CSE 3102: SOFTWARE ENGINEERING

Abdullah Al Shiam
Lecturer
Dept. of Computer Science and Engineering
Sheikh Hasina University, Netrokona

UNIT1: INTRODUCTION

What is Software?

Software is more than just a program code. Software is considered to be collection of executable programming code, associated libraries and documentations

What is engineering?

Applying scientific knowledge for practical application considering economic aspects.

What is Software engineering?

Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

What is the difference between software engineering and computer science?

1. Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.

2. Computer science theories are still insufficient to act as a complete foundation for software engineering (unlike e.g. physics and electrical engineering).

What is the difference between software engineering and system engineering?

- System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this process concerned with developing the software infrastructure, control, applications and databases in the system.

- System engineers are involved in system specification, architectural design, integration and deployment.

Why we create software or what are the targets of software development or what are the importance of software development?

Reasons/Targets/Importance of Software Development:

Software is the single most important technology on the world stage. The category of software is huge where categories may have subcategories. And this is all because, software is getting used in almost every aspects of human life. Everywhere we go we will find a reason or necessity to have software there.

Make use of hardware:
Software is important to make the hardware working.

Work Fast:
Software is important to make task faster to complete, such as, making computerized banner is faster than hand-made etc.

Security:
Software is important to build up security where it needs to be done, such as, in banks, money transaction etc.

Make Task Easier:
Software is important to make a task easier, such as, distribution of products, products information etc.

Make Task Reliable:
Software is important as the tasks performed by software are more reliable, such as, a calculation of billions done computer is more reliable than that by human.

Long Calculation:

Software is important to use the computers power or working efficiency to perform those tasks which cannot be done or controlled by human, such as, to measuring the temperature of world, distance of planets etc.

Operate Other Devices:
Software is important to operate those things which are totally computer controlled, such as, mobile phones, trains etc.

Better Service:
Software is important to ensure better service, such as, in emergency medi-care, to supplying product information through internet etc.

Shortly, it can be said that software is important to make things easier, faster, more reliable and safer.

Software Evolution:

The process of developing a software product using software engineering principles and methods is referred to as Software Evolution. This includes the initial development of software and its maintenance and updates, till desired software product is developed, which satisfies the expected requirements.

Evolution starts from the requirement gathering process. After which developers create a prototype of the intended software and show it to the users to get their feedback at the early stage of the software product development. The users suggest changes, on which several consecutive updates and maintenance keep on changing too. This process changes to the original software, till the desired software is accomplished.

Even after the user has the desired software in hand, the advancing technology and the changing requirements force the software product to change accordingly. Re-creating software from scratch and to go one-on-one with the requirement is

(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

(2) The study of approaches as in the above statement.

Fritz Bauer, a German computer scientist, defines software engineering as:

"Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and work efficiently on real machines." The only feasible and economical solution is to update the existing software so that it matches the latest requirements.

Objectives of Software Engineering:

To improve quality of software products
To increase customer satisfaction
To increase productivity
To increase job satisfaction
Software engineering is not programming. Programming is an important part of software engineering.
"This is not a programming course"

Software Applications:

1. System Software
2. Real-time Software
3. Business Software
4. Engineering & Scientific Software
5. Embedded Software
6. Personal Computer Software
7. Artificial Intelligence Software

Important components of Software Engineering are:

1. Software Development Life Cycle (SDLC)
2. Software Quality Assurance
3. Software project Management

4. Computer Aided Software Engineering (CASE)

Software Crisis – Problem & causes:

In the late 1960s, it became clear that the development of software is different from manufacturing other products. This is because employing more manpower (programmers) later in the software development does not always help speed up the development process. Instead, sometimes it may have negative impacts like delay in achieving the scheduled targets, degradation of software quality, etc. Though software has been an important element of many systems since a long time, developing software within a certain schedule and maintaining its quality is still difficult.

History has seen that delivering software after the scheduled date or with errors has caused large scale financial losses as well as inconvenience to many. Disasters such as the Y2Kproblem affected economic, political, and administrative systems of various countries around the world. This situation, where catastrophic failures have occurred, is known as **software crisis.**

Software project crisis management:

Delay detection
Crisis management
Crisis recovery
Assign responsibilities and authorities.
Update status frequently.
Relax resource constrains.
Have project personnel operate in burnout mode.
Establish a drop-dead date.
Clear out unessential personnel.

Characteristics of good software:

A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

• Operational

• Transitional

• Maintenance

Well-engineered and crafted software is expected to have the following characteristics:

Operational

This tells us how well the software works in operations. It can be measured on:

• Budget

• Usability

• Efficiency

• Correctness

• Functionality

• Dependability

• Security

• Safety

Transitional

This aspect is important when the software is moved from one platform to another:

• Portability

• Interoperability

• Reusability

• Adaptability

Maintenance

This aspect briefs about how well the software has the capabilities to maintain itself in the ever-changing environment:

• Modularity

• Maintainability

• Flexibility

• Scalability

Software does not Manufactured it is developed or engineered:

Software is developed or engineered but it is not manufactured in the classical sense: Although some similarities exist between software development and hardware manufacture, the two activities are fundamentally different. Software is a design of strategies, instruction which finally perform the designed, instructed tasks. And a design can only be developed, not manufactured. On the other hand, design is just a material of hardware.

Software is virtual. That is, software can be used using proper hardware. And we can only use it. But we can use, touch, and see hardware. Thus software never gets manufactured, they are developed.

In the case of software, a good design will introduce good software, if done through the design. But in the case of hardware, the design is only the half way. In the manufacturing phase it may introduce quality problem and this phase does not exist in the case of software.

Thus it can be said that the software are developed or engineered, not manufactured.

Software doesn't "wear out"- Explain this comparing with hardware:

Considering the given figure 01. This is often called the "bathtub curve". It indicates that, at the beginning of the life of hardware it shows high failure rate as it contains many defects. By time, the manufacturers or the designers repair these defects and it becomes idealized or gets into the steady state and continues. But after that, as time passes, the failure rate rises again and this may be caused by excessive temperature, dust, vibration, improper use and so on and at one time it becomes totally unusable. This state is the "wear out" state.

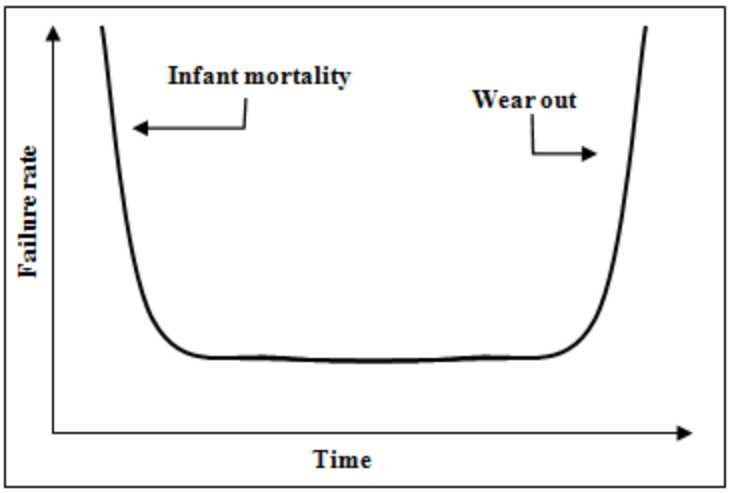


Figure 01: Failure curve for hardware

On the other hand, software does not wear out. Like hardware, software also shows high failure rate at its infant state. Then it gets modifications and the defects get corrections and thus it comes to the idealized state.

But though a software not having any defects it may get the need of modification as the users demand from the software may change. And when it occurs, the unfulfilled demands will be considered as defects and thus the failure rate will increase. After one modification another may get the necessity. In that way, slowly, the minimum failure rate level begins to rise which will cause the software deteriorated due to change, but it does not “wear out”.

In the following given figure 02, the idealized curve for software has been shown. Also the change to failure rate due to users demand and the rise of the minimum failure rate has been shown on the actual curve.

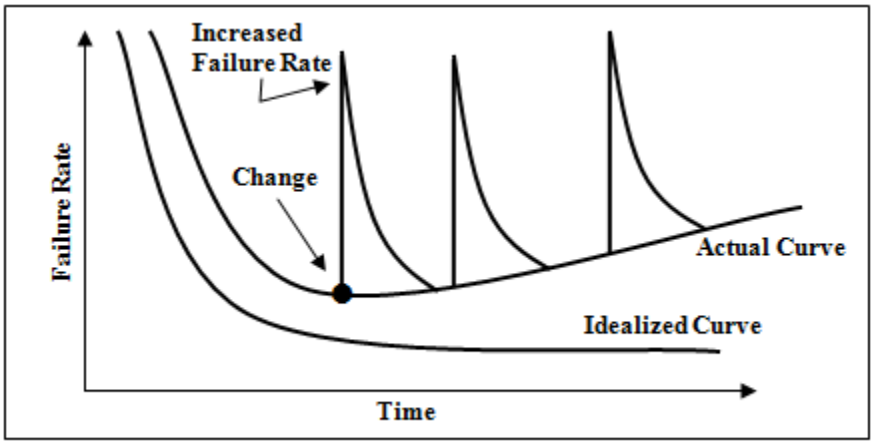


Figure 02: Failure Curves for software

Software Myths: What is software myth in software engineering:

Management Myth:

- We already have a book of standards and procedures for building software. It does provide my people with everything they need to know.
- If my project is behind the schedule, I always can add more programmers to it and catch up.
- If I decide to outsource the software project to a third party, I can just relax: Let them build it, and I will just pocket my profits.

Customer/User Myth:

- A general statement of objectives is sufficient to begin writing programs - we can fill in the details later.

- Project requirements continually change but this change can easily be accommodated because software is flexible.

Developer Myth:

- Let's start coding ASAP (as soon as possible), because once we write the program and get it to work, our job is done.

- Until I get the program running, I have no way of assessing its quality.

- The only deliverable work product for a successful project is the working program.

- Software engineering is baloney (nonsense). It makes us create tons of paperwork, only to slow us down.

UNIT2: SOFTWRAE DEVELOPMENT LIFE CYCLE

Describe the software development life cycle?

Software Development Life Cycle, SDLC for short, is a well-defined, structured sequence of stages in software engineering to develop the intended software product.

SDLC Activities:

SDLC provides a series of steps to be followed to design and develop a software product efficiently. SDLC framework includes the following steps:

Communication:

This is the first step where the user initiates the request for a desired software product. The user contacts the service provider and tries to negotiate the terms, submits the request to the service providing organization in writing.

Requirement Gathering:

The process to gather the software requirements from client, analyze, and document them is known as requirement engineering. The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'System Requirements Specification' document.

Feasibility Study:

After requirement gathering, the team comes up with a rough plan of software process. At this step the team analyzes if a software can be designed to fulfill all requirements of the user, and if there is any possibility of software being no more useful. It is also analyzed if the project is financially, practically, and technologically feasible for the organization to take up. There are many algorithms available, which help the developers to conclude the feasibility of a software project.

System Analysis:

At this step the developers decide a roadmap of their plan and try to bring up the best software model suitable for the project. System analysis includes understanding of software product limitations, learning system related problems or changes to be done in existing systems beforehand, identifying and addressing the impact of project on organization and personnel etc. The project team analyzes the scope of the project and plans the schedule and resources accordingly.

Software Design:

Next step is to bring down whole knowledge of requirements and analysis on the desk and design the software product. The inputs from users and information gathered in requirement gathering phase are the inputs of this step. The output of this step comes in the form of two designs; logical design, and physical design. Engineers produce meta-data and data dictionaries, logical diagrams, data-flow diagrams, and in some cases pseudo codes.

Coding:

This step is also known as programming phase. The implementation of software design starts in terms of writing program code in the suitable programming language and developing error-free executable programs efficiently.

Testing:

An estimate says that 50% of whole software development process should be tested. Errors may ruin the software from critical level to its own removal. Software testing is done while coding by the developers and thorough testing is conducted by testing experts at various levels of code such as module testing,
Software Engineering Tutorial
Program testing, product testing, in-house testing, and testing the product at user's end. Early discovery of errors and their remedy is the key to reliable software.

Integration:

Software may need to be integrated with the libraries, databases, and other program(s). This stage of SDLC is involved in the integration of software with outer world entities.

Implementation:

This means installing the software on user machines. At times, software needs post-installation configurations at user end. Software is tested for portability and adaptability and integration related issues are solved during implementation.

Operation and Maintenance:

This phase confirms the software operation in terms of more efficiency and less errors. If required, the users are trained on, or aided with the documentation on how to operate the software and how to keep the software operational. The software is maintained timely by updating the code according to the changes taking place in user end environment or technology. This phase may face challenges from hidden bugs and real-world unidentified problems.

Software Development Paradigm:

The software development paradigm helps a developer to select a strategy to develop the software. A software development paradigm has its own set of tools, methods, and procedures, which are expressed clearly and defines software development life cycle. A few of software development paradigms or process models are defined as follows:

Waterfall Model:

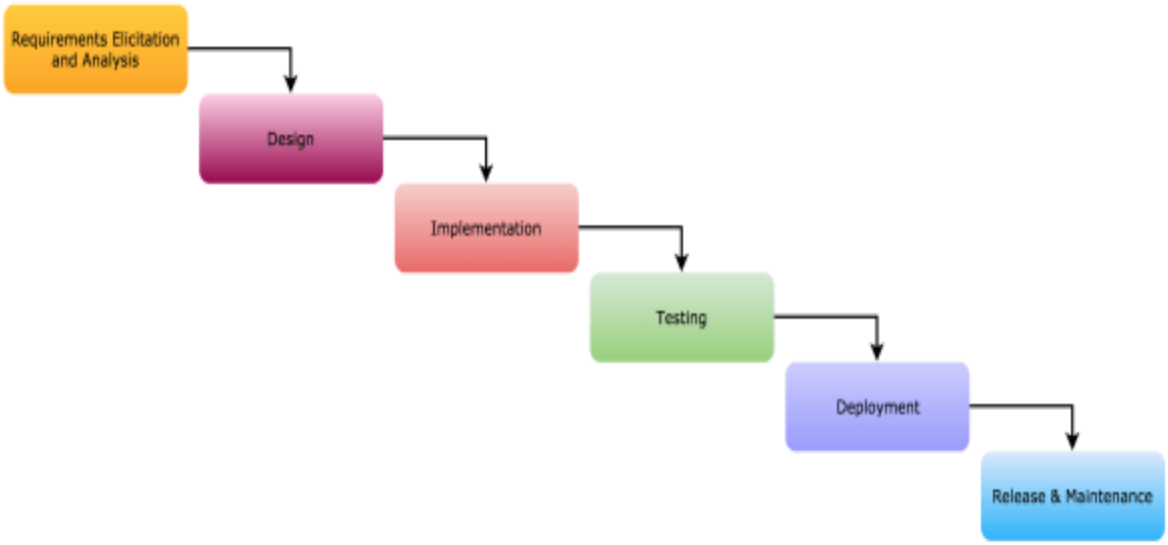
Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.
Following is a diagrammatic representation of different phases of waterfall model.

The sequential phases in Waterfall model are:

- **Requirement Gathering and analysis:** All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification doc.
- **System Design:** The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.
- **Implementation:** With inputs from system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.
- **Integration and Testing:** All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- **Deployment of system:** Once the functional and nonfunctional testing is done, the product is deployed in the customer environment or released into the market.
- **Maintenance:** There are some issues which come up in the client environment. To fix those issues patches are released. Also to enhance the

product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

Stages of Waterfall Model:



Waterfall Model Advantage & Disadvantage:

Advantage:

- It allows for departmentalization and managerial control.
- Simple and easy to understand and use.
- Easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- A schedule can be set with deadlines for each stage of development and a product can proceed through the development process like a car in a car-wash, and theoretically, be delivered on time.

Disadvantage:

- It does not allow for much reflection or revision.
- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.

Spiral Model:

The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.

Identification: This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.

This also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral the product is deployed in the identified market.

Design: Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and final design in the subsequent spirals.

Construct or Build: Construct phase refers to production of the actual software product at every spiral. In the baseline spiral when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.

Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number. These builds are sent to customer for feedback.

Evaluation and Risk Analysis: Risk Analysis includes identifying, estimating, and monitoring technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.

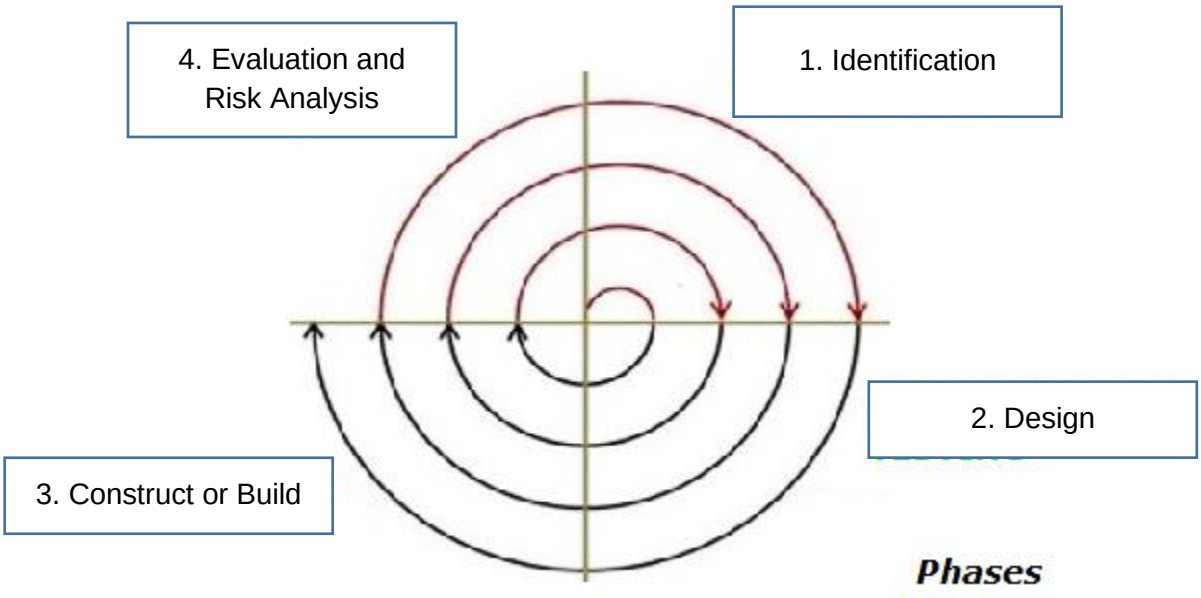


Figure: Spiral Model

Spiral Model Application:

The Spiral Model is widely used in the software industry as it is in sync with the natural development process of any product, i.e. learning with maturity which involves minimum risk for the customer as well as the development firms.

The following pointers explain the typical uses of a Spiral Model −

- When there is a budget constraint and risk evaluation is important.
- For medium to high-risk projects.

- Long-term project commitment because of potential changes to economic priorities as the requirements change with time.

- Customer is not sure of their requirements which is usually the case.

- Requirements are complex and need evaluation to get clarity.

- New product line which should be released in phases to get enough customer feedback.

- Significant changes are expected in the product during the development cycle.

Spiral Model Pros and Cons:

Pros:

- Changing requirements can be accommodated.
- Allows for extensive use of prototypes
- Requirements can be captured more accurately.
- Users see the system early.

Cons:

- Management is more complex.
- Not suitable for small or low risk projects and could be expensive for small projects.
- Process is complex
- Spiral may go indefinitely.
- Large number of intermediate stages requires excessive documentation.

Prototyping Model in Software Engineering:

The basic idea in Prototype model is that instead of freezing the requirements before a design or coding can proceed, a throwaway prototype is built to understand the requirements. This prototype is developed based on the currently known requirements. Prototype model is a software development model. By using this prototype, the client can get an "actual feel" of the system, since the interactions with prototype can enable the client to better understand the requirements of the desired system. Prototyping is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determining the requirements.

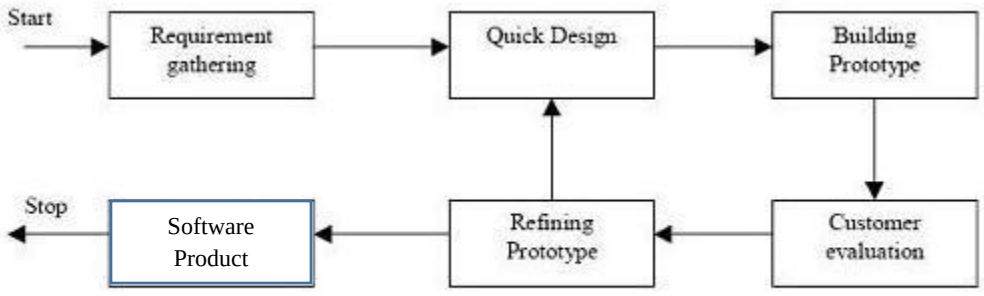
Following is the stepwise approach to design a software prototype:

- **Basic Requirement Identification:** This step involves understanding the very basics product requirements especially in terms of user interface. The more intricate details of the internal design and external aspects like performance and security can be ignored at this stage.

- **Developing the initial Prototype:** The initial Prototype is developed in this stage, where the very basic requirements are showcased and user interfaces are provided. These features may not exactly work in the same manner internally in the actual software developed and the workarounds are used to give the same look and feel to the customer in the prototype developed.

- **Review of the Prototype:** The prototype developed is then presented to the customer and the other important stakeholders in the project. The feedback is collected in an organized manner and used for further enhancements in the product under development.

- **Revise and enhance the Prototype:** The feedback and the review comments are discussed during this stage and some negotiations happen with the customer based on factors like, time and budget constraints and technical feasibility of actual implementation. The changes accepted are again incorporated in the new Prototype developed and the cycle repeats until customer expectations are met.



Prototyping Model

Advantage:

- Provides a working model to the user early in the process, enabling early assessment and increasing user's confidence.
- The developer gains experience and insight by developing a prototype there by resulting in better implementation of requirements.
- The prototyping model serves to clarify requirements, which are not clear, hence reducing ambiguity and improving communication between the developers and users.
- There is a great involvement of users in software development. Hence, the requirements of the users are met to the greatest extent.
- Helps in reducing risks associated with the software.

Disadvantage:

- If the user is not satisfied by the developed prototype, then a new prototype is developed. This process goes on until a perfect prototype is developed. Thus, this model is time consuming and expensive.
- The developer loses focus of the real purpose of prototype and hence, may compromise with the quality of the software. For example, developers may use some inefficient algorithms or inappropriate programming languages while developing the prototype.

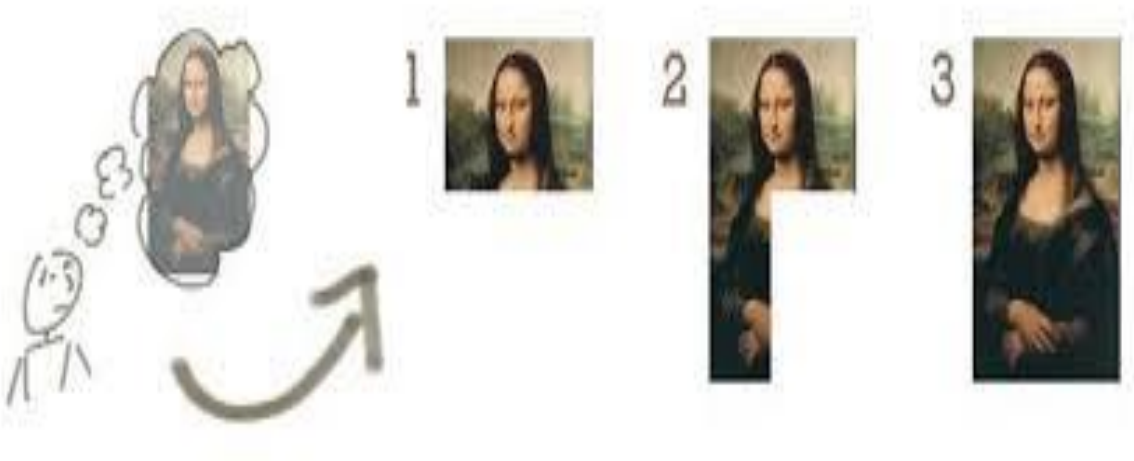
- Prototyping can lead to false expectations. For example, a situation may be created where the user believes that the development of the system is finished when it is not.
- The primary goal of prototyping is speedy development, thus, the system design can suffer as it is developed in series without considering integration of all other components.

Iterative/Incremental Model:

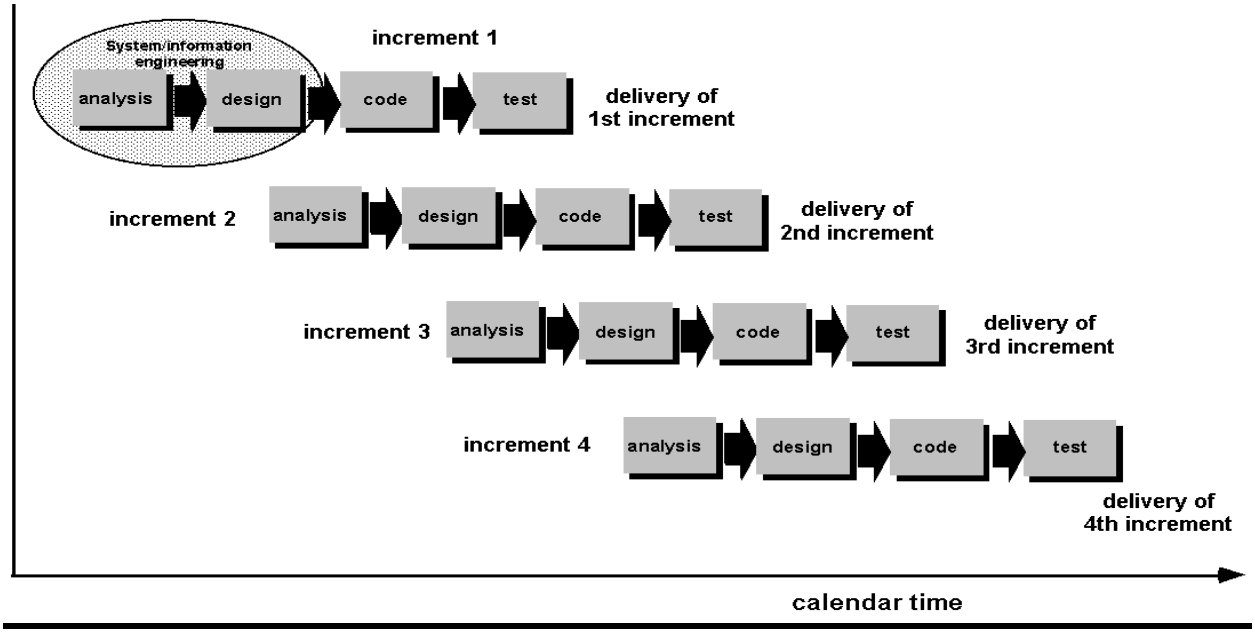
In Iterative model, iterative process starts with a simple implementation of a small set of the software requirements and iteratively enhances the evolving versions until the complete system is implemented and ready to be deploy.

Iterative Model - Design

Iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).



The following illustration is a representation of the Iterative and Incremental model –



Iterative and Incremental development is a combination of both iterative design or iterative method and incremental build model for development. "During software development, more than one iteration of the software development cycle may be in progress at the same time.

During each iteration, the development module goes through the requirements, design, implementation and testing phases. Each subsequent release of the module adds function to the previous release. The process continues till the complete system is ready as per the requirement.

When to use Incremental Model:

- Risk, funding, schedule, program complexity, or need for early realization of benefits.
- Most of the requirements are known up-front but are expected to evolve over time.
- A need to get basic functionality to the market early.
- On projects which have lengthy development schedules.

- On a project with new technology.
- Mostly such model is used in web applications and product based companies.

Advantages of Incremental model:

- Develop high risk or major function first.
- Each release delivers an operational product.
- Customer can respond to each build.
- Uses “divide and Conquer” break down of task.
- Lower initial delivery cost.
- Initial product delivery is faster.
- Customers get important functionality early.
- Risk of changing requirements is reduced.

Disadvantages of Incremental model:

- Require good planning and design.
- Requires early definition of a complete and fully functional system to allow for the definition of increments.
- Well-defined module interfaces are required.
- Total cost of complete system is not lower.

Agile model:

Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.

Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three weeks. Every iteration involves cross functional teams working simultaneously on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing.

At the end of the iteration a working product is displayed to the customer and important stakeholders.

Agile Principles:

There are 12 agility principles for those who want to achieve agility:

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.
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.

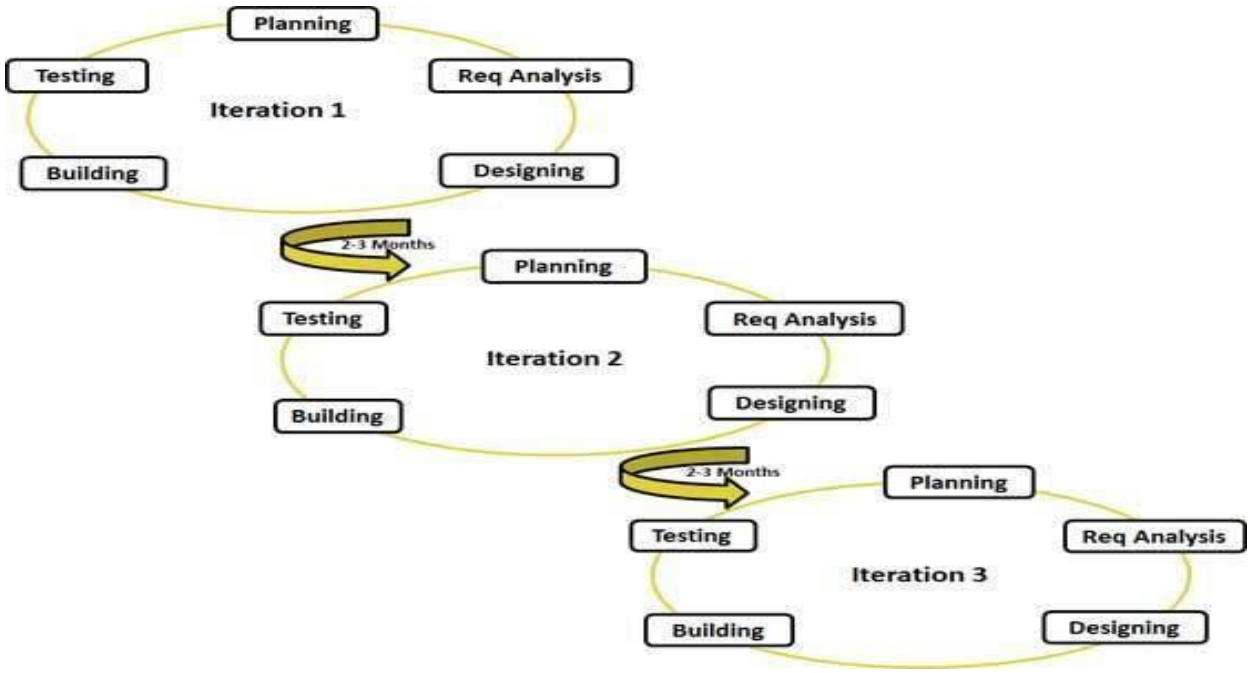


Figure: Agile Model

Advantages of Agile model:

- Customer satisfaction by rapid, continuous delivery of useful software.
- People and interactions are emphasized rather than process and tools. Customers, developers and testers constantly interact with each other.
- Working software is delivered frequently (weeks rather than months).
- Face-to-face conversation is the best form of communication. .
- Regular adaptation to changing circumstances.

Disadvantages of Agile model:

- In case of some large software deliverables, it is difficult to assess the effort required at the beginning of the software development life cycle.
- There is lack of emphasis on necessary designing and documentation.
- The project can easily get taken off track if the customer representative is not clear what final outcome that they want.
- Only senior programmers are capable of taking the kind of decisions required during the development process. Hence it has no place for newbie programmers

When to use Agile model:

- When new changes are needed to be implemented. The freedom agile gives to change is very important. New changes can be implemented at very little cost because of the frequency of new increments that are produced.
- To implement a new feature the developers need to lose only the work of a few days, or even only hours, to roll back and implement it.
- Unlike the waterfall model in agile model very limited planning is required to get started with the project. Agile assumes that the end users' needs are ever changing in a dynamic business and IT world.

Best well known agile method:

1. Extreme Programming (XP)
2. Scrum
3. Crystal
4. Adaptive Software Development
5. Feature Driven Development

Agile vs Traditional Model:

Suppose Google is working on project to come up with a competing product for MS Word:

That provides all the features provided by MS Word and any other features requested by the marketing team.

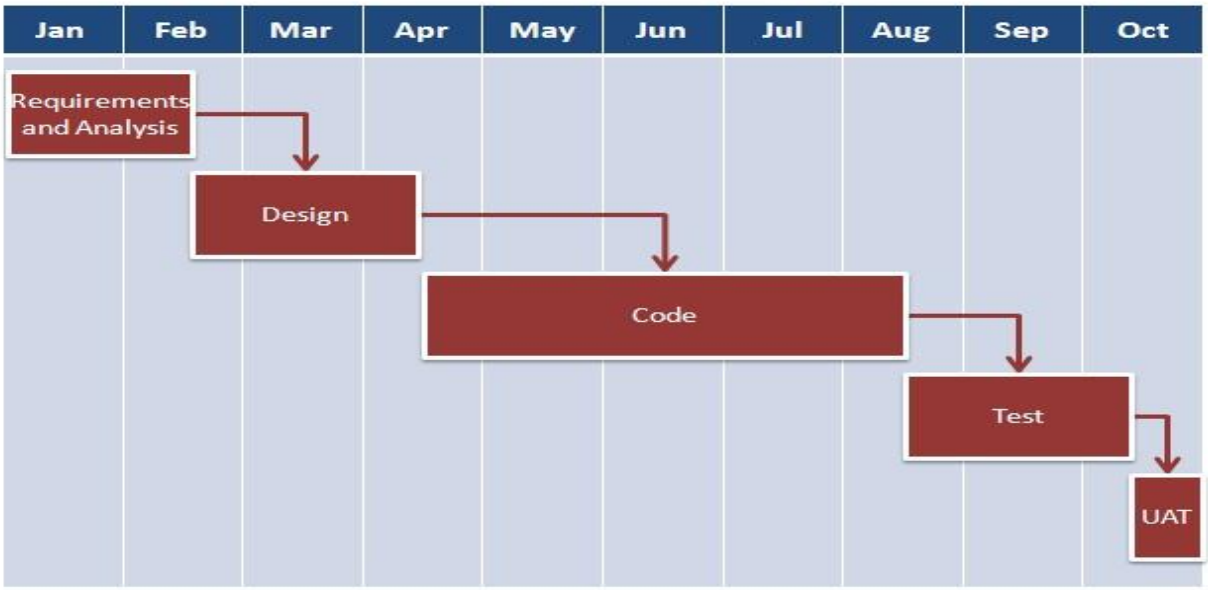
The final product needs to be ready in 10 months of time. Let us see how this project is executed in traditional and Agile methodologies.

In traditional Waterfall model –

- At a high level, the project teams would spend 15% of their time on gathering requirements and analysis (1.5 months)
- 20% of their time on design (2 months)
- 40% on coding (4 months) and unit testing
- 20% on System and Integration testing (2 months).
- At the end of this cycle, the project may also have 2 weeks of User Acceptance testing by marketing teams.

- In this approach, the customer does not get to see the end product until the end of the project, when it becomes too late to make significant changes.

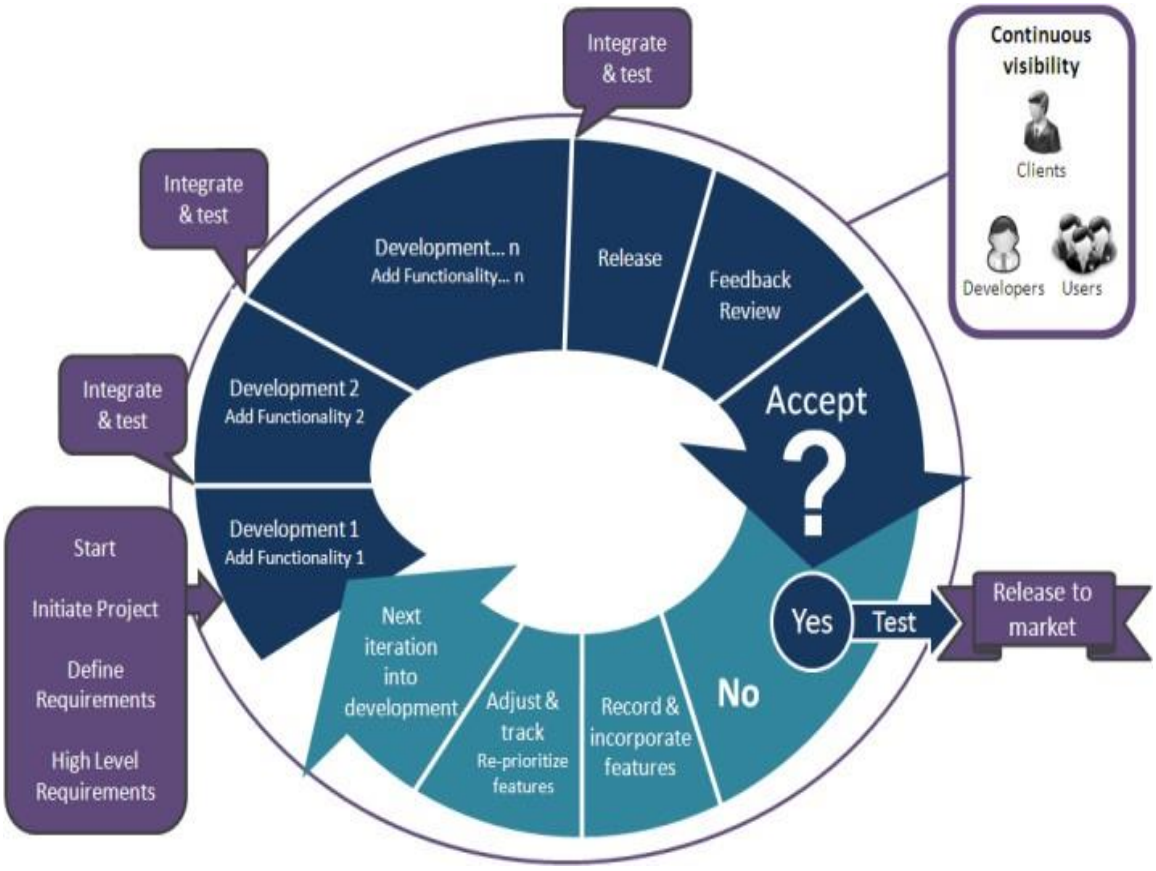
Project Schedule in Waterfall Model-



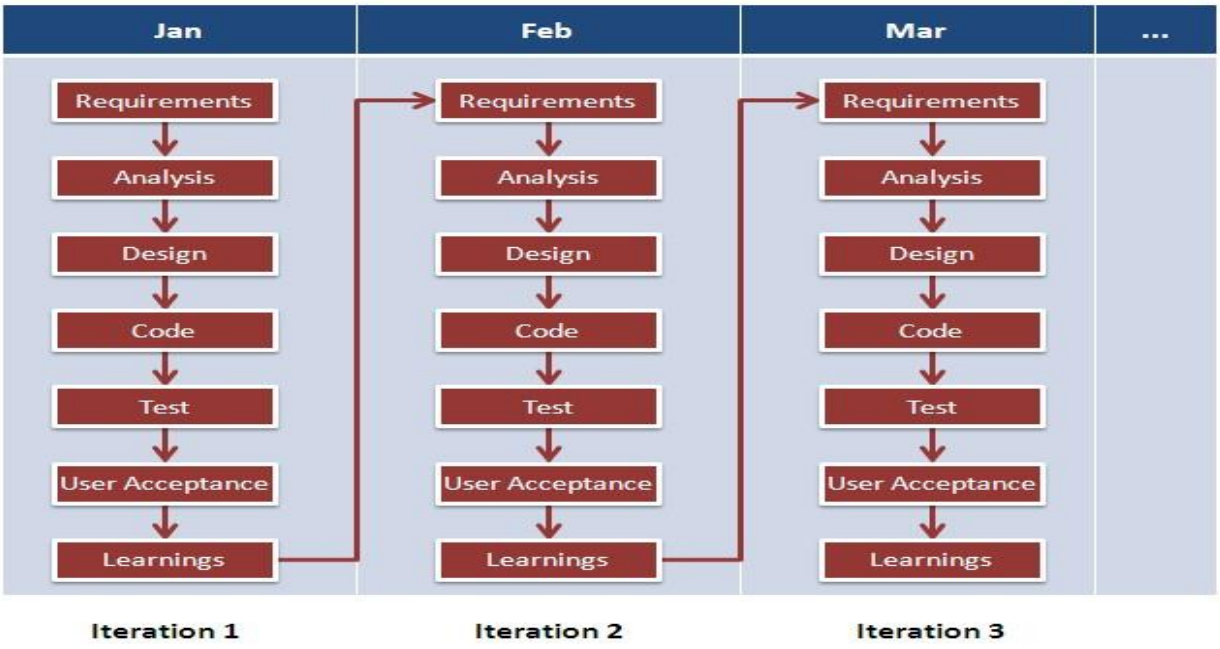
With Agile development methodology –

- In the Agile approach, each project is broken up into several 'Iterations'.
- All Iterations should be of the same time duration (between 2 to 8 weeks).
- At the end of each iteration, a working product should be delivered.
- In simple terms, in the Agile approach the project will be broken up into 10.
- Rather than spending 1.5 months on requirements gathering, in Agile software development, the team will decide the basic core features.
- Any remaining features that cannot be delivered in the first iteration will be taken up in the next iteration or subsequent iterations, based on priority.
- At the end of the first iterations, the team will deliver a working software with the features that were finalized for that iteration.

- There will be 10 iterations and at the end of each iteration the customer is delivered a working software that is incrementally enhanced and updated with the features that were shortlisted for that iteration.



Project Schedule in Agile Model-



Comparison between Agile and Waterfall:

Comparison between waterfall & agile

Waterfall	Agile
Uses stages or phases - requirement analysis, system design, implementation, testing, deployment and maintenance	Uses iterations known as sprints – confirmed requirements, develop and test system, released and start on the next project.
Suitable for big projects.	Suitable for small projects
Does not involve clients.	Clients are highly involved in the development of the project.
In waterfall, the project leader is called as project manager and most of them are from IT background.	Scrum Master is in control of the whole project and they may not have an IT background.
Interactions with users only happens when gathering the requirements and for user testing.	In Agile, constants meet ups is required to interact with users.
There is no turning back to the previous phase if there are any requirements or problems occur.	If there are any new requirements, the processes of the project are still running.

Requirement Gathering:

The process to gather the software requirements from client, analyze, and document them is known as requirement engineering. The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'System Requirements Specification' document.

Iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).

Requirement Engineering Process:

It is a four step process, which includes –

- Feasibility Study
- Requirement Gathering
- Software Requirement Specification
- Software Requirement Validation

Feasibility study:

After requirement gathering, the team comes up with a rough plan of software process. At this step the team analyzes if a software can be designed to fulfill all requirements of the user, and if there is any possibility of software being no more useful. It is also analyzed if the project is financially, practically, and technologically feasible for the organization to take up. There are many algorithms available, which help the developers to conclude the feasibility of a software project.

Requirement Gathering:

If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user. Analysts and engineers communicate with the client and end-users to know their ideas on what the software should provide and which features they want the software to include.

Requirement specification

A software requirements specification (SRS) is a document that captures complete description about how the system is expected to perform. It is usually signed off at the end of requirements engineering phase.

SRS is a document created by system analyst after the requirements are collected from various stakeholders.

SRS defines how the intended software will interact with hardware, external interfaces, speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations etc.

The requirements received from client are written in natural language. It is the responsibility of the system analyst to document the requirements in technical language so that they can be comprehended and used by the software development team.

Requirements Validation:

After requirement specifications are developed, the requirements mentioned in this document are validated. User might ask for illegal, impractical solution or experts may interpret the requirements inaccurately.

To check all the issues related to requirements, requirements validation is performed.

Requirements validation is similar to requirements analysis as both processes review the gathered requirements. Requirements validation studies the 'final draft' of the requirements document while requirements analysis studies the 'raw requirements' from the system stakeholders (users). Requirements validation and requirements analysis can be summarized as follows:

- Requirements validation: Have we got the requirements right?
- Requirements analysis: Have we got the right requirements?

Requirements validation determines whether the requirements are substantial to design the system. The problems encountered during requirements validation are listed below.

- Unclear stated requirements
- Conflicting requirements are not detected during requirements analysis
- Errors in the requirements elicitation and analysis
- Lack of conformance to quality standards.

To avoid the problems stated above, a requirements review is conducted, which consists of a review team that performs a systematic analysis of the requirements.

Requirements Validation Techniques

A number of other requirements validation techniques are used either individually or in conjunction with other techniques to check the entire system or parts of the system. The selection of the validation technique depends on the appropriateness and the size of the system to be developed. Some of these techniques are listed below.

1. **Test case generation:** The requirements specified in the SRS document should be testable. The test in the validation process can reveal problems in the requirement. In some cases test becomes difficult to design, which implies that the requirement is difficult to implement and requires improvement.
2. **Automated consistency analysis:** If the requirements are expressed in the form of structured or formal notations, then CASE tools can be used to check the consistency of the system. A requirements database is created using a CASE tool that checks the entire requirements in the database using rules of method or notation. The report of all inconsistencies is identified and managed.
3. **Prototyping:** Prototyping is normally used for validating and eliciting new requirements of the system. This helps to interpret assumptions and provide an appropriate feedback about the requirements to the user. For example, if users have approved a prototype, which consists of graphical user interface, then the user interface can be considered validated.

Requirements can be checked against following conditions -

• If they can be practically implemented

• If they are valid and as per functionality and domain of software

• If there are any ambiguities

• If they are complete

• If they can be demonstrated

Requirement Gathering Techniques/Process:

Requirements Elicitation is the process to find out the requirements for an intended software system by communicating with client, end users, system users, and others who have a stake in the software system development. There are various ways to discover requirements. Some of them are explained below:

✓ Interviews

Interviews are strong medium to collect requirements. Organization may conduct several types of interviews such as:

- Structured (closed) interviews, where every single information to gather is decided in advance, they follow pattern and matter of discussion firmly.
- Non-structured (open) interviews, where information to gather is not decided in advance, more flexible and less biased.
- Oral interviews.
- Written interviews
- One-to-one interviews which are held between two persons across the table.
- Group interviews which are held between groups of participants. They help to uncover any missing requirement as numerous people are involved.

✓ Surveys

Organization may conduct surveys among various stakeholders by querying about their expectation and requirements from the upcoming system.

✓ Questionnaires:

A document with pre-defined set of objective questions and respective options is handed over to all stakeholders to answer, which are collected and compiled. A shortcoming of this technique is, if an option for some issue is not mentioned in the questionnaire, the issue might be left unattended.

- ✓ Brainstorming:

  An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.

- ✓ Prototyping:

  Prototyping is building user interface without adding detail functionality for user to interpret the features of intended software product. It helps giving better idea of requirements. If there is no software installed at client's end for developer's reference and the client is not aware of its own requirements, the developer creates a prototype based on initially mentioned requirements. The prototype is shown to the client and the feedback is noted. The client feedback serves as an input for requirement gathering.

- ✓ Observation:

  Team of experts visit the client's organization or workplace. They observe the actual working of the existing installed systems. They observe the workflow at the client's end and how execution problems are dealt. The team itself draws some conclusions which aid to form requirements expected from the software.

Software Requirements Characteristics:

Gathering software requirements is the foundation of the entire software development project. Hence they must be clear, correct, and well-defined.

A complete Software Requirement Specifications must be:

- Clear

- Correct
- Consistent
- Coherent
- Comprehensible
- Modifiable
- Verifiable
- Prioritized
- Unambiguous
- Traceable
- Credible source

Category/Types of Software Requirements:

Broadly software requirements should be categorized in two categories:

Functional Requirements

Requirements, which are related to functional aspect of software fall into this category.

They define functions and functionality within and from the software system.

EXAMPLES -

- Search option given to user to search from various invoices.
- User should be able to mail any report to management.
- Users can be divided into groups and groups can be given separate rights.
- Should comply business rules and administrative functions.
- Software is developed keeping downward compatibility intact

Non-Functional Requirements

Requirements, which are not related to functional aspect of software, fall into this category. They are implicit or expected characteristics of software, which users make assumption of.

Non-functional requirements include -

- Security
- Logging
- Storage
- Configuration
- Performance
- Cost
- Interoperability
- Flexibility
- Disaster recovery
- Accessibility

What does “win-win” mean in the context of negotiation during the requirement engineering activity?

A “Win-Win situation is one in which the customer wins by getting the system or product that satisfies the majority of the customer’s needs and the software team wins by working to realistic and achievable budgets and deadlines.

Software System Analyst:

System analyst in an IT organization is a person, who analyzes the requirement of proposed system and ensures that requirements are conceived and documented properly and accurately. Role of an analyst starts during Software Analysis Phase of SDLC. It is the responsibility of analyst to make sure that the developed software meets the requirements of the client.

System Analysts have the following responsibilities:

- Analyzing and understanding requirements of intended software
- Understanding how the project will contribute to the organizational objectives
- Identify sources of requirement

- Validation of requirement
- Develop and implement requirement management plan
- Documentation of business, technical, process, and product requirements
- Coordination with clients to prioritize requirements and remove ambiguity
- Finalizing acceptance criteria with client and other stakeholders

Software Measures:

To assess the quality of the engineered product or system and to better understand the models that are created, some measures are used. These measures are collected throughout the software development life cycle with an intention to improve the software process on a continuous basis. Measurement helps in estimation, quality control, productivity assessment and project control throughout a software project. Also, measurement is used by software engineers to gain insight into the design and development of the work products. In addition, measurement assists in strategic decision-making as a project proceeds.

Software measurements are of two categories, namely, direct measures and indirect measures. Direct measures include software processes like cost and effort applied and products like lines of code produced, execution speed, and other defects that have been reported. Indirect measures include products like functionality, quality, complexity, reliability, maintainability, and many more.

Generally, software measurement is considered as a management tool which if conducted in an effective manner, helps the project manager and the entire software team to take decisions that lead to successful completion of the project.

A measurement is an indication of the size, quantity, amount or dimension of a particular attribute of a product or process. For example the number of errors in a system is a measurement.

Measurement process is characterized by a set of five activities, which are listed below.

- **Formulation:** This performs measurement and develops appropriate metric for software under consideration.
- **Collection:** This collects data to derive the formulated metrics.
- **Analysis:** This calculates metrics and the use of mathematical tools.
- **Interpretation:** This analyzes the metrics to attain insight into the quality of representation.
- **Feedback:** This communicates recommendation derived from product metrics to the software team.

Note that collection and analysis activities drive the measurement process. In order to perform these activities effectively, it is recommended to automate data collection and analysis, establish guidelines and recommendations for each metric, and use statistical techniques to interrelate external quality features and internal product attributes.

Other objectives of using software measures are listed below.

- Establish the quality of the current product or process.
- To predict future qualities of the product or process.
- To improve the quality of a product or process.
- To determine the state of the project in relation to budget and schedule.

Software Metrics in Software Engineering:

Once measures are collected they are converted into metrics for use. IEEE defines metric as 'a quantitative measure of the degree to which a system, component, or process possesses a given attribute.' The goal of software metrics is to identify and control essential parameters that affect software development.

A Metric is a measurement of the degree that any attribute belongs to a system, product or process. For example the number of errors per person hours would be a metric.

Other objectives of using software metrics are listed below.

- Measuring the size of the software quantitatively.
- Measuring the level of complexity involved.
- Measuring the testing techniques.
- Specifying when to stop testing.
- Determining the date of release of the software.
- Estimating cost of resources and project schedule.

Software metrics help project managers to gain an insight into the efficiency of the software process, project, and product. This is possible by collecting quality and productivity data and then analyzing and comparing these data with past averages in order to know whether quality improvements have occurred. Also, when metrics are applied in a consistent manner, it helps in project planning and project management activity. For example, schedule-based resource allocation can be effectively enhanced with the help of metrics.

What are the benefits of metrics in software engineering?

- It makes the better control, planning and clear visibility.
- It helps to increase the production and quality.
- With the help of this we can measure the size of the software.
- We can find out the cost of developed software.
- With the help of software metrics we can find out the errors which creates problem on the first level of development life cycle.

Software Project Management:

The job pattern of an IT company engaged in software development can be seen split in two parts:

Software Creation
Software Project Management

Project Manager:

A software project manager is a person who undertakes the responsibility of executing the software project. Software project manager is thoroughly aware of all the phases of SDLC that the software would go through. The project manager may never directly involve in producing the end product but he controls and manages the activities involved in production.

A project manager closely monitors the development process, prepares and executes various plans, arranges necessary and adequate resources, maintain communication among all team members in order to address issues of cost, budget, resources, time, quality and customer satisfaction.

The role and responsibility of a software project manager:

Software project managers may have to do any of the following tasks:

Planning: This means putting together the blueprint for the entire project from ideation to fruition. It will define the scope, allocate necessary resources, propose the timeline, delineate the plan for execution, lay out a communication strategy, and indicate the steps necessary for testing and maintenance.

Leading: A software project manager will need to assemble and lead the project team, which likely will consist of developers, analysts, testers, graphic designers, and technical writers. This requires excellent communication, people and leadership skills.

Execution: The project manager will participate in and supervise the successful execution of each stage of the project. This includes monitoring progress, frequent team check-ins and creating status reports.

Time management: Staying on schedule is crucial to the successful completion of any project, but it’s particularly challenging when it comes to managing software projects because changes to the original plan are almost certain to occur as the project evolves. Software project managers must be experts in risk management

and contingency planning to ensure forward progress when roadblocks or changes occur.

Budget: Like traditional project managers, software project managers are tasked with creating a budget for a project, and then sticking to it as closely as possible, moderating spend and re-allocating funds when necessary.

Maintenance: Software project management typically encourages constant product testing in order to discover and fix bugs early, adjust the end product to the customer's needs, and keep the project on target. The software project manager is responsible for ensuring proper and consistent testing, evaluation and fixes are being mad.

How to manage a software project successfully?

A recent article in Forbes suggests that there are eight ways to improve and streamline the software project management process; these eight suggestions include:

- Take non-development work off your team's plate to let them focus on developing
- Motivating your team by sharing others' success stories—like those of tech giants, which will inspire and excite your team
- Identify the project requirements
- Break down the plan and give them specific daily tasks
- Get a qualified project manager
- Communication is the key
- Use a Project management tool
- Evaluation of project

Why is project management important?

Clearly defines the plan of the project before it begins: The importance of planning in project management cannot be ignored. The more complex project, the

more scope there is for chaos. One of project management's primary functions is to tame the chaos by mapping out a clear plan of the project from beginning to end.

Establishes an agreed schedule and plan: Schedules help to eliminate delays or overruns and provide a plan to be followed for all those involved with the project.

Creates a base for teamwork: People are required to work in a team on a project. This is due to team synergy benefits through the sharing and support of knowledge and skills. Bringing people together in this way inspires team members to collaborate on a successful project.

Helps to keep control of costs: Depending on the scope of the project, some projects can incur organisations significant costs. It is important therefore to keep on budget and to control spending. Project management greatly reduces the risk of budget overruns.

Quality is continuously managed: More so than ever, it is important to produce quality results. Project management helps to identify, manage and control quality. Quality results make clients happy, which is a win-win situation for all involved.

Project Management Tools:

The risk and uncertainty rises multifold with respect to the size of the project, even when the project is developed according to set methodologies. There are tools available, which aid for effective project management. A few described are:

Gantt Chart:

Gantt chart was devised by Henry Gantt (1917). It represents project schedule with respect to time periods. It is a horizontal bar chart with bars representing activities and time scheduled for the project activities.

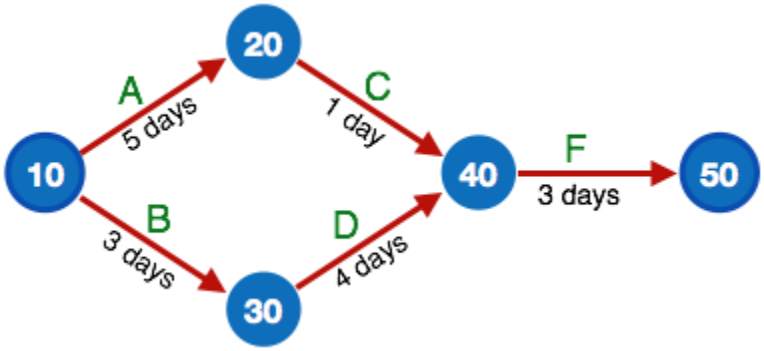
A Gantt chart, commonly used in project management, is one of the most popular and useful ways of showing activities (tasks or events) displayed against time. On the left of the chart is a list of the activities and along the top is a suitable time

scale. Each activity is represented by a bar; the position and length of the bar reflects the start date, duration and end date of the activity. This allows you to see at a glance:

months	1	2	3	4	5	6	7	8	9	10
project phases										
Planning										
Design										
Coding										
Testing										
Delivery										

Pert Chart:

Program Evaluation & Review Technique) (PERT) chart is a tool that depicts project as network diagram. It is capable of graphically representing main events of project in both parallel and consecutive ways. Events, which occur one after another, show dependency of the later event over the previous one.



Project Risk Management:

Risk management involves all activities pertaining to identification, analyzing and making provision for predictable and non-predictable risks in the project. Risk may include the following:

- Experienced staff leaving the project and new staff coming in.
- Change in organizational management.
- Requirement change or misinterpreting requirement.
- Under-estimation of required time and resources.

- Technological changes, environmental changes, business competition.

Categories of Risks:

There are, three related categories of risk:

1. Project Risks:

Risks which will affect the project schedule or resources. For example, stuff turnover, that is an experience team member of a project left the organization.

2. Product Risks: Risks that affect the quality or performance of the software being developed. For example a component isn't performing as expected.

3. Business Risks: Risks that affect the organization developing or procuring the software. For example, a competitor is developing a similar product that will challenge the product being developed.

Process of risks management:

Risks has to be listed and recovery actions has to be predetermined to avoid bigger impacts when development is underway. Following are the stages of risk management.

1. Risk Identification:

Identifying the possible risks in project, product and business.

2. Risk Analysis: After identifying the risks, the following consequences of the risk and what might cause the risk are then analyzed.

3. Risk Planning: After the risk is quite clear, plans has to be established by either avoiding it or minimizing it's effect or both.

4. Risk Monitoring: Though the risk has been identified, analyzed and planned well, with time, situation may change. Thus the risk has to be monitored to bring necessary changes to the plan of addressing the risk.

Figure: Risk Management