FAQs about software engineering

1a **What is software?**

A software system usually consists of a number of separate programs, configuration files, system documentation, and user documentation. There are two fundamental types of software product:

Generic products: These are stand-alone systems that are produced by a development organization and sold on the open market to any customer who is able to buy them. Eg: MS word, Windows XP etc.

Customized (or bespoke): These are systems which are commissioned by a particular customer. A software contractor develops the software especially for that customer. Eg: Bank management, Office automation etc.

3a **What is software engineering?**

Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use.

Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.

Software Engineering provides a set of:

Procedures: defines **what** are the activities and **when** they will be executed bind the methods and tools into a framework.

Methods: defines **How** each activity is executed using the technology provide the rules and steps for carrying out software engineering tasks, such as project planning, requirements analysis, design, coding, testing and maintenance.

Tools: provide automated or semi-automated support for methods. Tools that automates a range of software engineering methods, called Computer-Aided Software Engineering (CASE).

What's the difference between software engineering and computer science? Computer science is concerned with the theories and methods that underlie computers and software systems

While
Software engineering is concerned with the practical problems of producing software

What is the difference between software engineering and system engineering? System engineering is concerned with all aspects of the development and evolution of complex systems i.e., **manufacturing a product** While Software engineering is concerned with the practical problems of producing software i.e., developing software

What is a software process?

A software process is the set of activities and associated results that produce a software product. There are four fundamental process activities:

1. **Software specification** where customers and engineers define the software to be produced and the constraints on its operation
2. **Software development** where the software is designed and programmed
3. **Software validation** where the software is checked to ensure that it is what the customer requires
4. **Software evolution** where the software is modified to adapt it to changing customer and market requirements

What is a software process model?

A software process model is a simplified description of a software process; different process models depict different arrangement of the process activities.

What are the costs of software engineering?

The distribution of costs across the different activities in the software process depends on the process used and the type of software that is being developed. Roughly 60% of costs are development costs, 40% are testing costs

What are software engineering methods?

Defines **How** each activity is executed using the technology provide the rules and steps for carrying out software engineering tasks, such as project planning, requirements analysis, design, coding, testing and maintenance

1a **What are the attributes of good software?**

- **Maintainability:** Software must evolve to meet changing needs
- **Dependability:** Software must be trustworthy, reliable
- **Efficiency:** Software should not make wasteful use of system resources.
- **Usability:** Software must be usable by the users for which it was designed.

Professional and ethical responsibility

Software engineers must behave in an ethical and morally responsible way if they are to be respected as professionals some of these are:

- **Confidentiality:** you should normally respect the confidentiality of your employers or clients irrespective of whether a formal confidentiality agreement has been signed.
- **Competence:** you should not misrepresent your level of competence. You should not knowingly accept work that is outside your competence.
- **Intellectual property rights**: you should be aware of local laws governing the use of intellectual property such as patents and copyright. You should be careful to ensure that the intellectual property of employers and clients is protected.
- **Computer misuse:** you should not use your technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses).

6a Software engineers shall adhere to the following Eight Principles:

1. **PUBLIC** - Software engineers shall act consistently with the public interest.
2. **CLIENT AND EMPLOYER** - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. **PRODUCT** - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. **JUDGMENT** - Software engineers shall maintain integrity and independence in their professional judgment.
5. **MANAGEMENT** - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. **PROFESSION** - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. **COLLEAGUES** - Software engineers shall be fair to and supportive of their colleagues.
8. **SELF** - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Systems engineering

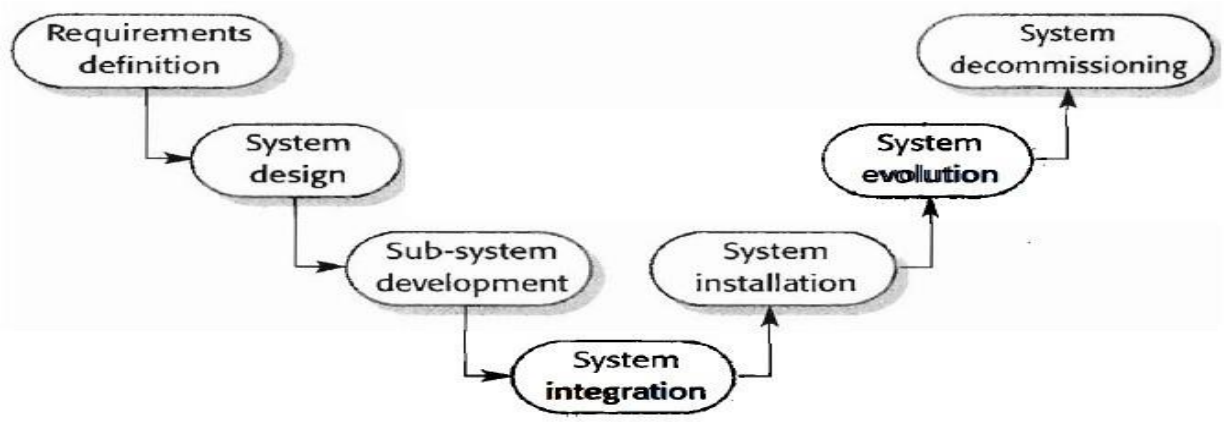
Systems engineering is the activity of specifying, designing, implementing, validating, deploying and maintaining socio-technical systems.

It is concerned with software, hardware and the system's interactions with users and its environment. They must consider the services, the constraints, the ways in which the system is used to fulfill its purpose.

Difference between the system engineering and the software development:

1. **Limited scope for rework** during system development once some system engineering decisions, have been made, they are very expensive to change or rework but software engineering allows changes to be made during system development, in response to new requirements.
2. **Interdisciplinary involvement** Many engineering disciplines may be involved in system engineering.

The Systems engineering Process



2a **Software Processes**

A software process is a **set of activities** that leads to the production of a software product. These activities may involve the **development of software from scratch** or extending and modifying existing systems and by **configuring and integrating off-the-shelf software** or system components.

Software processes are complex soComputer-aided software engineering (CASE) tools can support some process activities.

Fundamental activities of software processes:

1. **Software specification** The functionality of the software and constraints on its operation must be defined.
2. **Software design and implementation** The software to meet the specification must be produced.
3. **Software validation** The software must be validated to ensure that it does what the customer wants.
4. **Software evolution** The software must evolve to meet changing customer needs.

Software process models

A software process model is an abstract representation of a software process activites. Each process model represents a different arrangement of process activites from a particular perspective.

UMBRELLA Activities

1. **Software project tracking and control**
   - Allows the software team to assess progress against the project plan and take any necessary action to maintain the schedule.
2. **Risk management**
   - Assesses risks that may affect the outcome of the project or the quality of the product.
3. **Software quality assurance**
   - Defines and conducts the activities required to ensure software quality.
4. **Technical reviews**
   - Assesses software engineering work products in an effort to uncover & remove errors before they are propagated to the next activity.
5. **Measurement**
   - defines and collects process, project, and product measures that assist the team in delivering software that meets stakeholders' needs; can be used in conjunction with all other framework and umbrella activities.
6. **Software configuration management**
   - Manages the effects of change throughout the software process.
7. **Reusability management**
   - Defines criteria for work product reuse (including software components) and establishes mechanisms to achieve reusable components.
8. **Work product preparation and production**
   - Encompasses the activities required to create work products such as models, documents, logs, forms, and lists.

A process adopted for one project might be significantly different than a process adopted for another project. Among the differences are:

1. Overall flow of activities, actions, and tasks and the interdependencies among them
2. Degree to which actions and tasks are defined within each framework activity
3. Degree to which work products are identified and required
4. Manner in which quality assurance activities are applied
5. Manner in which project tracking and control activities are applied
6. Overall degree of detail and rigor with which the process is described
7. Degree to which the customer and other stakeholders are involved with the project
8. Level of autonomy given to the software team
9. Degree to which team organization and roles are prescribed

3a **General Principles**

- **The First Principle: *The Reason It All Exists.***
- **The Second Principle: *KISS (Keep It Simple, Stupid!)***
- **The Third Principle: *Maintain the Vision***
- **The Fourth Principle: *What You Produce, Others Will Consume***
- **The Fifth Principle: *Be Open to the Future***
- **The Sixth Principle: *Plan Ahead for Reuse***
- **The Seventh principle: *Think!***

1. The Reason It All Exists:

- This principle emphasizes understanding the **core purpose** behind the product or system being developed. Developers must keep in mind the **reason** for the system's existence, which drives all design and implementation decisions.

2. *KISS (Keep It Simple, Stupid!):*

- This principle focuses on **SIMPLICITY**. The goal is to avoid unnecessary complexity in design and development, making the system easier to understand, maintain, and scale.

3. *Maintain the Vision:*

- This highlights the importance of **staying aligned with the original vision** of the project. It serves as a reminder that, even as the system evolves, the development team should keep the **long-term goals and vision intact.**

4. *What You Produce, Others Will Consume:*

- This principle reminds developers that the system or **product they build will be used by others**.
  - Whether it's a team member or an end user, it's crucial to create something that is **user-friendly, well-documented, and reliable**.

5. *Be Open to the Future:*

- This principle emphasizes being **flexible and adaptive** to future changes. The system should be designed in a way that it can accommodate **growth** and **technological advancements** without needing a complete overhaul.

6. *Plan Ahead for Reuse:*

- This principle encourages developers to think about how **different parts of the system** can be **reused in future projects or components**. Planning for reuse makes future development faster and more efficient.

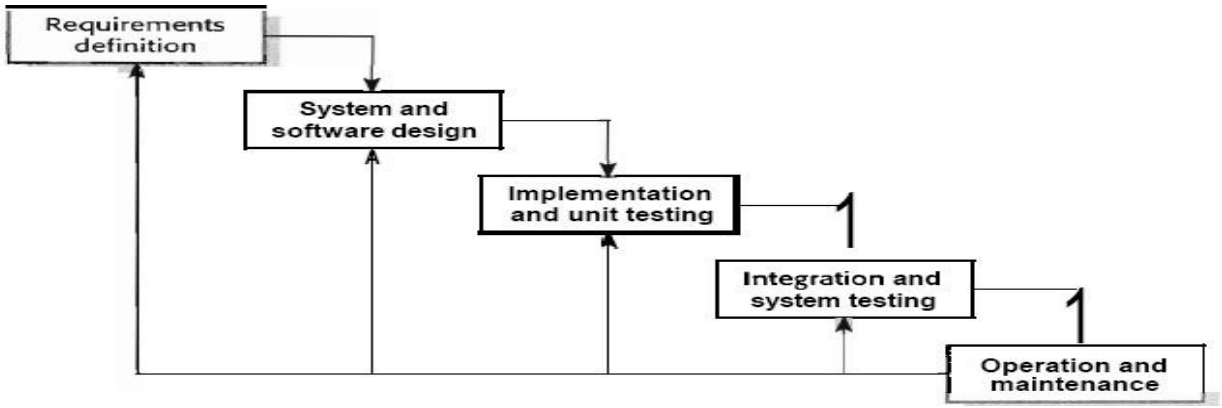
7. *Think!:*

- This principle calls for developers to **engage in thoughtful problem-solving**. It encourages **critical thinking** and careful consideration of design decisions, ensuring that the system is optimized, efficient, and well-architected.

7a **SOFTWARE MYTHS** any 3 with eg

- **Management myths.**
  - Beliefs about management practices that are widely accepted but often incorrect, leading to misunderstandings about how to effectively lead and organize teams.
- **Customer myths.**
  - Common misconceptions about customer behavior and preferences that can misguide businesses in their strategies and decision-making.
- **Practitioner's myths.**
  - Misunderstandings held by professionals in a specific field regarding best practices or effective methods that may be based on outdated information or assumptions.
- **Management Myths**
  - Adding more people to a project will speed it up.
  - A detailed plan guarantees project success.
  - The best managers don't need to understand the technical details.
  - Once a product is delivered, the job is done.
- **Customer Myths**
  - Customers always know what they want.
  - The customer is always right.
  - A product with more features is always better.
  - Once we see the prototype, we can make changes for free.
- **Practitioner's Myths**
  - If it works, it's good enough.
  - More code means more productivity.
  - We can fix it later.
  - Automation will solve all our problems.

9a **The waterfall model** with diagram



- The first published model called as the Classic Lifecycle Model developed by Royce in 1970
- Follows a documentation driven paradigm.
- Is a **time-ordered sequence of activities** called as lifecycle stages.
- Requirements of customer are well understood.
- Work flows from Communication activity to deployment in Linear Fashion/ a systematic and sequential approach to software
- Each framework activity starts only after the previous activity has finished.
- Iteration is possible only in a rigid manner.
- Neither emphasis nor encourages on reusability

The principal stages of the model map onto fundamental development activities:

1. **Requirements analysis and definition** The system's services, constraints and goals are established by consultation with system users. They are then defined in detail and serve as a system specification.
2. **System and software design** The **systems design process** partitions the requirements to either hardware or software systems. It establishes an overall system architecture. **Software design** involves identifying and describing the fundamental software system abstractions and their relationships.
3. **Implementation and unit testing** During this stage, the software design is implemented as a set of program units. Unit testing involves verifying that each unit meets its specification.
4. **Integration and system testing** The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.
5. **Operation and maintenance** The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages and improving and enhancing the system's services as new requirements are discovered

Advantages:

- Simplicity and the logical way of structuring the different activities in a software project.
- Documentation is produced at each phase and that it fits with other engineering process models
- Well suited only if the requirements are clearly understood

Disadvantages:

- It assumes that the requirements are completely ready before the design, which is not the case in most of the development scenarios
- Does not admit iteration between stages.
- A working version of the software is not available until late in the development process and if major errors are discovered at this stage, the results can be disastrous
- The customer is completely cut-off from the development until the acceptance testing stage.
- Blocking states for the project team members where they have to wait till the previous stage is completed.

When to use waterfall model

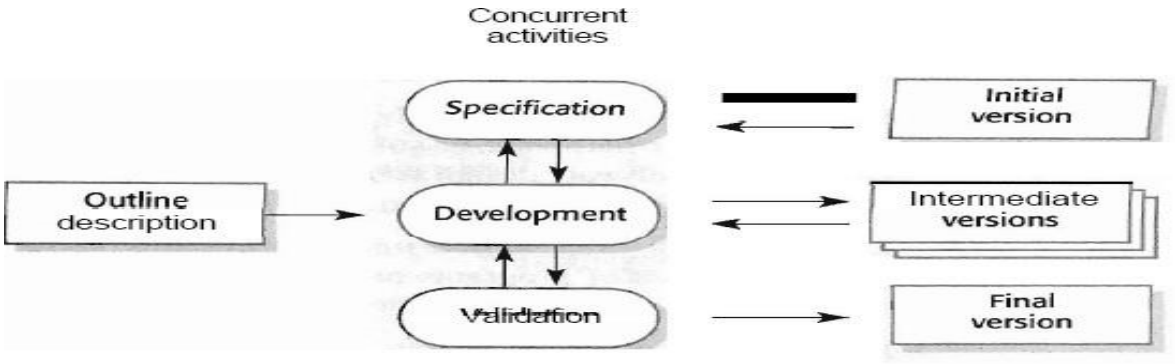
- The Waterfall model should only be used when the requirements are well understood and unlikely to change radically during system development
- It will be used for the small projects,with low risk,less cost, small team

Examples for Waterfall model

- Building a Hospital Management System.
- Designing a New Car Model.
- Developing a Library Management System.
- Building a Mobile Banking Application.
- Developing a Simple E-Commerce Website.

Evolutionary development

Evolutionary development is based on the idea of **developing an initial implementation**, exposing this to **user comment and refining** it through many versions until an adequate system has been developed. Specification, development and validation **activities are interleaved** rather than separate, with rapid feedback across activities.



There are two fundamental types of evolutionary development:

1. **Exploratory development** where the objective of the process is to work with the customer to explore their requirements and deliver a final system. The development **starts with the parts of the system that are understood**. The system evolves by adding new features proposed by the customer.
2. **Throwaway prototyping** where the objective of the process is to **understand the customer's requirements** and hence develop a **better requirements definition** for the system. The **prototype concentrates** on experimenting with the customer requirements that are **poorly understood**.

Merits

- The specification can be developed incrementally. As users develop a better understanding of their problem, this can be reflected in the software system.
- The customer is continiously involved in the development
- It admits iteration between stages
- A working version of the software is available very early and will be modified based on user feed back.
- Blocking states for the project team will not be there

De-Merits:

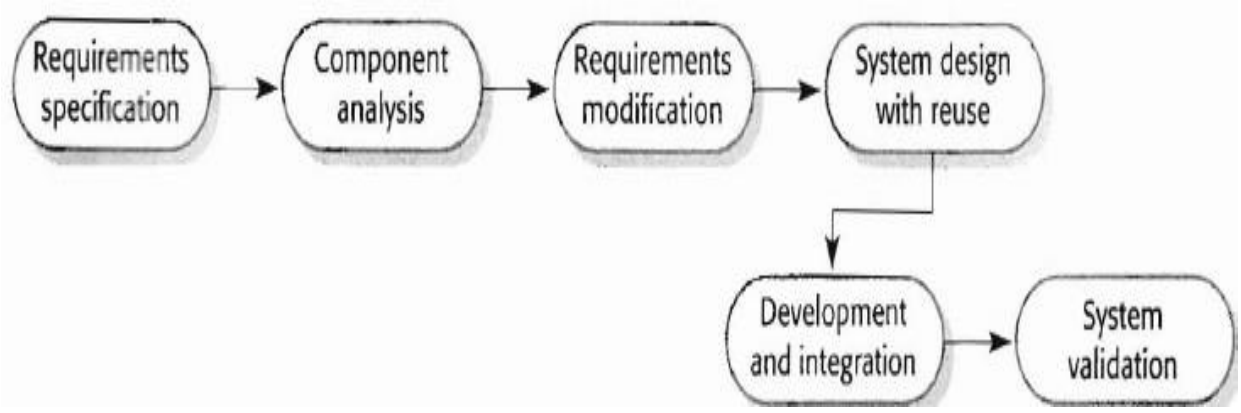
- The process is not visible Managers need regular deliverables to measure progress.
- High cost to produce documents that reflect every version of the system.
- Systems are ofen poorly structured Continual change tends to corrupt the software structure.
- Incorporating software changes becomes increasingly difficult and costly.
- The model is incapable for use in large, complex, long-lifetime systems

When to use Evolutionary development

- For small and medium-sized systems the evolutionary approach is the best approach to development.
- **Exploratory development** should be used if the **majority of the requirements are complex** and not properly understood, then **start with the simple and well understood requirements** and evolve the system as specified by the customer
- **Throwaway prototyping** should be used if the **majority of the requirements are simple and well understood**, then **construct a quick & dirty prototype to understand the critical requirements**, document the requirements and use any simple process model to construct the product.

Component-based software engineering (CBSE)

CBSE model is based on **software reuse**. This model relies on a large base of reusable software components and some integrating framework for these components. These components are systems in their own right (**COTS** or commercial off-the-shelf systems) that may provide specific functionality.



The stages in generic process model for CBSE.

1. **Component analysis** Given the requirements specification, a search is made for components to implement that specification. Usually, there is no exact match, and the components that may be used only provide some of the functionality required.
2. **Requirements modification** During this stage, the requirements are analysed using information about the components that have been discovered. They are then modified to reflect the available components.
3. **System design with reuse** During this phase, the framework of the system is designed or an existing framework is reused. The designers take into account the components that are reused and organise the framework to cater to this.
4. **Development and integration** Software that cannot be externally procured is developed, and the components and COTS systems are integrated to create the new system.
5. **System validation** the intergrated system is validated to ensure it meets the specified requirements

Advantages:

- Reduce the amount of software to be developed and so reducing cost and risks.
- Leads to faster delivery of the software
- Less resourcs is needed

Disadvantages:

- Requirements compromises are inevitable and this may lead to a system that does not meet the real needs of users
- Developing a framework to integrate the COTS may be difficult.
- Control over the system evolution is lost
- Extensive testing is needed during integration

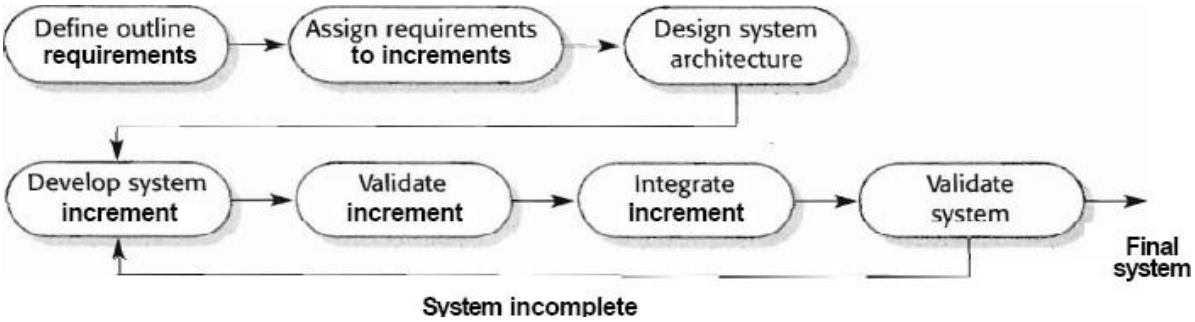
When to use CBSE

- The requirements are well defined, there are COTS available that can be reused.
- Used for the products that has to be developed rapidly at the less cost.
- The team has expertise of developing similar sort of product .

9b **Incremental delivery** with diagram

- Incremental delivery adapts incremental development process, customers identify, outline, the services to be provided by the system.
- Customers identify which of the services are most important and which are least important to them.
- A number of delivery increments are then defined, with each increment providing a sub-set of the system functionality.
- The allocation of services to increments depends on the service priority with the highest priority services delivered first.
- Once the system increments have been identified, the requirements for the services to be delivered in the first increment are defined in detail, and that increment is developed.
- During development, further requirements analysis for later increments can take place, but requirements changes for the current increment are not accepted.
- Once an increment is completed and delivered, customers can put it into service.
- Customers can experiment with the system that helps them clarify their requirements for later increments and for later versions of the current increment.

- As new increments are completed, they are integrated with existing increments so that the system functionality improves with each delivered increment.



Advantages:

- Customers do not have to wait until the entire system is delivered
- Customers can use the early increments to gain experience that informs their requirements for later system increments
- There is a lower risk of overall project failure
- As the highest priority services are delivered first, and later increments are integrated with them, so most important system services receive the most testing.

Disadvantages:

- Increments should be relatively small, so It is difficult to map the customer's requirements onto increments of the right size
- it can be hard to identify
- As all requirements are not defined it is hard to identify common facilities that are needed by all increments

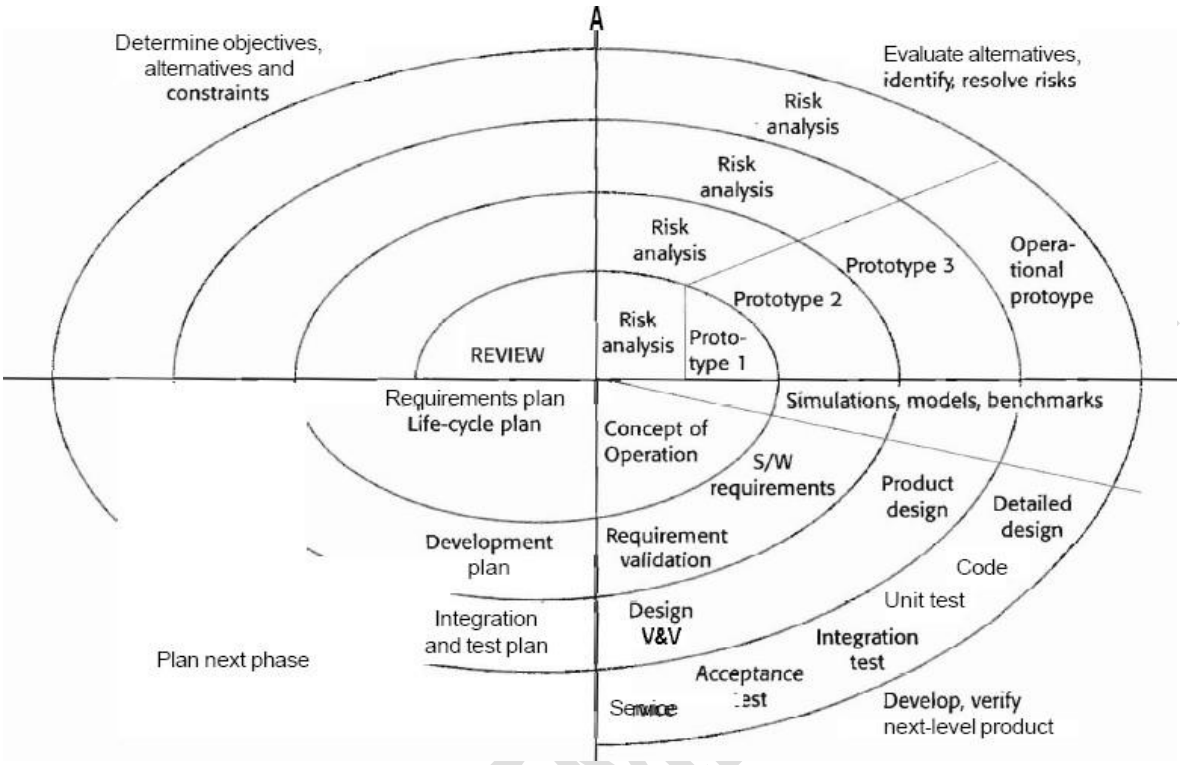
When to use Incremental Delivery

- Initial software requirements are reasonably well-defined.
- When there is a compelling need to provide a limited set of software functionality to users quickly and then refine and expand on that functionality in later releases.
- When staffing is unavailable for a complete implementation by business deadline
- Usually used for developing the products to reach the market early

10a **Boehm's Spiral Model** with diagram

The process is represented as a spiral. Each loop in the spiral represents a phase of the software process. Thus, the innermost loop might be concerned with system feasibility, the next loop with requirements definition, the next loop with system design and so on. The main difference between the spiral model and other software process models is the explicit recognition of risk

Spiral development



Each loop in the spiral is split into four sectors:

1. **Objective** setting Specific objectives for that phase of the project are defined. Constraints on the process and the product are identified and a detailed management plan is drawn up. Project risks are identified. Alternative strategies, depending on these risks, may be planned.
2. **Risk assessment and reduction** For each of the identified project risks, a detailed analysis is carried out. Steps are taken to reduce the risk.
3. **Development and validation** A development model for the system is chosen.
4. **Planning** The project is reviewed and a decision made whether to continue with a further loop of the spiral. If it is decided to continue, plans are drawn up for the next phase of the project.

Advantages:

- Defers elaboration of low risk software elements
- Incorporates prototyping as a risk reduction strategy
- Gives an early focus to reusable software
- Accommodates life-cycle evolution, growth, and requirement changes
- Incorporates software quality objectives into the product
- Focus on early error detection and design flaws
- Sets completion criteria for each project activity to answer the question: "How much is enough?"
- Uses identical approaches for development and maintenance
- The model is open ended so can be extended for project evolution
- can be used for hardware-software system development

Disadvantages:

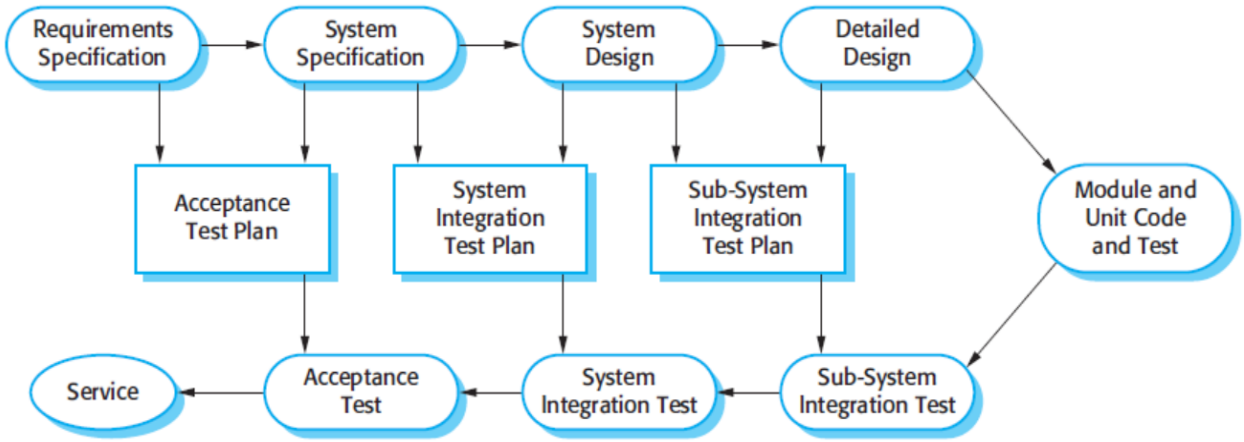
- Rigid process activities leads to no room for creativity
- Needs more managers, skilled professionals to adapt this model
- High cost for documentation and the process

When to use Spiral development:

- Massively big projects, whose scope is not defined
- Projects with high risk, high cost, complex.
- The projects for evolving existing legacy business system

Testing phases in a plan-driven software process

In a **plan-driven software process**, such as the Waterfall model, the testing phases are systematically structured and planned in advance, with a strong emphasis on documentation & predefined stages.



Plan-Driven Approach Characteristics:

- **Documentation**:
  - Every phase in a plan-driven process is documented extensively. Test plans, test cases, and expected outcomes are all documented ahead of time.
- **Sequential Progression**:
  - Each phase must be completed before the next begins. For example, system testing can't start until integration testing is finished.
- **Rigorous Planning**:
  - Test activities are planned early in the development lifecycle, often during the requirements phase.
    - This structured approach ensures thorough testing at every level of the software but may lack flexibility. If requirements change late in the development process, it can lead to rework and delays due to the rigid nature of the plan-driven process.

Outcomes of Plan Driven Testing

- The goal is to catch bugs early by ensuring that each part of the system works independently.
- Identifies issues with how different parts of the system work together.
- Ensures that the software system as a whole works correctly and meets the defined requirements.
- Confirms that the system is ready for release and meets user expectations.
- Maintains the stability of the software as new features or fixes are added.

Coping with Change

- **Change is inevitable in all large software projects.**
  - The system requirements change as the business procuring the system responds to external pressures and management priorities change.
- As **new technologies become available, new design and implementation possibilities emerge**.
  - Therefore whatever software process model is used, it is essential that it can accommodate changes to the software being developed.

Two approaches that may be used to reduce the costs of rework:

1. **Change avoidance**,
   - where the software process includes activities that can anticipate possible changes before significant rework is required.
2. **Change tolerance**,
   - where the process is designed so that changes can be accommodated at relatively low cost.

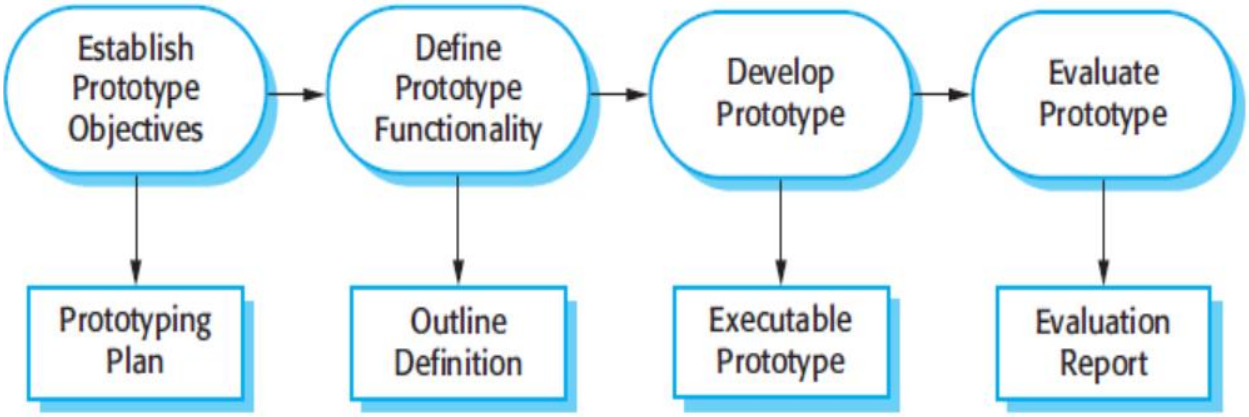
12a **2** ways of coping with change & changing system requirements.

1. **System prototyping**, where a version of the system or part of the system is developed quickly to check the customer's requirements and the feasibility of some design decisions.
   - This supports change avoidance as it allows users to experiment with the system before delivery and so refine their requirements.
2. **Incremental delivery**, where system increments are delivered to the customer for comment and experimentation.
   - Supports both change avoidance and change tolerance.

Prototyping

- A prototype is an initial version of a software system that is used to demonstrate concepts, try out design options, and find out more about the problem and its possible solutions.
- A software prototype can be used in a software development process to help anticipate changes that may be required:
  - Requirements engineering:, a prototype can help with the elicitation & validation of system requirements.
  - System design: a prototype can be used to explore particular software solutions & to support user interface design.

- Testing process to run back-to-back tests.



Benefits of prototyping

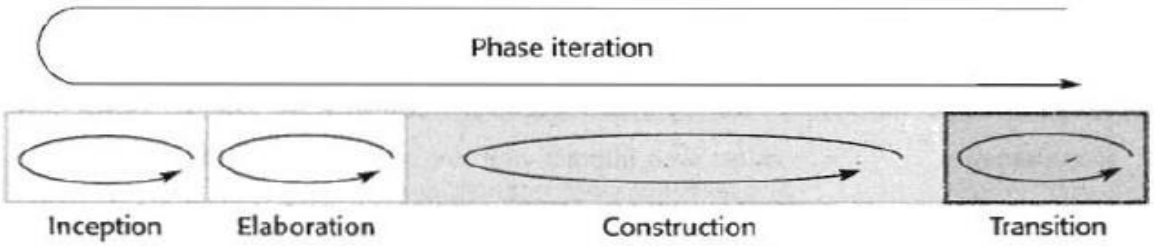
1. Improved system usability.
2. A closer match to users' real needs.
3. Improved design quality.
4. Improved maintainability.
5. Reduced development effort.

Bottleneck

- It may be impossible to tune the prototype to meet *non-functional requirements*, which were ignored during prototype development.
- Rapid change during development inevitably means that the prototype is undocumented.
  - The only design specification is the prototype code. This is not goodenough for long-term maintenance.
- The changes made during prototype development will probably have degraded the system structure. The system will be difficult and expensive to maintain.
- Organizational quality standards are normally relaxed for prototype development.

11a **The Rational Unified Process** with diagram

- The Rational Unified Process (RUP) is derived from UML
- Draws the best features and characteristics of conventional Software process models.
- Focuses on the importance of Customer communication & streamlined methods for describing the customer's view of the system.
- Emphasizes the important role of software architecture and helps architect focus on the right goals, like understandability, reliance to future changes, and reuse.
- The unified process is incremental, and iterative, providing the evolutionary feel.
- Provides necessary technology to support Object-Oriented-SE practice.



The phases of Rational Unified Process

The INCEPTION Phase:

- Encompasses both Customer Communication & Planning Activities.
- Collaborates with customers, end-users, and all stake holders for identifying the fundamental Business requirements in the form of USE-CASES.
- A rough Architecture for the system is proposed, a tentative outline of major subsystems and the function and features is considered.
- A Iterative, incremental project plan is developed.
- Planning identifies Resources, Assesses major risks, defines a Schedule, decomposes the project, and establishes basis for Incremental deliverables.

The ELABORATION Phase:

- Encompasses Customer communication and Modeling activities
- Refines and expands preliminary Use-Cases that were developed as part of Inception phase
- A close review is done on the Plan and any modification to the plan is made.

The CONSTRUCTION Phase:

- Identical to the construction activity of generic process.
- Using Architectural model as input, the construction phase develops / acquires software components that will make each use-case operational for end-users.
- All reqd. functionalities and features are implemented.
- Unit tests are designed & executed for each use-case.
- Integration activities are conducted.

The TRANSITION Phase:

- Encompasses Deployment activity
- Developed software is given to end-users for beta-testing.
- User feedback reports DEFECTS & NECESSARY changes.
- Necessary work products/ support information are created.
- The software is monitored.
- Support for operating environment is provided
- Defect reports & requests for changes are submitted and evaluated.

RUP ITERATION

- In-phase iteration
  - Each phase is iterative with results developed incrementally.
- Cross-phase iteration
  - As shown by the loop in the RUP model, the whole set of phases may be enacted incrementally.

When to use RUP

- For the large projects been developed based on object orientation
- If the software has to be developed based on user perspective

Six best software engineering practices used by RUP:

1. **Develop software iteratively** Plan and develop increments of the system based on customer priorities.
2. **Manage requirements** Explicitly document the customer's requirements and keep track of changes to these requirements.
3. **Use component-based architecture** Structure the system architecture into components
4. **Visually model software** Use graphical UML models to present static and dynamic views of the software
5. **Verify software quality** Ensure that the software meets the organizational quality standards
6. **Control changes** Manage changes to the software using a change management system and configuration management procedures and tools

Disadvantages:

- The RUP is not a suitable process for all types of development
- It needs object oriented specification, design and implementation
- Needs the use of UML to represent the intermediate work product

Static workflows in the RUP

Workflow	Description
Business modelling	Business processes modelled using business use cases.
Requirements	Actors who interact with the system are identified and use cases are developed to model the system requirements.
Analysis & Design	A design model is created and documented using architectural models, component models, object models and sequence models.
Implementation	Components in the system are implemented & structured into implementation sub-systems. Automatic code generation from design models helps accelerate this process.
Testing	Testing is an iterative process that is carried out in conjunction with implementation. System testing follows the completion of the implementation.
Deployment	A product release is created, distributed to users and installed in their workplace.
Configuration & change management	This supporting workflow manages changes to the system.
Project management	This supporting workflow manages the system development.
Environment	This workflow is concerned with making appropriate software tools available to the software development team.

8a **Process Activities**

The four basic process activities of specification, development, validation and evolution are organized differently in different development processes.

-> **Software specification**

Software specification or requirements engineering is the process of understanding and defining what services and constraints are required from the system.
This process produce a requirements document that is the specification for the system.

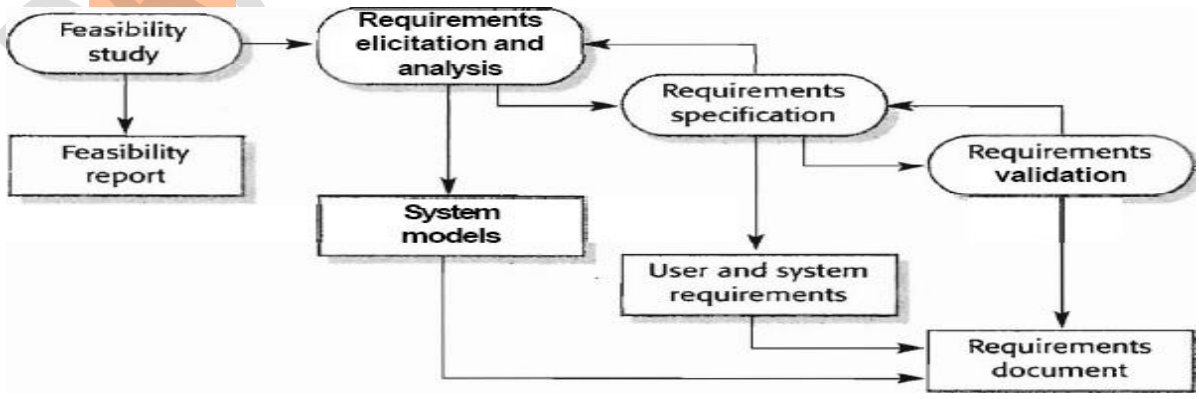
There are four main phases in the requirements engineering process**:**

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 cost-effective, budgetary constraints, legality etc,
- Then a decision is made whether to go ahead with a more detailed analysis.

2. Requirements elicitation and analysis:
- This is the process of deriving the system requirements through observation of existing systems, discussions with users and procurers, task analysis and so on.
- This may involve the development of one or more system models and prototypes to understand the system to be specified.

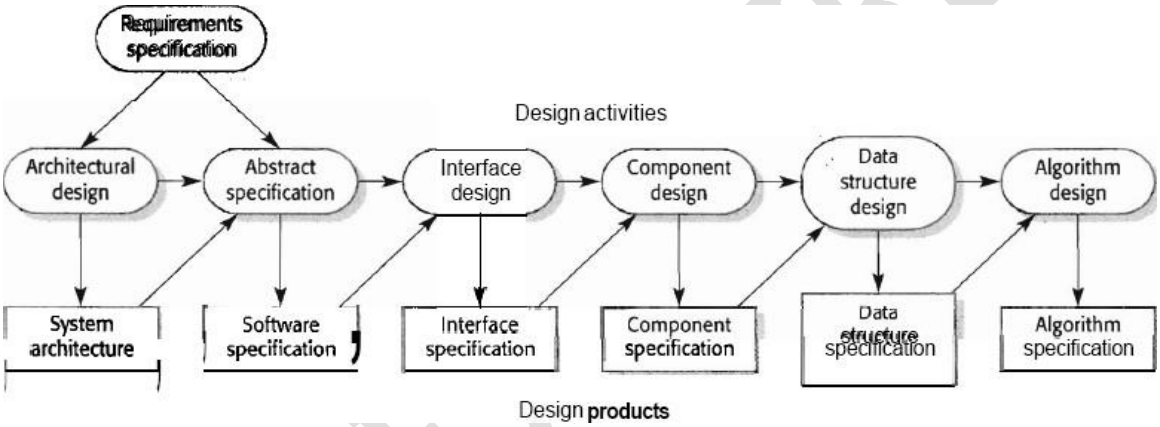
3. Requirements specification**:**
- The activity of translating the information gathered during the analysis activity into a document that defines a set of requirements.
- Two types of requirements included in this document.
- User requirements are abstract statements of the system requirements for 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 correctness, consistency and completeness.
- Errors in the requirements document are discovered and modified to correct them
- The activities of analysis, definition and specification of requirements are interleaved.



-> **Software Design & Implementation**

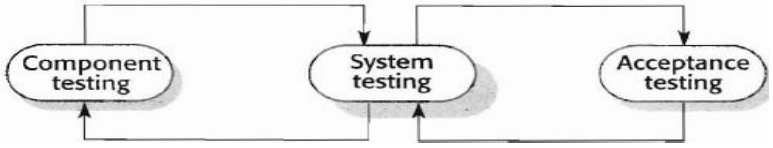
- The process of converting a system specification into an executable system. It involves processes of software design and programming.
- A software design is a description of the structure of the software to be implemented, the data, the interfaces and the algorithms used.
- Designing is done iteratively through a number of versions
- The design process involves adding detail as the design is developed with constant backtracking to correct errors and omissions in earlier designs.
- The design process may involve developing several models of the system at different levels of abstraction.
- The final results of the process are precise specifications of the algorithms and data structures to be implemented



The specific design process activities are:

1. **Architectural design** The sub-systems making up the system and their relationships are identified and documented.
2. **Abstract specification** For each sub-system, an abstract specification of its services and the constraints under which it must operate is produced.
3. **Interface design** For each sub-system, its interface with other sub-systems is designed and documented.
4. **Component design** Services are allocated to components and the interfaces of these components are designed.
5. **Data structure design** The data structures used in the system implementation are designed in detail and specified.
6. **Algorithm design** The algorithms used to provide services are designed in detail and specified.

-> **Software Validation**



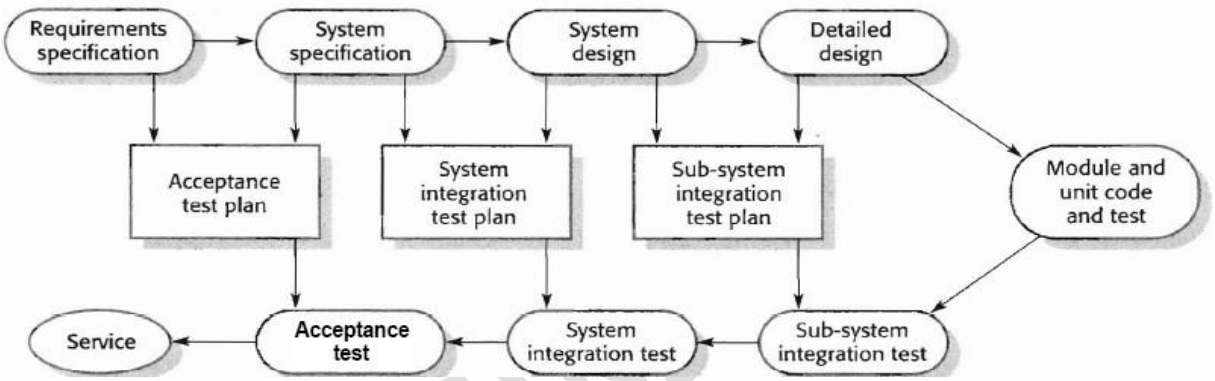
The TestingProcess

Verification and Validation (V & V) is intended to show that a system conforms to its specification and that the system meets the expectations of the customer.

The stages in the testing process are: (Figure 4.10)

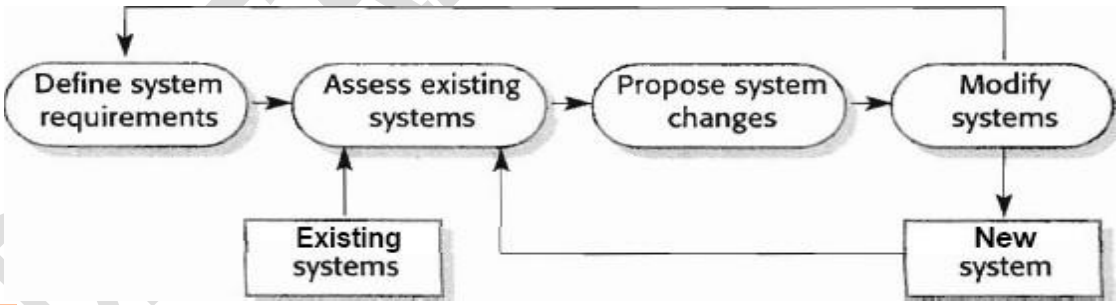
1. **Component (or unit) testing** Individual components are tested to ensure that they operate correctly.
2. **System testing** The components are integrated to make up the systemand tested to finding errors that result from interactions between components.
3. **Acceptance testing** The system is tested with data supplied by the customer which reveal errors and omissions in the system requirements

An independent team of testers will conduct testing using test plans, test cases, and test data that are developed from the system specification and design.



-> **Software evolution**

The software evolution process is to change the system according to the emerging requirements of the customers.



- It starts by defining the new requirements, constraints and services needed
- The existing system and its documents are assesed to find out the possiability , consequences and cost of changinging the system
- Then a proposal of the design of changing the system is prepared and contracted
- The existing system will be modified according to the design and an extensive testing is conducted

AGILE METHODOLOGY

- Agile methodology is a set of principles and practices for software development that emphasizes iterative progress, collaboration, and flexibility. It is designed to enhance the ability to respond to change and deliver high-quality software more efficiently.
- Agile methodologies typically involve breaking projects into smaller, manageable units of work (called iterations or *sprints*), and regularly reviewing and adjusting progress through feedback loops.
  - Popular frameworks that apply Agile principles include Scrum, Kanban, and Extreme Programming (XP).
- Rapid software development processes are designed to produce useful software quickly.
  - The software is not developed as a single unit but as a series of increments, with each increment including new system functionality.
- Although there are many approaches to rapid software development, they share some fundamental characteristics:
  - The processes of specification, design, and implementation are interleaved. The user requirements document only defines the most important characteristics of the system.
  - The system is developed in a series of versions.
  - System user interfaces are often developed using an interactive development system that allows the interface design to be quickly created by drawing and placing icons on the interface.

Agile Methods

- Dissatisfaction with the overheads involved in design methods (complexity, time consuming, rigid workflow, resource intensive) led to the creation of agile methods. These methods:
  - Focus on the code rather than the design;
  - Are based on an iterative approach to software development;
  - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.
- Agile methods are probably best suited to small/medium-sized business systems or PC products.

13a **Principles of agile methods**

Principle	Description
Customer involvement	**Their role is provide and prioritize new system requirements & to evaluate the iterations in s/m.**
Incremental delivery	**The software is developed in increments with the customer specifying the requirements to be included in each increment.**
People not process	**Skills of the development team should be recognized & exploited. Team members should be left to develop their own ways of working without prescriptive processes.**

Embrace change	**Expect the system requirements to change and so design the system to accommodate these changes.**
Maintain simplicity	**Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.**

Agility Principles

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

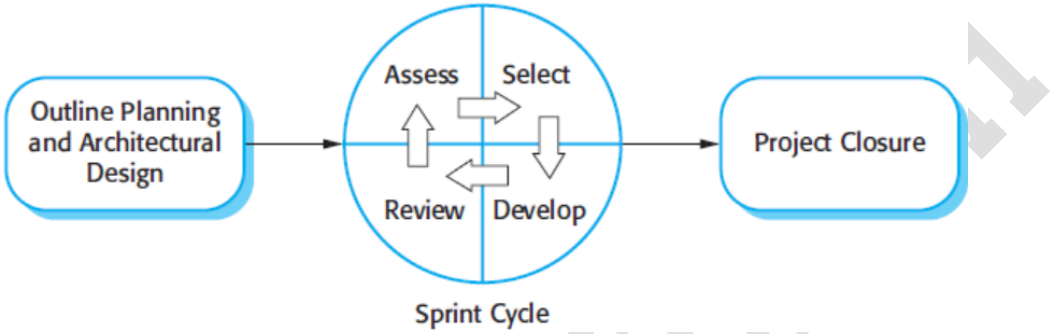
Principles underlying agile methods are sometimes difficult to realize:

1. It can be difficult to keep the interest of customers who are involved in the process.
2. Team members may be unsuited to the intense involvement that characterizes agile methods.
3. Prioritizing changes can be difficult where there are multiple stakeholders.
4. Maintaining simplicity requires extra work.
5. Contracts may be a problem as with other approaches to iterative development.

15a **S C R U M**

- Scrum principles are consistent with the agile manifesto and are used to guide development activities within a process that incorporates the following framework activities: requirements, analysis, design, evolution, and delivery.
- Scrum incorporates a set of process patterns that emphasize project priorities, compartmentalized work units, communication, and frequent customer feedback.

  - Scrum emphasizes the use of a set of software process patterns that have proven effective for projects with tight timelines, changing requirements, and business criticality. Each of these process patterns defines a set of development actions:
    - Backlog, Sprints, Scrum meetings, team leader / scrum master, Demos
- Innovative feature of Scrum is its central phase, namely the sprint cycles.
- A Scrum sprint is a planning unit in which the work to be done is assessed, features are selected for development, and the software is implemented.
- At the end of asprint, the completed functionality is delivered to stakeholders.



Three Phases in Scrum:

- First is an O**utline Planning Phase** where you establish the general objectives for the project and design the software architecture.
- **Series of Sprint Cycles**, where each cycle develops an increment of the system.
- **Project Closure Phase** wraps up the project, completes required documentation such as system help frames and user manuals, and assesses the lessons learned from the project.

✧ **Key characteristics of this process are as follows:**

- Sprints are fixed length, normally 2-4 weeks. They correspond to the development of a release of the system in XP.
- The starting point for planning is the product backlog, which is the list of work to be done on the project.
- The selection phase involves all of the project team who work with the customer to select the features and functionality to be developed during the sprint.
- Once these are agreed, the team organizes themselves to develop the software. Short daily meetings involving all team members are held to review progress and if necessary, reprioritize work ('Scrum master').
- At the end of the sprint, the work done is reviewed & presented to stakeholders.
  - The next sprint cycle then begins.

- ***Backlog***
- a prioritized list of project requirements or features that provide business value for the customer.
- Items can be added to the backlog at any time (this is how changes are introduced).
- The product manager assesses the backlog and updates priorities as required.

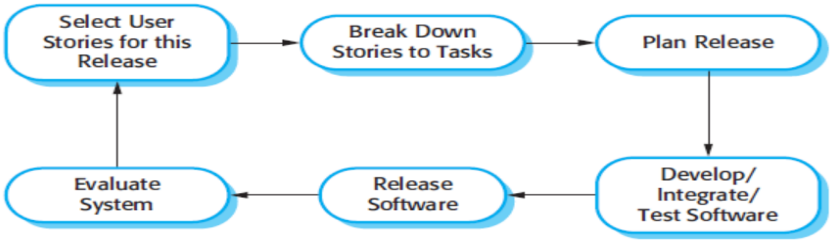
- ***Sprints***
    - consist of work units that are required to achieve a requirement defined in the backlog that must be fit into a predefined time-box Changes are not introduced during the sprint. Hence, the sprint allows team members to work in a short-term, but stable environment.
- *Scrum meetings*
    - are short (typically 15 minutes) meetings held daily by the Scrum team. Three key questions are asked and answered by all team members:
    - What did you do since the last team meeting?
    - What obstacles are you encountering?
    - What do you plan to accomplish by the next team meeting?
- *Demos*
    - deliver the software increment to the customer so that functionality that has been implemented can be demonstrated and evaluated by the customer.
    - It is important to note that the demo may not contain all planned functionality, but rather those functions that can be delivered within the time-box that was established.

SCRUM Advantages

1. The product is broken down into a set of manageable and understandable chunks.
2. Unstable requirements do not hold up progress.
3. The whole team has visibility of everything and consequently team communication is improved.
4. Customers see on-time delivery of increments and gain feedback on how the product works.
5. Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.

14a **Extreme programming** with diagram

- Extreme programming (XP) is perhaps the best known and most widely used of the agile methods.
- Coined by Beck (2000)
- Approach was developed by pushing recognized good practice, such as iterative development, to 'extreme' levels.
    - For example, in XP, several new versions of a system may be developed by different programmers, integrated and tested in a day.
- New versions may be built several times per day;
- Increments are delivered to customers every 2 weeks;
- All tests must be run for every build and the build is only accepted if tests run successfully.



Extreme Programming Release Cycle

Extreme Programming Practices

Incremental planning	Requirements are recorded on Story Cards and the Stories to be included in a release are determined by the time available and their relative priority. The developers break these Stories into development ÕTasksÕ
Small Releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple Design	Enough design is carried out to meet the current requirements and no more.
Test first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.
Pair Programming	Developers work in pairs, checking each otherÕs work and providing the support to always do a good job.
Collective Ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers own all the code. Anyone can change anything.
Continuous Integration	As soon as work on a task is complete it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of over-time are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site Customer	A representative of the end-user of the system (the Customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

XP and Agile Principles

1. Incremental development is supported through small, frequent system releases.
2. Customer involvement means full-time customer engagement with the team.
3. People not process through pair programming, collective ownership and a process that avoids long working hours.
4. Change supported through regular system releases.
5. Maintaining simplicity through constant refactoring of code.

Testing in XP

- To avoid some of the problems of testing and system validation, XP emphasizes the importance of program testing.
- XP includes an approach to testing that reduces the chances of introducing undiscovered errors into the current version of the system.

14b **Key features of testing in XP are:**

1. Test-first development,
2. Incremental test development from scenarios,
3. User involvement in the test development & validation,
4. The use of automated testing frameworks.

This approach does not necessarily lead to thorough program testing. There are three reasons for this:

1. Programmers prefer programming to testing and sometimes they take shortcuts when writing tests.
2. Some tests can be very difficult to write incrementally.
3. It difficult to judge the completeness of a set of tests. Although you may have a lot of system tests, your test set may not provide complete coverage.

14c **Pair Programming**

- In XP, programmers work in pairs, sitting together to develop code, helps develop common ownership of code and spreads knowledge across the team. It serves as an informal review process as each line of code is looked at by more than 1 person.
- It encourages refactoring as the whole team can benefit from this.
- Measurements suggest that development productivity with pair programming is similar to that of two people working independently.
- Pair programming is a collaborative approach to coding where two developers work together at one workstation.
- One person, the "driver," writes the code, while the other, the "navigator," reviews each line as it's written, providing guidance and catching errors.
- The two regularly switch roles to maintain engagement and improve code quality.
- Pair programming is a collaborative approach to coding where two developers work together at one workstation.
- One person, the "driver," writes the code, while the other, the "navigator," reviews each line as it's written, providing guidance and catching errors.
- The two regularly switch roles to maintain engagement and improve code quality.
- Pair programming is often used in Agile methodologies, and it works best when there's a clear understanding of roles and when participants switch frequently to maintain balance.
- **Key benefits of pair programming include:**

1. **Improved code quality**: Immediate feedback helps to catch bugs and logical errors early.
2. **Knowledge sharing**: Both participants learn from each other's skills, techniques, and problem-solving approaches.
3. **Increased collaboration**: Working together encourages communication, & trust.
4. **Reduced context switching**: Since both developers focus on the same task, there's less chance for distraction.