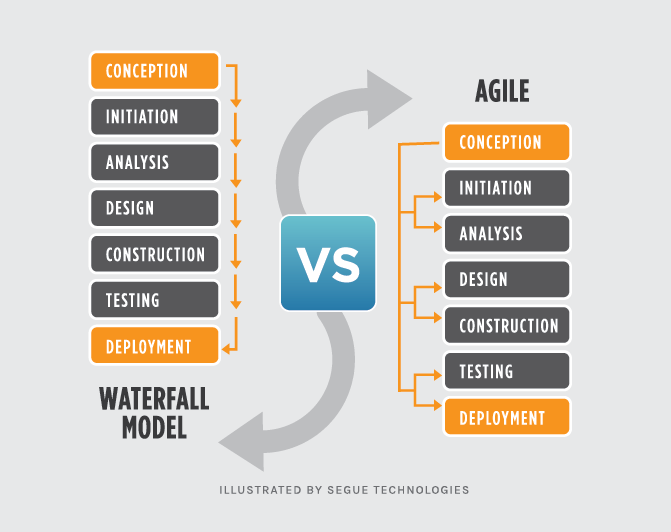
The very high degree of customer involvement, while great for the project, may present problems for some customers who simply may not have the time or interest for this type of participation.

Agile works best when members of the development team are completely dedicated to the project.

Because Agile focuses on time-boxed delivery and frequent reprioritization, it’s possible that some items set for delivery will not be completed within the allotted timeframe. Additional sprints (beyond those initially planned) may be needed, adding to the project cost. In addition, customer involvement often leads to additional features requested throughout the project. Again, this can add to the overall time and cost of the implementation.

The close working relationships in an Agile project are easiest to manage when the team members are in the same physical space, which is not always possible. However, there are a variety of ways to handle this issue, such as webcams, collaboration tools, etc.

The iterative nature of Agile development may lead to a frequent refactoring if the full scope of the system is not considered in the initial architecture and design. Without this refactoring, the system can suffer from a reduction in overall quality. This becomes more pronounced in larger-scale implementations, or with systems that include a high level of integration.



**The 7 stages of the SDLC**

1. Planning:

This is the first phase in the systems development process. It identifies whether there is the need for a new system to achieve a business’s strategic objectives. This is a preliminary plan (or a feasibility study) for a company’s business initiative to acquire the resources to build on an infrastructure to modify or improve a service. The company might be trying to meet or exceed expectations for their employees, customers and stakeholders too. The purpose of this step is to find out the scope of the problem and determine solutions. Resources, costs, time, benefits and other items should be considered at this stage.

2. Systems Analysis and Requirements:

The second phase is where businesses will work on the source of their problem or the need for a change. In the event of a problem, possible solutions are submitted and analysed to identify the best fit for the goal(s) of the project. This is where teams consider the functional requirements of the project or solution. It is also where system analysis takes place - or analysing the needs of the end users to ensure the new system can meet their expectations. Systems analysis is vital in determining what a business’s needs are, as well as how they can be met, who will be responsible for individual pieces of the project, and what sort of timeline should be expected.

3. Systems Design:

The third phase describes, in detail, the necessary specifications, features and operations that will satisfy the functional requirements of the proposed system which will be in place. This is the step for end users to discuss and determine their specific business information needs for the proposed system. It’s during this phase that they will consider the essential components (hardware and/or software) structure (networking capabilities), processing and procedures for the system to accomplish its objectives.

4. Development:

The fourth phase is when the real work begins—in particular, when a programmer, network engineer and/or database developer are brought on to do the major work on the project. This work includes using a flow chart to ensure that the process of the system is properly organized. The development phase marks the end of the initial section of the process. Additionally, this phase signifies the start of production. The development stage is also characterized by instillation and change. Focusing on training can be a huge benefit during this phase.

5. Integration and Testing:

The fifth phase involves systems integration and system testing (of programs and procedures)—normally carried out by a Quality Assurance (QA) professional—to determine if the proposed design meets the initial set of business goals. Testing may be repeated, specifically to check for errors, bugs and interoperability. This testing will be performed until the end user finds it acceptable. Another part of this phase is verification and validation, both of which will help ensure the program’s successful completion.

6. Implementation:

The sixth phase is when most of the code for the program is written. Additionally, this phase involves the actual installation of the newly-developed system. This step puts the project into production by moving the data and components from the old system and placing them in the new system via a direct cutover. While this can be a risky (and complicated) move, the cutover typically happens during off-peak hours, thus minimizing the risk. Both system analysts and end-users should now see the realization of the project that has implemented changes.

7. Operations and Maintenance:

The seventh and final phase involves maintenance and regular required updates. This step is when end users can fine-tune the system, if they wish, to boost performance, add new capabilities or meet additional user requirements.

**Prototype model and phases**

This Prototype Model is same as waterfall model, but in this model, we need to develop prototype and customer interaction will be there. Since there is customer interaction there will be less chance of rejection. When to use Prototype model:

* Whenever the customer not clears about the requirement in this situation we generally go for prototype model.
* If it is complex project then prototype model makes clear understand the requirement.
* Prototyping make sure that the customer constantly works with the system and provide a feedback about the system.

Phases of the Prototype model:

* Identify some requirements to begin with: Get a list of some major requirements which define the need for the new system including the main input output information.
* Develop initial prototype: Develop a basic initial prototype which only has UI screens.
* Review the prototype: End users and SME’s work and examine the prototype and provide feedback for improvements/enhancements.
* Revise and enhance the prototype: Scope is changed based on feedback from end users and the prototype is enhanced and refined to accommodate user feedback.

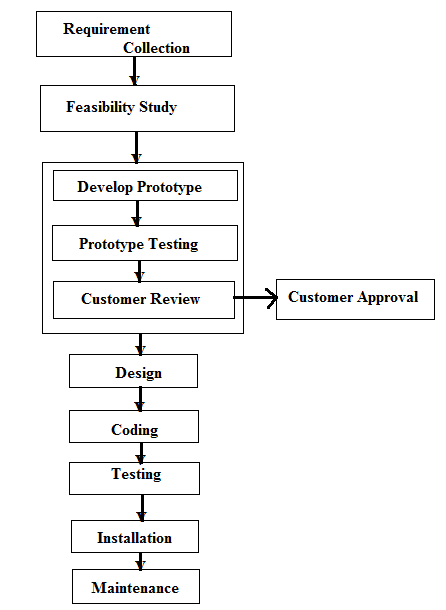
Advantages of Prototype model:

* Customer satisfaction exists, because customer can feel the product at very early stage.
* If there is missing functionality can be identified easily
* There will be less chance of software rejection.
* Requirement changes are allowed.
* Due to customer approval, we can find the errors at early stage.
* Customer involvement will be there in the development where its leads to better solutions for any confusion / complexity / difficult functions
* The developed prototype can be re-used by developer and test engineer.

Disadvantages of Prototype model:

* There are no parallel deliverables
* It is a time consuming if customer ask for changes in prototype
* This methodology may increase the system complexity as scope of the system may expand beyond original plans.
* The invested effort in the preparation of prototypes may be too much if not properly monitored.
* Customer may get confused in the prototypes and real systems.

The prototype model:



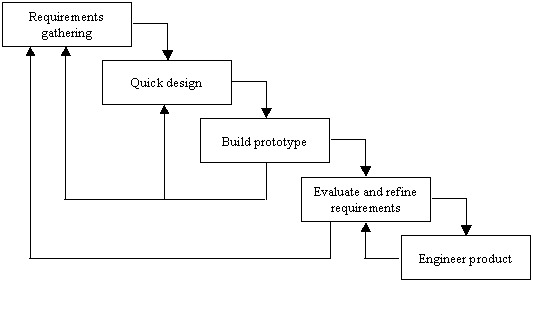
**Extreme Programming and extreme prototyping**

Extreme Programming (XP) is an agile software development framework that aims to produce higher quality software, and higher quality of life for the development team. XP is the most specific of the agile frameworks regarding appropriate engineering practices for software development. The five values of XP are communication, simplicity, feedback, courage, and respect.

Pair Programming means all production software is developed by two people sitting at the same machine. The idea behind this practice is that two brains and four eyes are better than one brain and two eyes. You effectively get a continuous code review and quicker response to nagging problems that may stop one person dead in their tracks.

Teams that have used pair programming have found that it improves quality and does not actually take twice as long because they are able to work through problems quicker and they stay more focused on the task at hand, thereby creating less code to accomplish the same thing.

Extreme prototyping is used in the web development domain. It consists of three sequential phases. First, a basic prototype with all the existing pages is presented in the HTML format. Then the data processing is simulated using a prototype services layer. Finally, the services are implemented and integrated to the final prototype. This process is called Extreme Prototyping used to draw attention to the second phase of the process, where a fully functional UI is developed with very little regard to the actual services.



**Agile manifesto**

Agile is a time boxed, iterative approach to software delivery that builds software incrementally from the start of the project, instead of trying to deliver it all at once near the end. It works by breaking projects down into little bits of user functionality called user stories, prioritizing them, and then continuously delivering them in short two-week cycles called iterations.

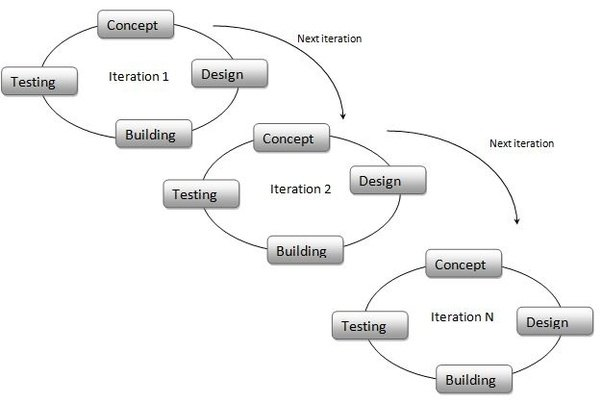
Agile Values:

* Individuals and Interactions over processes and tools
* Working Software over comprehensive documentation
* Customer Collaboration over contract negotiation
* Responding to Change over following a plan

The Manifesto for Agile Software Development is based on twelve principles:

* Customer satisfaction by early and continuous delivery of valuable software.
* Welcome changing requirements, even in late development.
* Deliver working software frequently (weeks rather than months)
* Close, daily cooperation between business people and developers
* Projects are built around motivated individuals, who should be trusted
* Face-to-face conversation is the best form of communication (co-location)
* Working software is the primary measure of progress
* Sustainable development, able to maintain a constant pace
* Continuous attention to technical excellence and good design
* Simplicity—the art of maximizing the amount of work not done—is essential
* Best architectures, requirements, and designs emerge from self-organizing teams
* Regularly, the team reflects on how to become more effective, and adjusts accordingly

In agile model, the product or solution is first divided into features which need to be developed. If there are new features identified during the complete product release it again gets planned across sprints. Agile Sprint duration is decided based on feature to be developed. Every sprint goes through the phases of Requirement, Design, Development and Testing phase. The most important of the principles is customer satisfaction by giving rapid and continuous delivery of small and useful software.



**Software testing**

Unit Testing:

Unit tests designed in the module design phase are executed on the code during this validation phase. Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.

Integration Testing:

Integration testing is associated with the architectural design phase. Integration tests are performed to test the coexistence and communication of the internal modules within the system.

System Testing:

System testing is directly associated with the system design phase. System tests check the entire system functionality and the communication of the system under development with external systems. Most of the software and hardware compatibility issues can be uncovered during this system test execution.

Acceptance Testing:

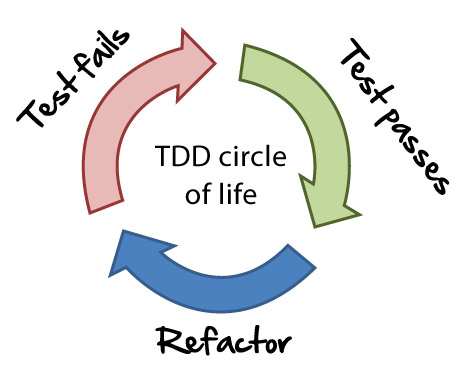
Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests uncover the compatibility issues with the other systems available in the user environment. It also discovers the non-functional issues such as load and performance defects in the actual user environment.

Test Driven Development:

Test Driven Development is about writing the test first before adding new functionality to the system. This seems backwards as first, but doing this:

* Defines success up front.
* Helps break our design down into little pieces, and
* Leaves us with a nice suite of unit tests proving our stuff works.

Agile developers work in this circle of life when adding new code. Write the test first. Make it pass. Then refactor.



**Scrum**

Scrum is a framework, which is often part of agile software development that emphasizes teamwork, accountability and iterative progress toward a well-defined goal. The framework begins with a simple premise: Start with what can be seen or known. After that, track the progress and tweak as necessary. Agile is the broader umbrella which Scrum falls under. Agile has four main values and twelve principles. Scrum has its own set of values and principles and provides a lightweight “framework” to help teams become Agile.

The product owner is the project’s key stakeholder. Typically, the product owner will be the primary user of the product, or at least have a deep understanding of who will. Despite this expertise, the product owner does not get to determine how much work happens in the sprint cycles, or alter the goals for that sprint. Product owners must be available to the team, and engage actively with it. Communication is a huge part of this, as the product owner communicates with both the team and other stakeholders.

The Scrum Master is the person who ensures the team keeps to the values and practices of Scrum, sort of like a coach. They remove impediments, facilitate meetings and work with product owners. Interestingly, the Scrum Master is a servant-leader who doesn’t have authority over the team, but does have authority over the process. They can’t fire people, but they can alter how long the sprints are. This can make the role more challenging than a traditional management role.

In a Scrum team, everyone works together to do whatever it takes to complete tasks they’ve all agreed on for a sprint. The Scrum team might have five to nine people on it. When projects are larger, you work with teams of teams, rather than making larger teams. In this case, teams may designate one member to attend meetings with people from other teams for something analogous to the daily Scrum, though it only takes place every few days. This section includes a helpful diagram of such a scenario.

Product Owner:

* Define sprint scope
* Define each user story items in detail
* Prepare notes or include them in the agenda the thing concern, the question for a user story so the team can come prepare and have those answers before the before.
* Make sure backlog is in priority order
* Describe definition of done for each user story and for sprint

Scrum Master

Practicalities are taken care of, since scrum master is the facilitator, make sure that all the merchandising are available for the meeting

* Making sure video, audio conference is reserved and working
* Pre-order coffee and light refreshment, “no coffee no code”
* Enough post-it notes and markers, whiteboard etc.
* Available capacity for the team (team members’ holidays, public holidays, etc.)
* Keep tracking the teams’ velocity vs the amount of work committed by the team

Development Team:

* Ask questions to clear any ambiguity or else “speak now or hold your ... “
* Do not commit more than you cannot deliver
* Remember to think of your personal availability for the coming sprint.

A scrum sprint is a regular, repeatable work cycle in scrum methodology during which work is completed and made ready for review. Scrum sprints are basic units of development in the scrum methodology. Generally, scrum sprints are less than 30 days long.

The meetings are usually timeboxed to between 5 and 15 minutes, and take place with participants standing up to remind people to keep the meeting short and to-the-point. The stand-up meeting is sometimes also referred to as the "stand-up", "morning rollcall" or "daily scrum".

The Product Backlog is an ordered list of everything that is known to be needed in the product. It is the single source of requirements for any changes to be made to the product. The Product Owner is responsible for the Product Backlog, including its content, availability, and ordering. However, the Sprint Backlog is the set of Product Backlog items selected for the Sprint, plus a plan for delivering the product Increment and realizing the Sprint Goal.

Fibonacci:

In Agile software development, the Fibonacci scale consists of a sequence of numbers used for estimating the relative size of user stories in points.

Estimating the effort involved in developing product backlog requirements is an ongoing process. The Fibonacci sequence is an excellent sizing technique for relative estimating. With Fibonacci, if something is bigger, you get an idea of how much bigger it is. The last two numbers in the sequence are added together to create the next number. It looks like this:

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, and so on

Initial, high-level requirements are estimated at the product roadmap level:

For scrum teams, the development teams understand that requirements with Fibonacci number estimates from 1 through 8 can be brought into a sprint. This level of refinement usually results in a user story.

Requirements with estimates numbered from 13 through 34 are those that you would let into a release but need to be broken down further before you would let them into a sprint.

Requirements from 55 through 144 are too big for a release but are estimable at the order-of- magnitude product roadmap level. These requirements typically reflect features.

Requirements larger than 144 needs to be broken down before the development team can give any semblance of an accurate estimate, so don’t estimate above 144. These may represent broader themes.

Whatever the Fibonacci number, only the highest-priority cards get broken down into sprint-level sizes (which shouldn’t be more than an 8). So, if you have a high-priority requirement with a 21 Fibonacci number assigned to it, it needs to be broken into smaller requirements before it can come into a sprint.

**User centred design**

User centred design (UCD) is a design philosophy and a process in which the needs, wants, and limitations of the end user of an interface or document are given extensive attention at each stage of the design process. This helps to put customers at the heart of the online experience. By designing for users first and foremost, not only does the user experience improve (and thus, profit), but so does development efficiency and cost – designing based on user’s requirements means less redevelopment work is needed further down the line.

Five user centred design:

1. A clear understanding of user and task requirements.

2. Incorporating user feedback to define requirements and design.

3. Early and active involvement of the user to evaluate the design of the product.

4. Integrating user-centred design with other development activities.

5. Iterative design process.

A persona, (also user persona, customer persona, buyer persona) in user-centred design and marketing is a fictional character created to represent a user type that might use a site, brand, or product in a similar way. The purpose of personas is to create a reliable and realistic representations of your key audience segments for reference.

Benefits of a using personas:

* Personas help the team to align & focus decisions by considering a real person.
* Helps the PO & the team evaluate new features.
* Allows Visual Designers, UX, Copywriters to ensure the content, interactions, interface, behaviours & the labels are appropriate to the audience.
* Developers decide which approaches to take based on user behaviours.

User stories are short, simple descriptions of a feature told from the perspective of the person, user or customer of the system. It encourages the team to discuss the features or the problem.

As a < type of user>

I want < goal / objective>

so that < benefit / value >

Sometimes a story is too big to estimate or complete in a sprint or the details are sketchy. This is an epic. Think of it as a headline or a chapter heading. So, we must break it down into smaller stories. Acceptance criteria allows you to describe when a story is complete. They are typically shown as a list or follow this simple template:

given <some precondition>

when <I do some action>

then <I expect some result>

Benefits of acceptance criteria:

* Allows the team to gain a deeper, shared understanding of the story/feature from a user’s perspective.
* Allows the PO to answer what they need for this feature to provide value.
* Allows developers and testers to derive tests.
* Helps the team know when to stop adding functionality to a story.

**End of Revision Book**