UNIT-1 COVERS

- software crisis
- Software characteristics
- Software quality attributes
- Software process model
- Process iteration
- Process activities
- Computer aided software engineering
- Functional and non-functional requirements
- User requirements
- System requirement
- Interface specification
- The software requirement documents
- Feasibility study
- Requirement elicitation and analysis
- Requirement validation and management

What is computer software?

- It is the product that software professionals build and then support over the long term.
- It encompasses programs that execute within a computer of any size and architecture.
- Generally it is defined as a set of instructions,data or programs used to operate computers and execute specific tasks.
- It can be thought of as the variable part of a computer.
- The two main categories of software are application software and system software.
- Other types of software include programming software which provides the programming tools software developers need middleware and driver software.
- Early software was written for specific computers and sold with the hardware it ran on then it began to be sold on floppy disks and later on CDs and DVDs.
- Today most software is purchased and directly downloaded over the internet.

WHAT IS SOFTWARE ENGINEERING?

- IEEE defines software engineering as: The application of a systematic,disciplined,quantifiable approach to the development, operation and maintenance of software;that is the application of engineering to software.
- The study of approaches as the above statement.

1.1 SOFTWARE CRISIS

- Software Crisis is a term used in computer science for the difficulty of writing useful and efficient computer programs in the required time.
- The software crisis was due to using the same workforce, same methods, same tools even though rapidly increasing in software demand, the complexity of software, and software challenges.

1.1 SOFTWARE CRISIS

Causes of Software Crisis:

- The cost of owning and maintaining software was as expensive as developing the software
- At that time Projects were running over-time
- At that time Software was very inefficient
- The quality of the software was low quality
- Software often did not meet user requirements
- The average software project overshoots its schedule by half
- At that time Software was never delivered
- Non-optimal resource utilization.
- Difficult to alter, debug, and enhance.
- The software complexity is harder to change.

1.1 SOFTWARE CRISIS

factors contributing to the software crisis:

- Poor project management.
- Lack of adequate training in software engineering.
- Less skilled project members.
- Low productivity improvements.

1.1 SOFTWARE CRISIS

Solution of Software Crisis:

One possible solution to a software crisis is Software Engineering because software engineering is a systematic,disciplined, and quantifiable approach. For preventing software crises, there are some guidelines:

- Reduction in software over budget.
- The quality of software must be high.
- Less time is needed for a software project.
- Experienced and skilled people working over the software project.
- Software must be delivered.
- Software must meet user requirements.

1.2 SOFTWARE CHARACTERISTICS

Software has characteristics that are considerably difficult than those of hardware:

1. **Software is developed or engineered;it is not manufactured in the classical sense:**

- Although some similarities exist between software development and hardware manufacturing, **the activities are functionally different**.
- In both activities, high quality is achieved **through good design**, but the manufacturing phase for hardware **can introduce quality problems** that are nonexistent for software.
- Software projects cannot be managed as if they were manufacturing projects.

1.2 SOFTWARE CHARACTERISTICS

2. Software doesnot wear out:

- Hardware exhibits relatively high failure rates early in its life ; defects are corrected and the failure rate drops to a steady state level for some period of time.
- As time passes failure rate rises again as hardware components suffer from the cumulative effects of dust,vibration,abuse,temperature extremes, and many other environmental maladies.
- Software is not susceptible to the environmental maladies that cause hardware to wear out.

1.2 SOFTWARE CHARACTERISTICS

3**. Although the industry is moving toward component-based construction, most software continues to be custom built:**

- As an engineering discipline evolves, a collection of standard design components is created.
- A software component should be designed and implemented so that it can be reused in many different programs.
- Modern reusable component encapsulate both data and the processing that is applied to the data, enabling the software engineer to create new applications from reusable components for interface construction

1.2 SOFTWARE CHARACTERISTICS

The characteristics of software include:

1. It is intangible, meaning it cannot be seen or touched.
2. It is non-perishable, meaning it does not degrade over time.
3. It is easy to replicate, meaning it can be copied and distributed easily.
4. It can be complex, meaning it can have many interrelated parts and features.
5. It can be difficult to understand and modify, especially for large and complex systems.
6. It can be affected by changing requirements, meaning it may need to be updated or modified as the needs of users change.
7. It can be affected by bugs and other issues, meaning it may need to be tested and debugged to ensure it works as intended.

1.3 Software Quality Attributes

- Software Quality Attributes are features that facilitate the measurement of a software product's performance by Software Testing professionals.
- They play a vital role in helping software architects guarantee that a software application will perform as expected based on the specifications provided by the client.
- **we will be focusing on the most important ones that we have mentioned below.**

1.3 SOFTWARE QUALITY ATTRIBUTES

1.3 SOFTWARE QUALITY ATTRIBUTES

Reliability:

- Reliability refers to the ability of software to perform its intended functions consistently and predictably, without failures or errors, under specified conditions and for a specified period.
- Reliable software minimizes unexpected crashes, errors, or downtime, providing a trustworthy user experience.

Usability:

- Usability relates to how easily users can interact with the software and accomplish their tasks effectively and efficiently.
- Usable software has a well-designed user interface, intuitive navigation, clear instructions, and a user-friendly overall experience, ensuring user satisfaction and productivity.

1.3 SOFTWARE QUALITY ATTRIBUTES

Performance:

- Performance refers to how well the software system responds and performs its functions under various workload conditions.
- It includes factors like response time, throughput, resource utilization, scalability, and efficiency.
- High-performance software delivers quick and efficient results, even with large data sets or high user loads.

Security:

- Security encompasses the measures taken to protect the software system and its data from unauthorized access, breaches, vulnerabilities, and attacks.
- Secure software ensures data confidentiality, integrity, availability, and compliance with privacy regulations, protecting against threats and maintaining user trust.

1.3 SOFTWARE QUALITY ATTRIBUTES

Maintainability:

- Maintainability refers to the ease with which software can be modified, updated, extended, or repaired over its lifecycle. Maintainable software has clear and well-structured code, modular design, proper documentation, and suitable development practices. It enables efficient maintenance, bug fixing, and future enhancements.

Portability:

- Portability relates to the software's ability to run on different platforms, operating systems, or environments without requiring significant modifications.
- Portable software is designed and developed to be platform-independent, ensuring easy deployment and adaptability across diverse environments.

1.3 SOFTWARE QUALITY ATTRIBUTES

Testability:

- Testability refers to how easily the software system can be tested to ensure its correctness, detect defects, and verify its behavior.
- Testable software has well-defined requirements, clear specifications, and a modular architecture, enabling efficient and effective testing processes.

Interoperability:

- Interoperability is the software's ability to interact, exchange data, and operate seamlessly with other systems, software components, or external interfaces.
- Interoperable software adheres to standard protocols, data formats, and communication mechanisms, facilitating integration and compatibility with other systems.

1.4 SOFTWARE PROESS MODEL

The software process:

- The process is a collection of activities, actions, and tasks that are performed when some product is to be created.
- In the context of software engineering a process is not a rigid prescription for how to build computer software. Rather it is an adaptable approach that enables the people doing the work to pick and choose the appropriate set of work actions and tasks.

1.4 SOFTWARE PROESS MODEL

- The generic process framework encompasses a set of umbrella activities that are applicable across the entire software process.
- A generic process framework for software engineering encompasses five activities:

Communication:

- Before any technical work can commence, it is critically important to communicate and collaborate with the customer and other stakeholder.
- The intent is to understand stakeholders objectives for the project and to gather requirements that help define software features and functions.

1.4 SOFTWARE PROESS MODEL

Planning:

- The planning activities creates a map that helps guide the team as it makes the journey.
- The map-called a software project plan-defines the software engineering work by describing the technical tasks to be conducted, the risk that are likely, the resources that will be required, the work products to be produced and work schedule.

1.4 SOFTWARE PROESS MODEL

Modeling:

It is required for the better understand the problem and how you are going to solve it.

To better understand software requirements and the design that will achieve those requirements.

Construction:

This activity combines code generation and the testing that is required to uncovers

Deployment:

The software is delivered to the customer who evalutes the delivered product feedback.

1.4 SOFTWARE PROESS MODEL

A software process model is an abstraction of the actual process, which is being described.

Each model represents a process from a specific perspective.

Basic software process models on which different type of software process models can be implemented:

- **A model will define the following:**
- The tasks to be performed
- The input and output of each task
- The pre and post-conditions for each task
- The flow and sequence of each task

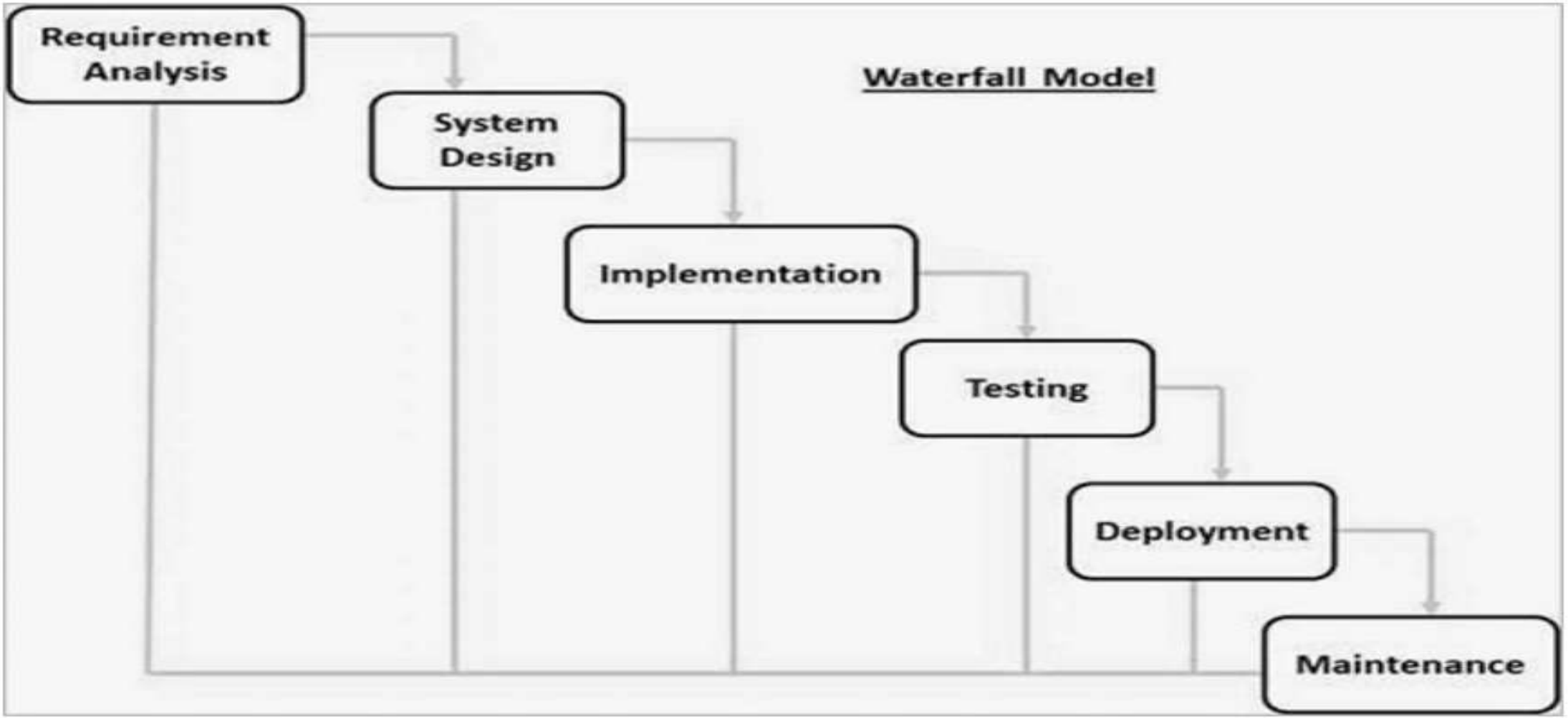
1.4 SOFTWARE PROESS MODEL

Types of software process models

Waterfall Model

- The waterfall model is a linear, sequential approach to the software development lifecycle (SDLC) that is popular in software engineering and product development.
- In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

1.4 SOFTWARE PROESS MODEL



1.4 SOFTWARE PROESS 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 document.
- **System Design** − The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
- **Implementation** − With inputs from the 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 non-functional 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.

1.4 SOFTWARE PROESS MODEL

Advantages:

- 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.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

1.4 SOFTWARE PROESS MODEL

Diadvantages:

- 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. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

1.4 SOFTWARE PROESS MODEL

When to use waterfall model??

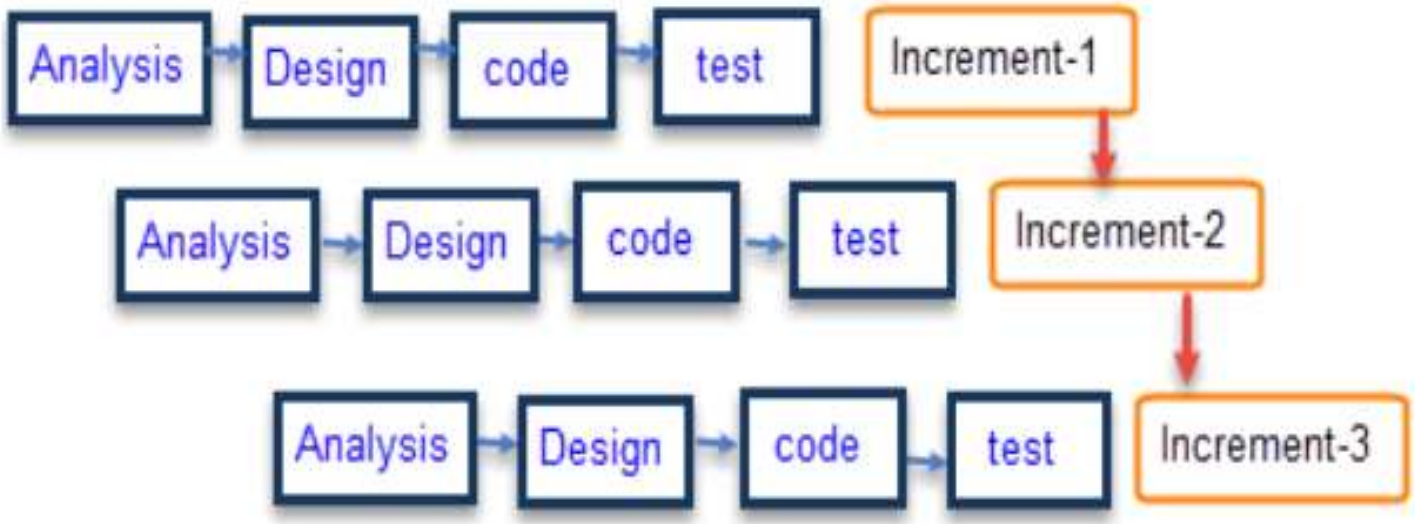
- Requirements are very well documented, clear and fixed.
- Product definition is stable.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- Ample resources with required expertise are available to support the product.
- The project is short.

1.4 SOFTWARE PROESS MODEL

2. Incremental model:

- Incremental Model is a process of software development where requirements are broken down into multiple standalone modules of software development cycle.
- Incremental development is done in steps from analysis design, implementation, testing/verification, maintenance.
- Each iteration passes through the **requirements, design, coding and testing phases**.

1.4 SOFTWARE PROESS MODEL



1.4 SOFTWARE PROESS MODEL

Advantages and Disadvantages of Incremental Model

•The software will be generated quickly during the software life cycle	•It requires a good planning designing
•It is flexible and less expensive to change requirements and scope	•Problems might cause due to system architecture as such not all requirements collected up front for the entire software lifecycle
•Throughout the development stages changes can be done	•Each iteration phase is rigid and does not overlap each other
•This model is less costly compared to others	•Rectifying a problem in one unit requires correction in all the units and consumes a lot of time

1.4 SOFTWARE PROESS MODEL

When to use incremental model?

- Requirements of the system are clearly understood
- When demand for an early release of a product arises
- When software engineering team are not very well skilled or trained
- When high-risk features and goals are involved
- Such methodology is more in use for web application and product based companies

1.4 SOFTWARE PROESS MODEL

Evolutionary process model:

- Evolutionary models are iterative type models.
- They allow to develop more complete versions of the software

Following are the evolutionary process models.

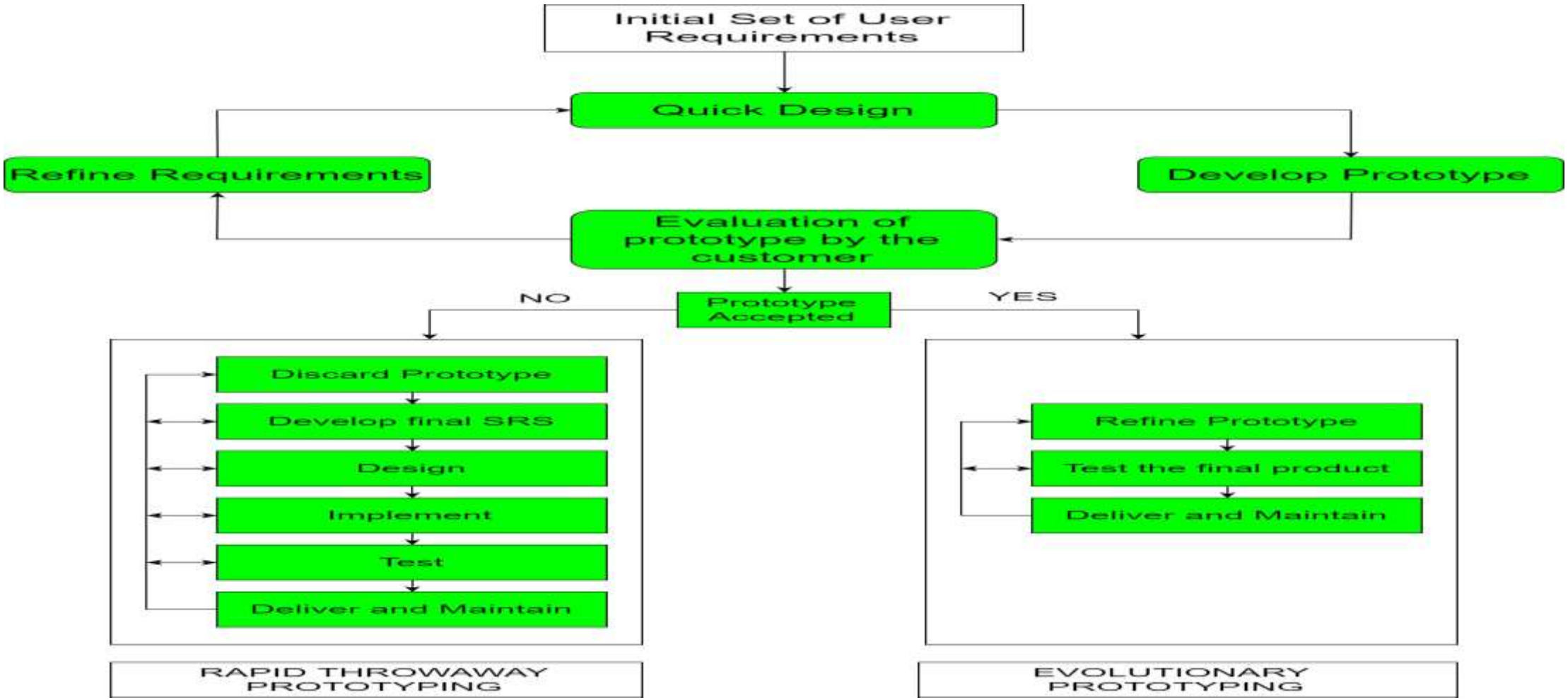
1. The prototyping model
2. The spiral model
3. Concurrent development model

1.4 SOFTWARE PROESS MODEL

The Prototyping model

- Prototype is defined as first or preliminary form using which other forms are copied or derived.
- Prototype model is a set of general objectives for software.
- It does not identify the requirements like detailed input, output.
- It is software working model of limited functionality.
- In this model, working programs are quickly produced.

1.4 SOFTWARE PROESS MODEL



1.4 SOFTWARE PROESS MODEL

Advantages of Prototyping Model

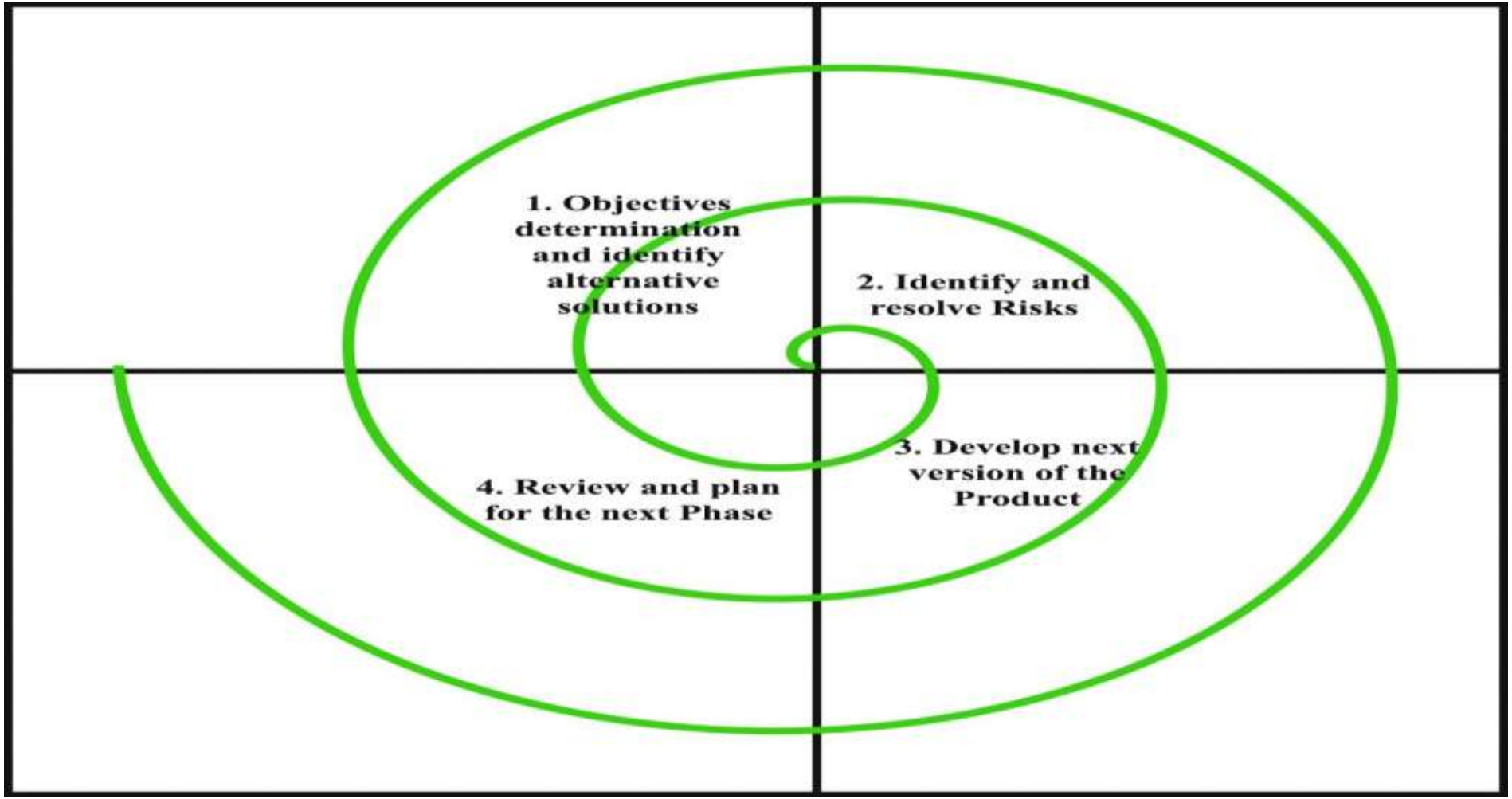
- The customers get to see the partial product early in the life cycle. This ensures a greater level of customer satisfaction and comfort.
- New requirements can be easily accommodated as there is scope for refinement.
- Errors can be detected much earlier thereby saving a lot of effort and cost, besides enhancing the quality of the software.
- Prototyping can help reduce the risk of project failure by identifying potential issues and addressing them early in the process.
- Prototyping can facilitate communication and collaboration among team members and stakeholders, improving overall project efficiency and effectiveness.

1.4 SOFTWARE PROESS MODEL

Disadvantages of the Prototyping Model

- Costly with respect to time as well as money.
- There may be too much variation in requirements each time the prototype is evaluated by the customer.
- Poor Documentation due to continuously changing customer requirements.
- It is very difficult for developers to accommodate all the changes demanded by the customer.
- There is uncertainty in determining the number of iterations that would be required before the prototype is finally accepted by the customer.

1.4 SOFTWARE PROESS MODEL



1.4 SOFTWARE PROESS MODEL

When to use prototyping model??

- The Prototyping Model should be used when the requirements of the product are not clearly understood or are unstable.
- The prototyping model can also be used if requirements are changing quickly.
- This model can be successfully used for developing user interfaces, high-technology software-intensive systems, and systems with complex algorithms and interfaces.
- The prototyping Model is also a very good choice to demonstrate the technical feasibility of the product.

1.4 SOFTWARE PROESS MODEL

2. Spiral model:

- The Spiral Model is a software development life cycle (SDLC) model that provides a systematic and iterative approach to software development.
- The Spiral Model is a risk-driven model, meaning that the focus is on managing risk through multiple iterations of the software development process.

1.4 SOFTWARE PROESS MODEL

Each phase of the Spiral Model is divided into four quadrants as shown in the above figure. The functions of these four quadrants are discussed below-

Objectives determination and identify alternative solutions:

- Requirements are gathered from the customers and the objectives are identified, elaborated, and analyzed at the start of every phase.

Identify and resolve Risks:

- During the second quadrant, all the possible solutions are evaluated to select the best possible solution.
- Then the risks associated with that solution are identified and the risks are resolved using the best possible strategy.
- At the end of this quadrant, the Prototype is built for the best possible solution.

1.4 SOFTWARE PROESS MODEL

Develop next version of the Product:

- During the third quadrant, the identified features are developed and verified through testing.
- At the end of the third quadrant, the next version of the software is available.

Review and plan for the next Phase:

- In the fourth quadrant, the Customers evaluate the so far developed version of the software.
- In the end, planning for the next phase is started.

1.4 SOFTWARE PROESS MODEL

Advantages of Spiral Model:

Below are some advantages of the Spiral Model.

1. **Risk Handling:** The projects with many unknown risks that occur as the development proceeds, in that case, Spiral Model is the best development model to follow due to the risk analysis and risk handling at every phase.
2. **Good for large projects:** It is recommended to use the Spiral Model in large and complex projects.
3. **Flexibility in Requirements:** Change requests in the Requirements at later phase can be incorporated accurately by using this model.
4. **Customer Satisfaction:** Customer can see the development of the product at the early phase of the software development and thus, they habituated with the system by using it before completion of the total product.

1.4 SOFTWARE PROESS MODEL

1. **Iterative and Incremental Approach**: The Spiral Model provides an iterative and incremental approach to software development, allowing for flexibility and adaptability in response to changing requirements or unexpected events.
2. **Emphasis on Risk Management**: The Spiral Model places a strong emphasis on risk management, which helps to minimize the impact of uncertainty and risk on the software development process.
3. **Improved Communication**: The Spiral Model provides for regular evaluations and reviews, which can improve communication between the customer and the development team.
4. **Improved Quality**: The Spiral Model allows for multiple iterations of the software development process, which can result in improved software quality and reliability

1.4 SOFTWARE PROESS MODEL

Disadvantages of Spiral Model:

Below are some main disadvantages of the spiral model.

1. **Complex:** The Spiral Model is much more complex than other SDLC models.
2. **Expensive:** Spiral Model is not suitable for small projects as it is expensive.
3. **Too much dependability on Risk Analysis:** The successful completion of the project is very much dependent on Risk Analysis. Without very highly experienced experts, it is going to be a failure to develop a project using this model.
4. **Difficulty in time management:** As the number of phases is unknown at the start of the project, so time estimation is very difficult.
5. **Complexity:** The Spiral Model can be complex, as it involves multiple iterations of the software development process.
6. **Time-Consuming:** The Spiral Model can be time-consuming, as it requires multiple evaluations and reviews.
7. **Resource Intensive:** The Spiral Model can be resource-intensive, as it requires a significant investment in planning, risk analysis, and evaluations.

1.4 SOFTWARE PROESS MODEL

When to use the Spiral Model?

- When a project is vast in software engineering, a spiral model is utilised.
- A spiral approach is utilised when frequent releases are necessary.
- When it is appropriate to create a prototype
- When evaluating risks and costs is crucial
- The spiral approach is beneficial for projects with moderate to high risk.
- The SDLC's spiral model is helpful when requirements are complicated and ambiguous.
- If modifications are possible at any moment
- When committing to a long-term project is impractical owing to shifting economic priorities.

1.4 SOFTWARE PROESS MODEL

- **Agile software Development Model:**
- The meaning of Agile is swift or versatile.
- "**Agile process model**" refers to a software development approach based on iterative development.
- Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning.
- Plans regarding the number of iterations, the duration and the scope of each iteration are clearly defined in advance.
- Each iteration is considered as a short time "frame" in the Agile process model, which typically lasts from one to four weeks.

1.4 SOFTWARE PROESS MODEL

The four values of Agile

- Individual interactions are more important than processes and tools.
- A focus on working software rather than thorough documentation.
- Collaboration instead of contract negotiations.
- A focus on responding to change.

1.4 SOFTWARE PROESS MODEL

The 12 principles of Agile

The Agile Manifesto also outlined **12 core principles** for the development process. They are as follows:

1.Satisfy customers through early and continuous delivery of valuable work.

2.Break big work down into smaller tasks that can be completed quickly.

3.Recognize that the best work emerges from self-organized teams.

4.Provide motivated individuals with the environment and support they need and trust them to get the job done.

5.Create processes that promote sustainable efforts.

6.Maintain a constant pace for completed work.

1.4 SOFTWARE PROESS MODEL

7. Welcome changing requirements, even late in a project.
8. Assemble the project team and business owners on a daily basis throughout the project.
9. Have the team reflect at regular intervals on how to become more effective, then tune and adjust behavior accordingly.
10. Measure progress by the amount of completed work.
11. Continually seek excellence.
12. Harness change for a competitive advantage.

1.4 SOFTWARE PROESS MODEL

The Agile software development cycle

- The Agile software development cycle can be broken down into the following six steps:
- concept
- inception
- iteration/construction
- release
- production
- retirement

1.4 SOFTWARE PROESS MODEL

- The first step, *concept*, involves the identification of business opportunities in each potential project as well as an estimation of the time and work that will be required to complete the project.
- During the second step, *inception*, team members are identified, funding is established and the initial requirements are discussed with the customer.
- The third step, *iteration/construction*, is when teams start creating working software based on requirements and continuous feedback.
- The fourth step, *release*, involves final QA testing, resolution of any remaining defects, finalization of the system and user documentation and, at the end, release of the final iteration into production.
- After the release, the fifth step, *production*, focuses on the ongoing support necessary to maintain the software.
- removal of a system release from production, and occasionally even the complete system itself.

1.4 SOFTWARE PROESS MODEL

Advantages :

Faster time-to-market: Agile's iterative approach enables faster delivery of working software increments, allowing for early and frequent releases.

Adaptability to change: Agile's flexible nature accommodates changing requirements and priorities, reducing the risk of delivering outdated or irrelevant software.

Enhanced customer satisfaction: Continuous customer collaboration ensures the software aligns closely with their needs, increasing customer satisfaction and engagement.

Improved quality: Regular feedback and testing throughout the development process help identify and address issues early, leading to higher-quality software.

Increased team collaboration and motivation: Agile methodologies foster teamwork, communication, and shared accountability, leading to more engaged and motivated development teams.

1.4 SOFTWARE PROESS MODEL

Types of Agile methodologies

The most widely used Agile methods include the following:

Scrum

Lean software development

Extreme programming

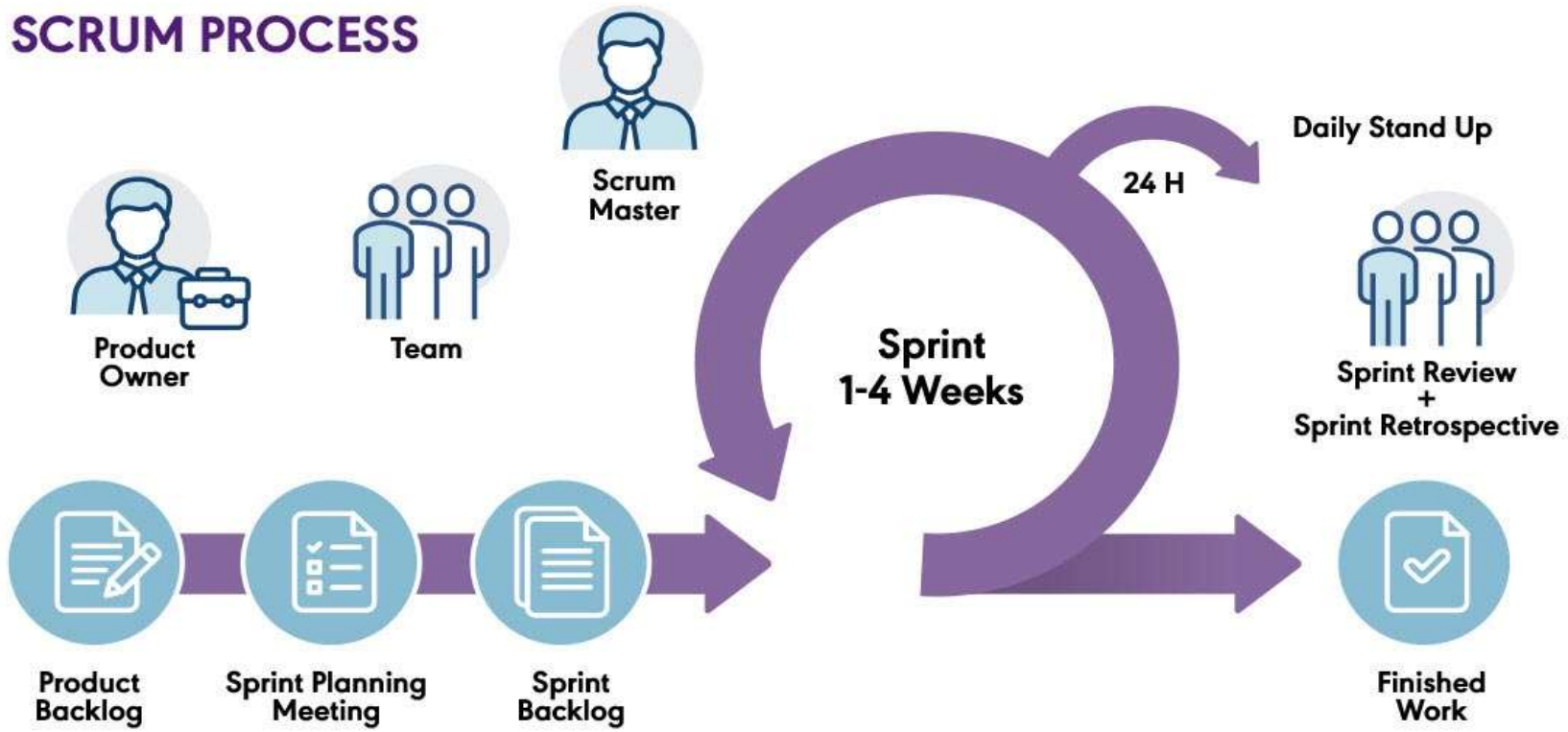
Crystal

Kanban

Dynamic systems development method

Feature-driven development

1.4 SOFTWARE PROESS MODEL



1.4 SOFTWARE PROESS MODEL

Scrum:

Scrum is an Agile framework for managing and organizing complex projects, primarily in the field of software development.

Key Concepts in Scrum:

Scrum Team: A Scrum team is a cross-functional group that includes the development team, product owner, and Scrum Master. The team works together to deliver the product increment.

Product Backlog: The product backlog is a prioritized list of requirements, features, and user stories that represents the work to be done. It is continuously refined and adjusted based on feedback and changing priorities.

Sprints: Sprints are time-boxed iterations during which the Scrum team works to deliver a potentially shippable product increment. Sprints are usually two to four weeks long and have a fixed duration.

1.4 SOFTWARE PROESS MODEL

Sprint Planning: At the start of each sprint, the Scrum team conducts a sprint planning meeting. They collaborate to determine which items from the product backlog will be worked on during the sprint and create a sprint backlog.

Daily Scrum: The daily scrum is a short daily meeting where the development team members synchronize their work. Each team member provides an update on their progress, discusses any obstacles or challenges, and plans their activities for the day.

Sprint Review: At the end of each sprint, a sprint review is held to demonstrate the completed work to stakeholders. Feedback is gathered, and the product backlog is adjusted based on the input received.

Sprint Retrospective: The sprint retrospective takes place after the sprint review. The Scrum team reflects on their processes, identifies areas for improvement, and discusses actions to enhance productivity and effectiveness in future sprints.

Scrum Master: The Scrum Master serves as a facilitator and coach for the Scrum team. They ensure that the Scrum framework is followed, remove any obstacles hindering progress, and promote a collaborative and productive work environment.

1.4 SOFTWARE PROESS MODEL

Benefits of Scrum:

Transparency: Scrum provides transparency into the project's progress, challenges, and priorities through regular meetings and shared artifacts.

Adaptability: Scrum allows for changing requirements and priorities, enabling the team to respond quickly and deliver value incrementally.

Improved collaboration: Scrum promotes collaboration and effective communication within the team and with stakeholders, enhancing understanding and alignment.

Higher quality: Scrum's iterative nature ensures frequent testing, feedback, and integration, leading to higher-quality deliverables.

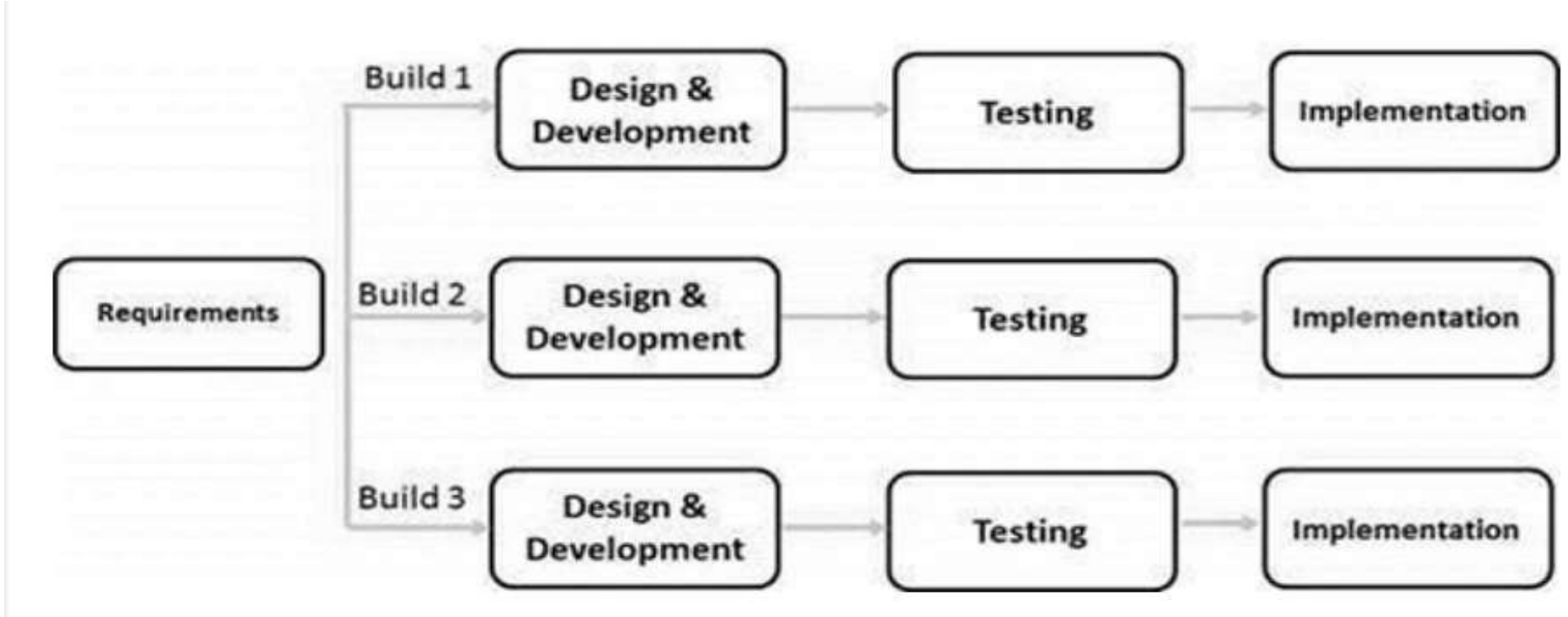
Faster time-to-market: With short, time-boxed sprints, Scrum enables faster delivery of valuable increments, allowing for rapid response to market demands.

1.5 PROCESS ITERATION

Iterative model:

- 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.
- 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.

1.5 PROCESS ITERATION



1.5 PROCESS ITERATION

When to use??

- Requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some functionalities or requested enhancements may evolve with time.
- There is a time to the market constraint.
- A new technology is being used and is being learnt by the development team while working on the project.
- Resources with needed skill sets are not available and are planned to be used on contract basis for specific iterations.
- There are some high-risk features and goals which may change in the future.

1.5 PROCESS ITERATION

Pros:

- Some working functionality can be developed quickly and early in the life cycle.
- Results are obtained early and periodically.
- Parallel development can be planned.
- Progress can be measured.
- Less costly to change the scope/requirements.
- Testing and debugging during smaller iteration is easy.
- Risks are identified and resolved during iteration; and each iteration is an easily managed milestone.

1.5 PROCESS ITERATION

- Easier to manage risk - High risk part is done first.
- With every increment, operational product is delivered.
- Issues, challenges and risks identified from each increment can be utilized/applied to the next increment.
- Risk analysis is better.
- It supports changing requirements.
- Initial Operating time is less.
- Better suited for large and mission-critical projects.
- During the life cycle, software is produced early which facilitates customer evaluation and feedback.

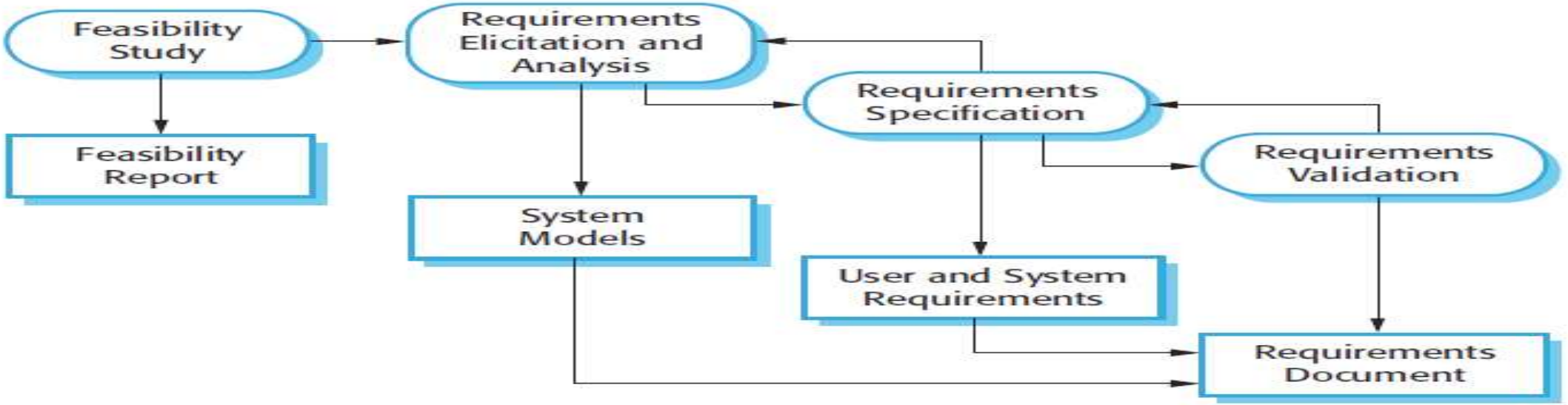
1.5 PROCESS ITERATION

Cons:

- More resources may be required.
- Although cost of change is lesser, but it is not very suitable for changing requirements.
- More management attention is required.
- System architecture or design issues may arise because not all requirements are gathered in the beginning of the entire life cycle.
- Defining increments may require definition of the complete system.
- Not suitable for smaller projects.
- Management complexity is more.
- End of project may not be known which is a risk.
- Highly skilled resources are required for risk analysis.
- Projects progress is highly dependent upon the risk analysis phase.

1.6 PROCESS ACTIVITIES

- The four basic process activities of specification, development, validation, and evolution are organized differently in different development processes.
- Software specification or requirements engineering is the process of understanding and defining what services are required from the system and identifying the constraints on the system's operation and development.
- Requirements engineering is a particularly critical stage of the software process as errors at this stage unavoidably lead to later problems in the system design and



1.6 PROCESS ACTIVITIES

1. Software specification:

Software specification or requirement engineering is the process of understanding and defining what services are required from the system and identifying the constraint on the systems operation and development.

There are four main activities in the requirements engineering process:

1. Feasibility study

- An estimate is made of whether the identified user needs may be satisfied using current software and hardware technologies.
- The study considers whether the proposed system will be cost-effective from a business point of view and if it can be developed within existing budgetary constraints. A feasibility study should be relatively cheap and quick. The result should inform the decision of whether or not to go ahead with a more detailed analysis (feasibility report).

2. Requirements elicitation:

This is the process of deriving the system requirements through observation of existing systems, discussions with potential users and buyer, task analysis. This may involve the development of one or more system models and prototypes. These help the system developer understand the system to be specified.

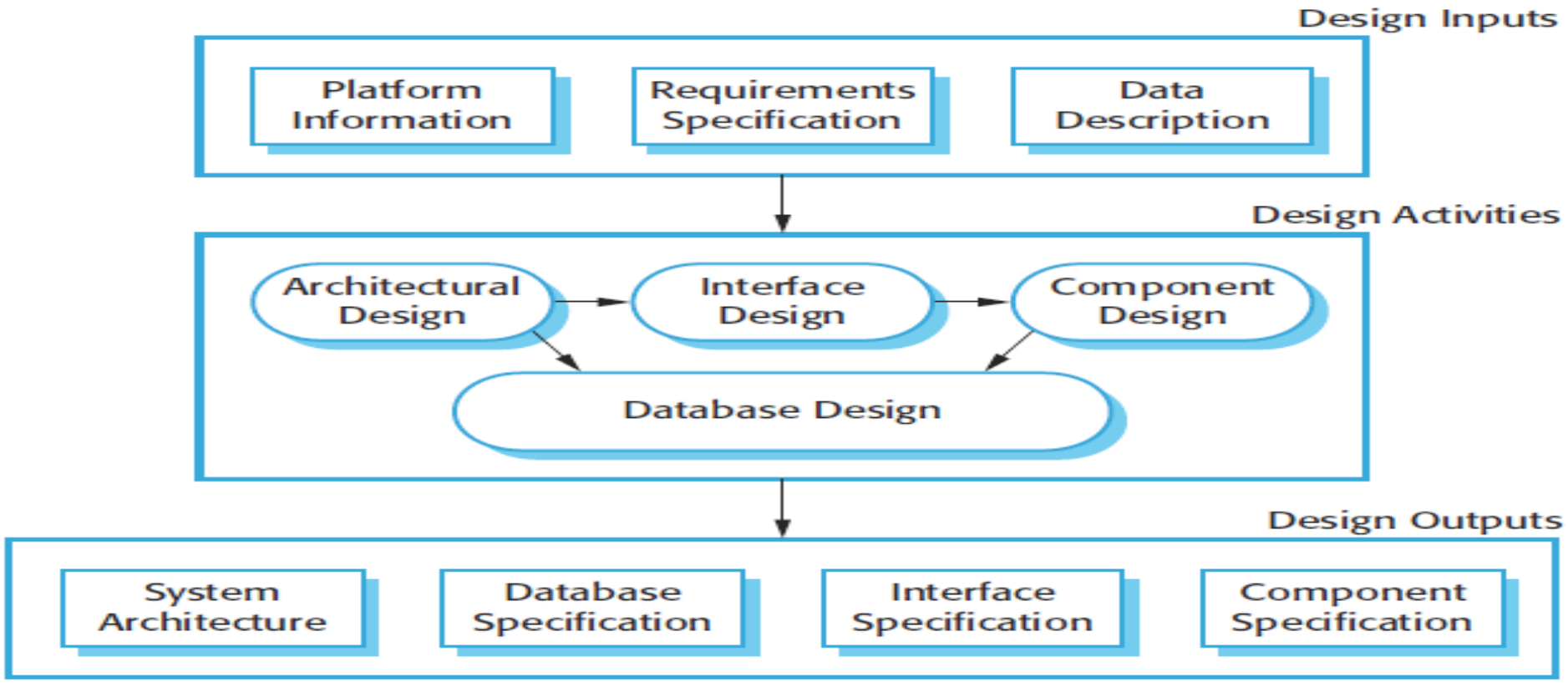
1.6 PROCESS ACTIVITIES

3. Requirements specification: Requirements specification is the activity of translating the information gathered during the analysis activity into a document that defines a set of requirements. Two types of requirements may be included in this document. User requirements are abstract statements of the system requirements for the customer and end-user of the system; System requirements are a more detailed description of the functionality to be provided.

4. Requirements validation: This activity checks the requirements for realism , consistency, and completeness. During this process, errors in the requirements document are inevitably discovered. It must then be modified to correct these problems.

1.6 PROCESS ACTIVITIES

2. Software design and implementation



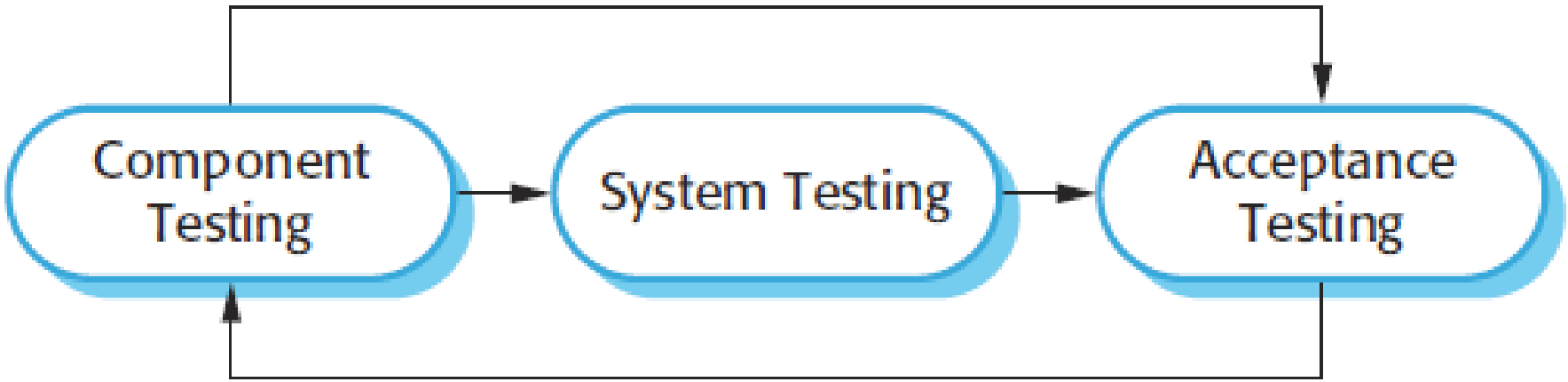
1.6 PROCESS ACTIVITIES

The implementation stage of software development is the process of converting a system specification into an executable system. It always involves processes of software design and programming.

There are four activities that may be part of the design process for information systems as shown in the previous figure:

- **Architectural design**, where the software engineer identifies the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships, and how they are distributed.
- **Interface design**, where the software engineer defines the interfaces between system components. This interface specification must be unambiguous. Once interface specifications are agreed, the components can be designed and developed concurrently.
- **Component design**, where the software engineer takes each system component and design how it will operate. This may be a simple statement of the expected functionality to be implemented, with the specific design left to the programmer. Alternatively, it may be a list of changes to be made to a reusable component or a detailed design model
- **Database design**, where the software engineer designs the system data structures and how these are to be represented in a database. The work here depends on whether an existing database is to be reused or a new database is to be created.

1.6 PROCESS ACTIVITIES



1.6 PROCESS ACTIVITIES

Software validation

- Software validation or, more generally, verification and validation (V&V) is intended to show
- that a system both conforms to its specification and that it meets the expectations of the system
- customer. Program testing, where the system is executed using simulated test data, is the principal
- validation technique.

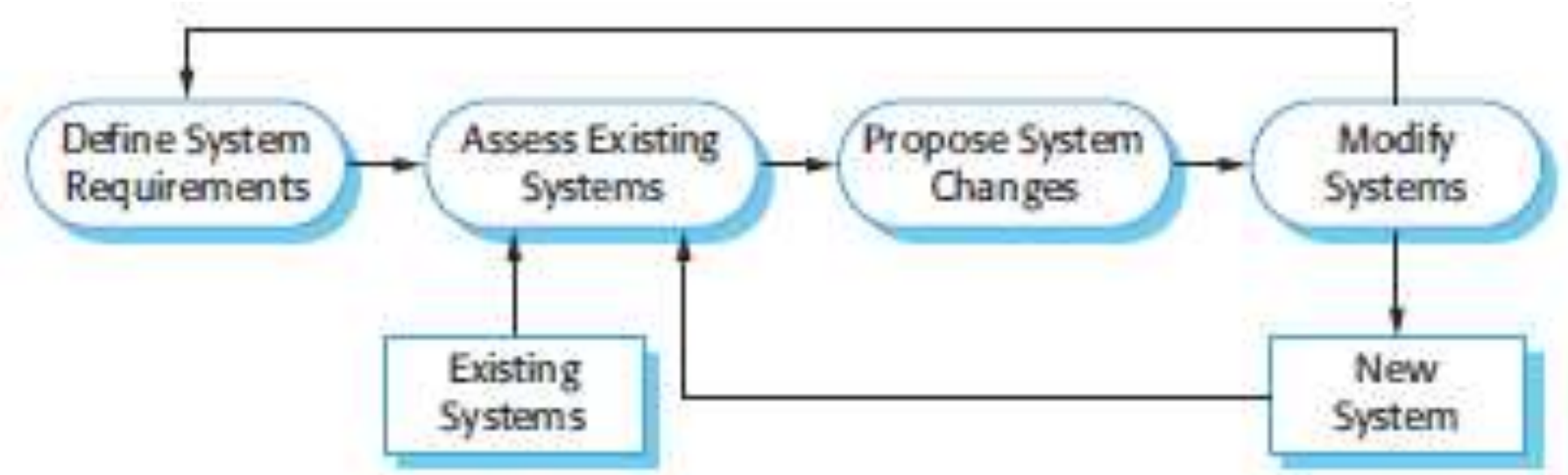
The stages in the testing process are:

Component (or unit) testing: Individual components are tested to ensure that they operate correctly. Each component is tested independently, without other system components. Components may be simple entities such as functions or object classes, or may be coherent groupings of these entities.

- **System testing:** System components are integrated to create a complete system. This process is
- concerned with finding errors that result from not expected interactions between components and
- component interface problems.
- **Acceptance testing:** This is the final stage in the testing process before the system is accepted for operational use. The system is tested with data supplied by the system customer rather than with simulated test data.

1.6 PROCESS ACTIVITIES

Software evolution



1.6 PROCESS ACTIVITIES

Software evolution

- The flexibility of software systems is one of the main reasons why more and more software is being incorporated in large, complex systems.
- Once a decision has been made to manufacture hardware, it is very expensive to make changes to the hardware design.
- However, changes can be made to software at any time during or after the system development.
- Even extensive changes are still much cheaper than corresponding changes to system hardware.

1.7 COMPUTER-AIDED SOFTWARE ENGINEERING

- Case Tools, also known as Computer-Aided Software Engineering Tools, refer to a set of software applications that assist software developers, analysts, and designers in the development and maintenance of software systems.
- These tools provide automated support throughout the various stages of the software development life cycle (SDLC), including requirements gathering, analysis, design, coding, testing, and maintenance.
- The primary purpose of Case Tools is to improve productivity, enhance the quality of software systems, and facilitate effective collaboration among team members.

Some common types of Case Tools include:

Diagramming Tools: These tools help in creating visual representations of software systems, such as flowcharts, data flow diagrams, entity-relationship diagrams, and UML diagrams. They aid in understanding the structure and behavior of the system.

1.7 COMPUTER-AIDED SOFTWARE ENGINEERING

Requirements Management Tools: These tools assist in gathering, documenting, and managing software requirements. They allow users to define and track requirements, trace dependencies, and ensure that all necessary features and functionalities are included.

Modeling Tools: Modeling tools enable developers to create detailed models of the software system, including its architecture, components, and interactions. These models serve as a blueprint for the implementation phase and aid in understanding system complexities.

Code Generation Tools: Code generation tools automate the process of generating code from high-level models or specifications**.**

Testing Tools: Testing tools help in automating the testing process by providing features for test case generation, test execution, and result analysis.

Configuration Management Tools: These tools assist in managing changes to software artifacts, including version control, configuration tracking, and release management.

1.7 COMPUTER-AIDED SOFTWARE ENGINEERING

Some example of CASE tools:

- **Microsoft Visio**: Visio is a diagramming tool that allows users to create a wide range of visual representations, including flowcharts, process diagrams, UML diagrams, and more. It is widely used for requirements gathering, system design, and documentation.
- **JIRA**: JIRA is primarily an issue tracking and project management tool but can also be used as a Case Tool for software development.
- **Selenium**: Selenium is an open-source testing framework that provides tools for automating web application testing.
- **Git:** Git is a widely used distributed version control system that allows developers to manage source code efficiently.

1.8 FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

Functional requirement:

- These are the statement of services the system should provide, how the system should react to particular inputs, and how the system should behave in particular situation.
- The functional requirement should for a system describe what the system should do.
- When expressed as user requirements, functional requirements are usually described in an abstract way that can be understood by system users.

1.8 FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

- For example:

Here are examples of functional requirement for the MHC-PMS system, used to maintain information about patients receiving treatment for mental health problem:

1. A user shall be able to search the appointments list for all clinics
2. The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day.
3. Each staff member using the system shall be uniquely identified by his or her eight-digit employee number

1.8 FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

Non-functional requirement:

- Non-functional requirement are the requirement that are not directly concerned with the specific services delivered by the system to its users.
- Non-functional requirements such as performance, security or availabity, usually specify characteristicsf of the system as a whole.
- System users can usually find ways to work around a system function that doesnot really meet their needs.
- However failing to meet a non-functional requirement can mean that the whole system is unusable.
- For example: if an aircraft system does not meet its reliability requirements, it will not be certified as safe for operation.

1.8 FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

For example:

Performance: Dr. Smith logs into the system and expects it to respond quickly. She navigates through different sections, such as patient records, appointment scheduling, and treatment plans, and experiences minimal delays in loading and processing data. The system meets the nonfunctional requirement of efficient performance, ensuring a smooth user experience for Dr. Smith.

Usability: Dr. Smith finds the system's user interface intuitive and user-friendly. She can easily navigate through various features, input patient information, and view appointment schedules without requiring extensive training. The system fulfills the nonfunctional requirement of usability, making it convenient for Dr. Smith to perform her tasks efficiently.

1.8 FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

Reliability: Dr. Smith relies on the MHC-PMS system for managing appointments, accessing patient records, and creating treatment plans. She expects the system to be available and reliable, with minimal downtime. The system meets the nonfunctional requirement of reliability, ensuring high availability and data integrity, so Dr. Smith can carry out her clinical tasks without disruptions.

Compatibility: Dr. Smith uses a specific browser and operating system in her clinic. The MHC-PMS system is designed to be compatible with her preferred browser and operating system, ensuring that she can access and use the system seamlessly. This meets the nonfunctional requirement of compatibility, providing a consistent user experience across different platforms.

1.9 USER REQUIREMENTS

- User requirements, also known as user needs or user stories, are statements that describe the needs, expectations, and desired functionalities from the perspective of the system's end users or stakeholders.
- These requirements serve as a foundation for system design and development

Here's a breakdown of user requirements:

User-Centric Perspective: User requirements focus on the needs and goals of the users or stakeholders who will interact with the system. They reflect the user's perspective and describe what they want the system to do or enable them to achieve.

1.9 USER REQUIREMENTS

User Goals and Objectives: User requirements reflect the goals and objectives of the users or stakeholders. They articulate what users aim to accomplish by using the system, such as improving efficiency, automating processes, enhancing collaboration, or simplifying complex tasks.

User Experience and Usability: User requirements often address the usability and user experience aspects of the system. They describe the desired ease of use, intuitiveness, responsiveness, and overall satisfaction that users expect when interacting with the system.

1.9 USER REQUIREMENTS

Context and Constraints: User requirements may also take into account the specific context, constraints, or limitations of the users or stakeholders. These can include factors such as regulatory compliance, security requirements, integration with existing systems, or technical capabilities of the user's environment.

User Acceptance Criteria: User requirements may be accompanied by acceptance criteria, which define measurable conditions or criteria that must be met for the user requirement to be considered satisfied. Acceptance criteria help ensure that the system meets the users' expectations and desired outcomes.

1.10 SYSTEM REQUIREMENTS

- System requirements define the capabilities, behaviors, and characteristics that a system must possess to meet the needs and expectations of its users and stakeholders.
- They outline the functional and nonfunctional aspects of the system and serve as a basis for system design, development, and testing. Here's an explanation of system requirements:

Functional Requirements: Functional requirements describe the specific functionalities and features that the system should have. They outline what the system should do and how it should behave. Examples of functional requirements include:

"The system shall allow users to create and edit documents."

"The system shall provide a search functionality to retrieve relevant information."

"The system shall support online payment processing for customer transactions."

1.10 SYSTEM REQUIREMENTS

Nonfunctional Requirements:

- Nonfunctional requirements specify the qualities, characteristics, and constraints that the system should exhibit. They focus on aspects such as performance, security, usability, reliability, and scalability. Examples of nonfunctional requirements include:
- "The system shall respond to user actions within 2 seconds under normal operating conditions."
- "The system shall adhere to industry-standard security practices, including encryption of sensitive data."
- "The system shall provide an intuitive and user-friendly interface, requiring minimal training for users to navigate."

1.10 SYSTEM REQUIREMENTS

Constraints: Constraints are limitations or conditions that must be considered during system development. They may arise from technical, operational, regulatory, or budgetary factors. Examples of constraints include:

"The system must be compatible with Windows 10 operating system."

"The system must comply with GDPR (General Data Protection Regulation) regulations."

"The system development budget is limited to $100,000."

1.10 SYSTEM REQUIREMENTS

Performance Requirements: Performance requirements specify the system's expected performance characteristics, such as response time, throughput, and resource utilization. They define benchmarks against which the system's performance can be evaluated and measured.

Design Constraints: Design constraints outline specific design choices or standards that must be followed during system development. These constraints ensure consistency, interoperability, and maintainability. Examples include adherence to coding conventions, use of specific software frameworks, or integration with existing systems.

User Requirements: User requirements capture the needs, expectations, and desired functionalities from the perspective of the system's end users or stakeholders. They describe what users want the system to accomplish or enable them to do. User requirements provide user-centric input to guide system design and development.

1.11 INTERFACE SPECIFICATION

- A point where two systems, subjects, organizations etc. meet and interact is known as interface.
- An interface specification, in the context of software development, refers to a document or set of guidelines that define how different software components or systems communicate and interact with each other.
- The interface specification serves as a contract between the components or systems involved, ensuring that they understand how to exchange information and collaborate effectively.
- It provides a clear definition of the inputs, outputs, and behaviors expected from each component, allowing developers to build interoperable software.

1.11 INTERFACE SPECIFICATION

Here are some common elements typically included in an interface specification:

Methods and Functions: The specification defines the available methods or functions that can be invoked by the calling component, along with their input parameters, return types, and exceptions.

Data Structures: It describes the data structures used by the interface, including the format, data types, and any constraints or validation rules.

Protocols and Communication Mechanisms: The specification may outline the protocols or communication mechanisms to be used for exchanging data between components, such as HTTP, REST, messaging queues, or other custom protocols.

Error Handling: It defines the error conditions that may occur during communication and how they should be handled, including error codes, error messages, and potential recovery strategies.

1.11 INTERFACE SPECIFICATION

Security Considerations: The specification may address security measures, such as authentication, authorization, encryption, or other access control mechanisms, to ensure secure communication between components.

Performance and Scalability: It may provide guidelines or requirements related to performance and scalability aspects, specifying expected response times, throughput, or maximum load the interface should handle.

Versioning and Compatibility: If the interface is subject to changes or updates over time, the specification may include guidelines on versioning and backward compatibility to ensure smooth transitions and minimize disruptions.

Documentation and Examples: The specification should be accompanied by comprehensive documentation and examples illustrating how to use the interface effectively.

1.12 THE SOFTWARE REQUIREMENT DOCUMENTS

The software requirement document sometimes called the software requirements specification or SRS is an **official statement of what the system developers should implement.**

It serves as a communication tool between stakeholders, including clients, project managers, and development teams, to ensure a shared understanding of the desired system behavior and features.

The requirement document has a **diverse set of users,** ranging from the senior management of the organization that is paying for the system to the engineers responsible for developing the software.

For example:

System customers:

Specify the requirements and read them to check that they meet their needs. Customer specify changes to the requirements.

1.12 THE SOFTWARE REQUIREMENT DOCUMENTS

Managers:

Use the requirements document to prepare a proposal for the development and to plan the system development process.

System engineers:

Use the requirements to understand what system is to be developed.

System Test Engineer:

Use the requirements to develop validation tests for the system

System Maintenance Engineers:

Use the requirements to understand the system and the relationship between its parts.

1.11 THE SOFTWARE REQUIREMENT DOCUMENTS

A requirement specification document typically includes the following sections:

Introduction: This section provides an overview of the document, its purpose, and the intended audience. It may also include background information about the project, stakeholders, and any relevant industry standards or regulations.

Scope: The scope section defines the boundaries and context of the software system. It outlines the specific functionalities and features to be included in the system and identifies any explicit exclusions or limitations.

Functional Requirements: This section describes the specific functions, tasks, and behaviors that the software system should perform. It defines the inputs, outputs, and interactions between the system and its users, including actors and use cases. It may also include diagrams, such as flowcharts or sequence diagrams, to illustrate system behavior.

1.11 THE SOFTWARE REQUIREMENT DOCUMENTS

Non-Functional Requirements: Non-functional requirements specify the quality attributes or constraints of the software system, rather than its specific functionalities. This section includes requirements related to performance, reliability, security, usability, maintainability, scalability, compatibility, and other aspects that affect the overall system behavior.

User Interface (UI) Design: If the system has a graphical user interface (GUI), this section outlines the UI design guidelines, including layouts, navigation, visual elements, and interaction patterns.

1.12 THE SOFTWARE REQUIREMENT DOCUMENTS

System Architecture: This section provides an overview of the system's high-level architecture, including the main components, modules, and their interactions. It may include diagrams, such as block diagrams or deployment diagrams, to illustrate the system structure.

Data Requirements: This section specifies the data entities, their attributes, relationships, and any constraints or validation rules. It may also outline the data storage and retrieval mechanisms, such as databases or external data sources.

Assumptions and Constraints: This section documents any assumptions made during the requirement gathering process and any constraints or limitations that need to be considered during the system development or implementation.

1.12 THE SOFTWARE REQUIREMENT DOCUMENTS

Dependencies: This section identifies any external dependencies or integrations required for the system, such as third-party libraries, APIs, or hardware devices.

Testing and Validation: This section outlines the testing requirements and strategies to validate the system against the specified requirements. It may include test scenarios, acceptance criteria, and performance benchmarks.

Project Timeline and Deliverables: This section provides an overview of the project timeline, milestones, and deliverables associated with the requirements. It helps in planning and tracking the project progress.

Glossary: The glossary section includes a list of key terms and definitions used throughout the document to ensure a common understanding of the terminology.

1.13 FEASIBILITY STUDY

Feasibility Study in Software Engineering is a study to evaluate feasibility of proposed project or system.

feasibility study is the feasibility analysis or it is a measure of the software product in terms of how much beneficial product development will be for the organization in a practical point of view.

Types of Feasibility Study :

Technical Feasibility –

In Technical Feasibility current resources both hardware software along with required technology are analyzed/assessed to develop project.

1.13 FEASIBILITY STUDY

Along with this, feasibility study also analyzes technical skills and capabilities of technical team, existing technology can be used or not, maintenance and up-gradation is easy or not for chosen technology etc.

Operational Feasibility –

In Operational Feasibility degree of providing service to requirements is analyzed along with how much easy product will be to operate and maintenance after deployment.

Along with this other operational scopes are determining usability of product, Determining suggested solution by software development team is acceptable or not etc.

1.13 FEASIBILITY STUDY

Economic Feasibility –

In Economic Feasibility study cost and benefit of the project is analyzed.

Means under this feasibility study a detail analysis is carried out what will be cost of the project for development which includes all required cost for final development like hardware and software resource required, design and development cost and operational cost and so on.

Legal Feasibility –

In Legal Feasibility study project is analyzed in legality point of view.

This includes analyzing barriers of legal implementation of project, data protection acts or social media laws, project certificate, license, copyright etc.

1.13 FEASIBILITY STUDY

Schedule Feasibility –

In Schedule Feasibility Study mainly timelines/deadlines is analyzed for proposed project which includes how many times teams will take to complete final project which has a great impact on the organization as purpose of project may fail if it can’t be completed on time.

1.13 FEASIBILITY STUDY

Schedule Feasibility –

In Schedule Feasibility Study mainly timelines/deadlines is analyzed for proposed project which includes how many times teams will take to complete final project which has a great impact on the organization as purpose of project may fail if it can’t be completed on time.

1.14 Requirement Elicitation and Analysis

- This is also known as the **gathering of requirements**. Here, requirements are identified with the help of customers and existing systems processes, if available.
- Requirements elicitation and analysis may involve a variety of different kinds of people in an organization.
- A system stakeholder is anyone who should have some direct or indirect influence on the system requirements.
- Stakeholders include end users who will interact with the system and anyone else in an organization who will be affected by it.
- Other system stakeholders might be engineers who are developing or maintaining other related system, business managers, domain experts and trade union representatives.

1.14 Requirement Elicitation and Analysis

The process activities are:

1. **Requirement discovery**:

- This is the process of interacting with stakeholders of the system to discover their requirements.
- Domain requirements from stakeholdes and documentation are also discovered during this activity.

2. **Requirement classification and organization**:

- This activity takes the unstructured collection of requirements, group related requirements and organize them into coherent clusters.
- The most common way of grouping requirements is to use a model of the system architecture to identify sub-systems and to associated requirements with ease sub-system.

1.14 Requirement Elicitation and Analysis

3. **Requirements prioritization and negotiation**:

- When multiple stakeholder are envolved, requirements will conflict.
- This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiation.

4. **Requirement specification:**

- The requirement are documented and input into next round of the spiral.
- Formal or informal requirement documents may be produced.

1.15 Requirement Validation and Management

- **Requirement validation and requirement management** are two distinct but interconnected processes in the field of requirements engineering.
- While requirement validation **focuses on ensuring the accuracy, completeness, consistency, and feasibility of requirements**.
- Requirement management encompasses the activities **involved in organizing, documenting, prioritizing, and controlling requirements throughout the entire software development lifecycle.**

Requirement Validation:

It involves checking the requirements for correctness, completeness, consistency, feasibility, traceability, and testability.

1.15 Requirement Validation and Management

The goal is to **identify and resolve any issues, errors, ambiguities, or contradictions in the requirements early in the development process**, minimizing the risks associated with misunderstood or poorly defined requirements.

Requirement Management:

Requirement management is a broader process that involves activities related to capturing, documenting, organizing, prioritizing, tracking, and controlling requirements throughout the project lifecycle.

1.15 Requirement Validation and Management

It includes the following key aspects:

Elicitation: Gathering requirements from stakeholders through techniques such as interviews, workshops, and observations.

Documentation: Capturing requirements in a structured format, such as a requirements document or a requirements management tool. This includes documenting functional and non-functional requirements, constraints, assumptions, and dependencies.

Prioritization: Assessing and assigning priorities to requirements based on their importance, business value, and urgency. This helps in making informed decisions when resource constraints or changes occur.

1.15 Requirement Validation and Management

Traceability: Establishing and maintaining traceability links between requirements, business goals, and other project artifacts. This enables stakeholders to understand the rationale behind each requirement and facilitates impact analysis when changes occur.

Change Management: Handling and controlling changes to requirements throughout the project lifecycle. This involves evaluating change requests, analyzing their impact, and managing the process of incorporating approved changes into the requirements baseline.

Communication and Collaboration: Facilitating effective communication and collaboration among stakeholders, including business analysts, developers, testers, and project managers. This ensures a shared understanding of the requirements and promotes collaboration throughout the project.

Versioning and Baselines: Managing different versions or baselines of requirements to track changes over time and support configuration management.