SPM-Module-1(Chapter-2)

Software Processes: Software process models, Process activities, coping with change, the rational unified process.
Agile software development -Agile methods, Scrum, Extreme Programming.

Software Processes:

- A software process is a set of related activities that leads to the production of a software product.
- There are many different software processes, but all must include four activities that are fundamental to software engineering:
  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.
- There are also supporting process activities such as documentation and software configuration management.
- As well as activities, process descriptions may also include:
  1. **Products:**
     - These are the outcomes of a process activity.
     - Example: the outcome of the activity of architectural design may be a model of the software architecture.
  2. **Roles**:
     - The responsibilities of the people involved in the process.
     - Examples: project manager, configuration manager, programmer, etc.
  3. **Pre- and post-conditions:** These are statements that are true before and after a process activity has been carried out or a product produced.
     - Example: before architectural design begins, a pre-condition may be that all requirements have been approved by the customer; after this activity is finished, a post-condition might be that the UML models describing the architecture have been reviewed.
- software processes are categorized as
  - **Plan-driven Processes:**
    - All activities are well planned- when should work start and end(schedule), how many people should work on particular activity, budget etc.
    - Here, it is difficult to change requirements because as planning is done already, if we change any requirement, it will affect whole planning.
  - **Agile Processes:**
    - Planning is not finalized,it is incremental.we can change plan whenever required(acc to our requirements).
    - We can easily change customer requirements as planning is not finalized.
    - i.e.,we can adopt ant type of changes as required.
- There are no right or wrong software processes. Based on the type of software we are developing, we can select plan-driven or agile processes.

Software Process Models

- A software process model is a simplified representation of a software process.
- Each process model represents a process from a particular perspective, and thus provides only partial information about that process.

  Generic Process Models:
  - Also called Process Paradigms.
  - *These represents process from a architectural perspective.*
  - These generic models are not definitive descriptions of software processes.
  - There are 3 generic process models.
    1) ***Waterfall model***
       - Plan-driven model.
       - Separate and distinct phases of specification and development.

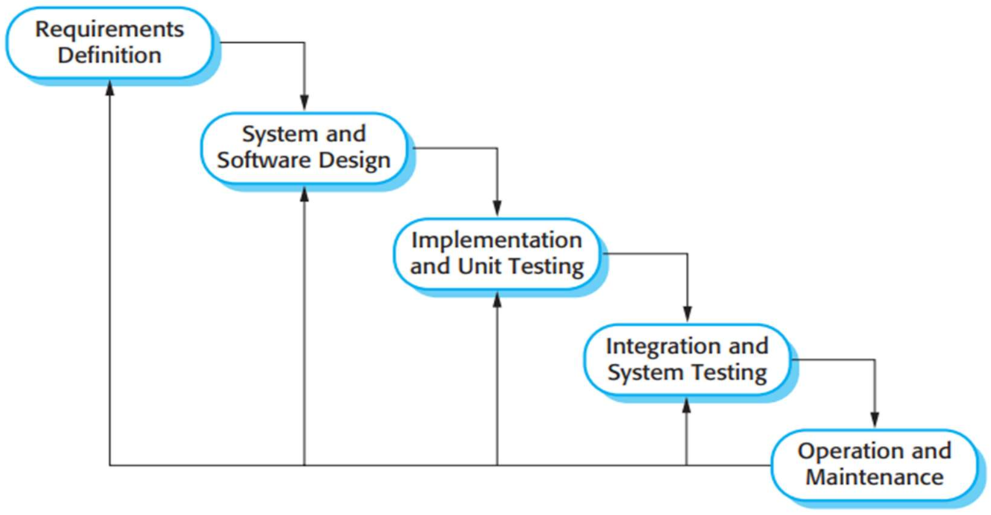
   - *All the phases will be implemented separately.*

2) ***Incremental developmentl***
   - May be plan-driven or agile.
   - Specification, development and validation are interleaved.
3) ***Reuse-oriented Software Engineering***
   - May be plan-driven or agile.
   - *The software is assembled from existing components.*

1.The waterfall model:

- The waterfall model is an example of a plan-driven process. plan and schedule all of the process activities before starting work on them.
- Because of the cascade from one phase to another, this model is known as the 'waterfall model' or software life cycle.
- This takes the fundamental process activities of specification, development, validation, and evolution and represents them as separate process phases such as requirements specification, software design, implementation, testing, and so on.
- The principal stages of the waterfall model directly reflect the 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 allocates the requirements to either hardware or softwaresystems by establishing an overall system architecture. Software design involves identifyingand describing the fundamental software system abstractions and their relationships.*
3. ***Implementation and unit testing***
   - *During this stage, the software design is realized as a set of programs or 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 of the life cycle, improving the implementation ofsystem units and enhancing the system's services as new requirements are discovered.*


Requirements Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

- **Waterfall model problems:**
  - *Inflexible partitioning of the project into distinct stages makes it difficult to respond tochanging customer requirements.*
  - *This model is only appropriate when the requirements are well-understood and changes will be*

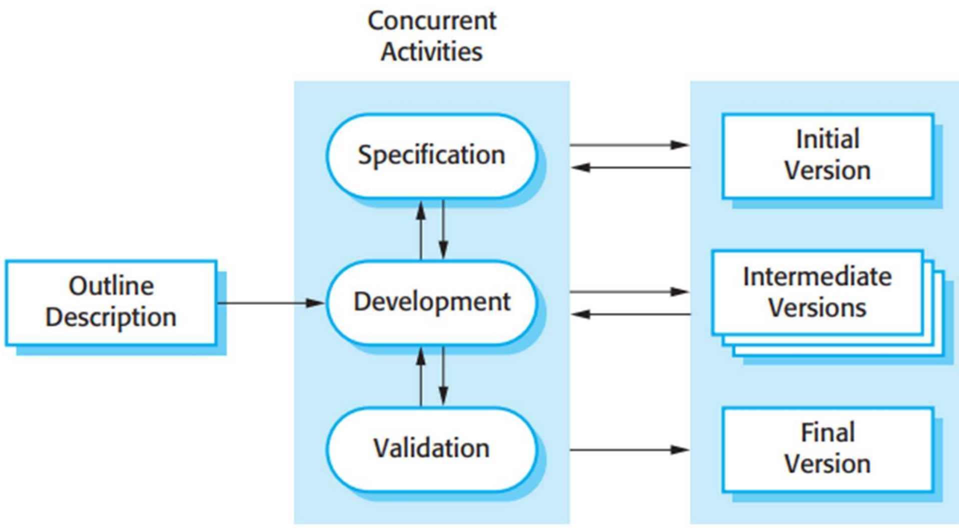
fairly limited during the design process.

- **Advantages of Waterfall Model:**
  - Simple and easy to understand.
  - Each phase has specific deliverables and review process, so quite easy to manage.
  - Phases are implemented separately and completed one at a time.
  - phases do not overlap.
  - Suitable for smaller projects where requirements are precisely defined and well understood.

- **Disadvantages of Waterfall Model:**
  - Once an application is in the testing stage, it is very difficult to go back and change.
  - No effective software is produced until last, throughout the lifecycle.(product available at the end of the phase).
  - High amount of risk and uncertainty.
  - Not good for complex and object-oriented projects.
  - Poor model for long projects.
  - Not suitable for moderate to high risk of changing.
  - Costly and time consuming.

2.*Incremental development*

- *It is a fundamental part of agile approaches.*
- *It is better than a waterfall model for most business, e-commerce, and personal systems.*
- Incremental development in some form is now the most common approach for the development of application systems.
- This approach can be either plan-driven, agile, or, a mixture of these approaches.
- *Incremental development reflects the way that we solve problems.*
- Incremental development is based on the idea of developing an initial implementation, exposing this to user comment and evolving it through several versions until an adequate system has been developed.
- Specification, development, and validation activities are interleaved rather than separate, with rapid feedback across activities.
- By developing the software incrementally, it is cheaper and easier to make changes in the software as it is being developed..
- *Each increment or version of the system incorporates some of the functionality that is needed by the customer.*
- *The early increments of the system include the most important or most urgently required functionality. This means that the customer can evaluate the system at a relatively early stage in the development to see if it delivers what is required. If not, then only the current increment has to be changed and, possibly, new functionality defined for later increments.*



- **Incremental development has three important benefits, compared to the waterfall model:**
  1. *The cost of accommodating changing customer requirements is reduced.*
     - The amount of analysis and documentation that has to be redone is much less than is required

with the waterfall model.

2. *It is easier to get customer feedback on the development work that has been done.*
   - Customers can comment on demonstrations of the software and see how much has been implemented.
3. *More rapid delivery and deployment of useful software to the customer is possible.*
   - *Customers are able to use and gain value from the software earlier than is possible with a waterfall process.*

➢ ***Incremental Development has two problems:***

1. *The process is not visible.*
   ➢ *Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of thesystem.*
2. *System structure tends to degrade as new increments are added.*
   ➢ *Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.*

3. Reuse-oriented Software Engineering

➢ In most software projects, there is some software reuse. This often happens informally when people working on the project of known designs or code that are similar to what is required. They look for these, modify them as needed, and incorporate them into their system.

➢ This informal reuse takes place irrespective of the development process that is used, whether the development process is traditional, agile, or hybrid.

➢ *Software development increasingly focuses on systematically reusing existing components to enhance efficiency.*

➢ *Reuse-oriented approaches depend on:*
   - A large repository of reusable components.
   - A framework for integrating these components.

➢ There are three types of software component that may be used in a reuse-oriented process:

1. **Web Services**:
   - Developed to meet standard protocols and can be invoked remotely.
   - Example: REST APIs or SOAP-based services.
2. **Collections of Objects**:
   - Packaged objects designed for integration within component frameworks such as .NET or J2EE.
   - Example: Java Beans in J2EE or .NET class libraries.
3. **Stand-alone Software Systems (COTS)**:
   - Configured for use in a particular environment.
   - Commercial off-the-shelf software that provides specialized functionality.
   - Example: MS Office for document management, or QuickBooks for accounting.

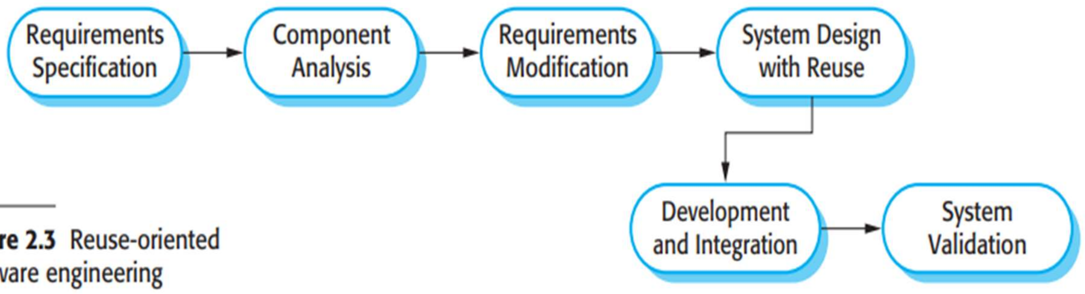

Requirements Specification
Component Analysis
Requirements Modification
System Design with Reuse
Development and Integration
System Validation

Figure 2.3 Reuse-oriented software engineering

- **Stages in Reuse-Oriented Process**
    1. **Component Analysis:** Search for existing components to satisfy the requirements. Exact matches are rare; selected components often cover only part of the functionality.
    2. **Requirements Modification:** Adjust initial requirements based on available components.If requirements can't be modified, return to component analysis to seek alternatives.
    3. **System Design with Reuse:** The system's framework is either newly designed or adapted from existing frameworks. Incorporates identified reusable components and designs additional components where necessary.
    4. **Development and Integration:** Develop software for functionalities not covered by reused components. Integrate all reusable components and any custom-developed software.

- **Advantages of Reuse-Oriented Development**
    1. **Reduced Development Effort:** Less new code is written, leading to faster completion of projects.
    2. **Cost Savings:** Decreases development and testing costs.
    3. **Risk Reduction:** Established, tested components minimize risks of bugs or failures.
    4. **Faster Delivery:** Speeds up the time-to-market due to pre-existing, reusable modules.

- **Challenges and Limitations**
    1. **Requirements Compromises**:
        - Reusing components may force deviations from initial user requirements.
        - This could lead to systems that partially meet user expectations.
    2. **Loss of Control over System Evolution**:
        - Updates to reusable components (especially COTS) are controlled by external vendors.
        - Organizations using them might struggle to align with future system updates or modifications.

Coping with Change

- *The system requirements change as the business procuring the system responds to external pressures and management priorities change.*
- *Change adds to the costs of software development because it usually means that work that has been completed has to be redone. This is called rework.*
- ***There are two related 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. For example, a prototype system may be developed to show some key features of the system to customers. They can experiment with the prototype and refine their requirements before committing to high software production costs.

    2. **Change tolerance**, where the process is designed so that changes can be accommodated at relatively low cost. This normally involves some form of incremental development.

- ***There are 2 ways of coping with change and 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. This supports both change avoidance and change tolerance. It avoids the premature commitment to requirements for the whole system and allows changes to be incorporated into later increments at relatively low cost.

1. 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:*
    1. In the requirements engineering process, a prototype can help with the elicitation and validation of system requirements.
    2. In the system design process, a prototype can be used to explore particular software solutions and to support user interface design.
- *System prototypes allow users to see how well the system supports their work.*
- *They may get new ideas for requirements and find areas of strength and weakness in the software. They may then propose new system requirements.*
- *A system prototype may be used while the system is being designed to carry out design experiments to check the feasibility of a proposed design.*
- *For example, a database design may be prototyped and tested to check that it supports efficient data access for the most common user queries.*
- *A process model for prototype development is shown in fig 5.1.*

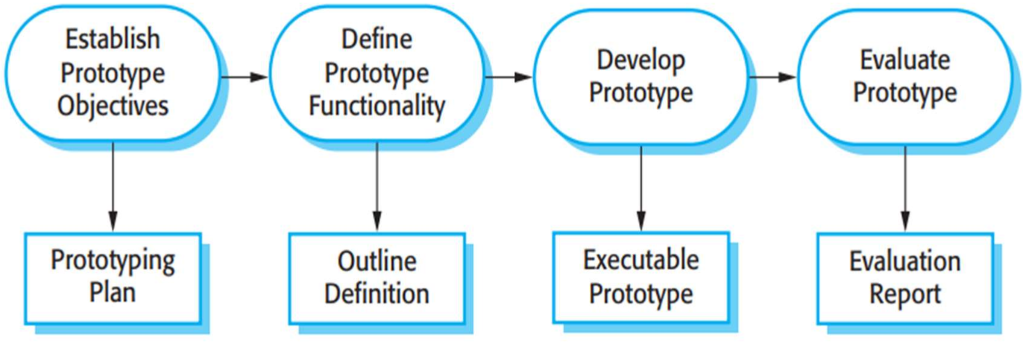


Fig 5.1: The process of prototype development

- *The objectives of prototyping should be made explicit from the start of the process.*
- *These may be to develop a system to prototype the user interface, to develop a system to validate functional system requirements, or to develop a system to demonstrate the feasibility of the application to managers.*
- *The next stage in the process is to decide what to put into and, perhaps more importantly, what to leave out of the prototype system.*
- *The final stage of the process is prototype evaluation.*
- *Developers are sometimes pressured by managers to deliver throwaway prototypes, particularly when there are delays in delivering the final version of the software.*
- *However, this is usually unwise:*
    1. It may be impossible to tune the prototype to meet non-functional requirements, such as performance, security, robustness, and reliability requirements, which were ignore during prototype development.
    2. Rapid change during development inevitably means that the prototype is undocumented. The only design specification is the prototype code. This is not good enough for long-term maintenance.
    3. The changes made during prototype development will probably have degraded the system structure. The system will be difficult and expensive to maintain.
    4. Organizational quality standards are normally relaxed for prototype development.

2. Incremental Delivery

- *Incremental delivery (Fig 5.2) is an approach to software development where some of the developed increments are delivered to the customer and deployed for use in an operational environment. In an incremental delivery process, customers identify, in outline, the services to be provided by the system.*

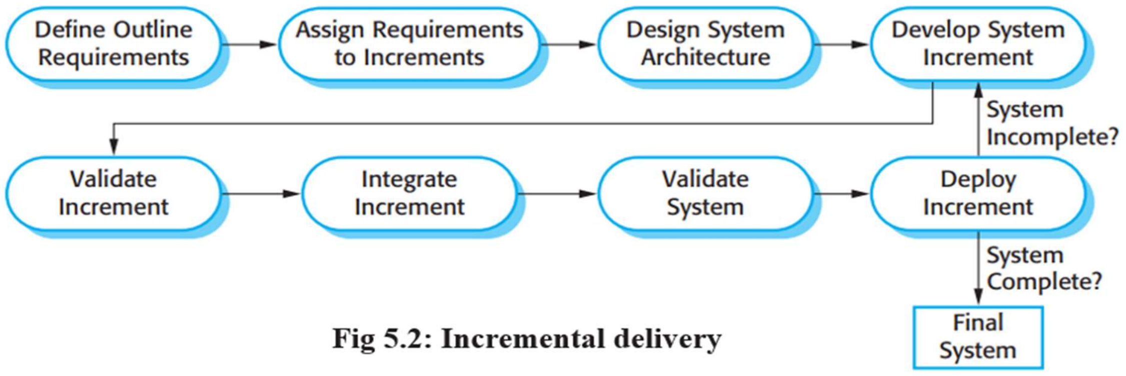


Fig 5.2: Incremental delivery

- *They identify which of the services are most important and which are least important to them.*
- *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.*

- ***Incremental delivery has a number of advantages:***
  1. Customers can use the early increments as prototypes and gain experience that informs their requirements for later system increments. Unlike prototypes, these are part of the real system so there is no re-learning when the complete system is available.
  2. Customers do not have to wait until the entire system is delivered before they can gain value from it. The first increment satisfies their most critical requirements so they can use the software immediately.
  3. The process maintains the benefits of incremental development in that it should be relatively easy to incorporate changes into the system.
  4. As the highest-priority services are delivered first and increments then integrated, the most important system services receive the most testing. This means that customers are less likely to encounter software failures in the most important parts of the system.

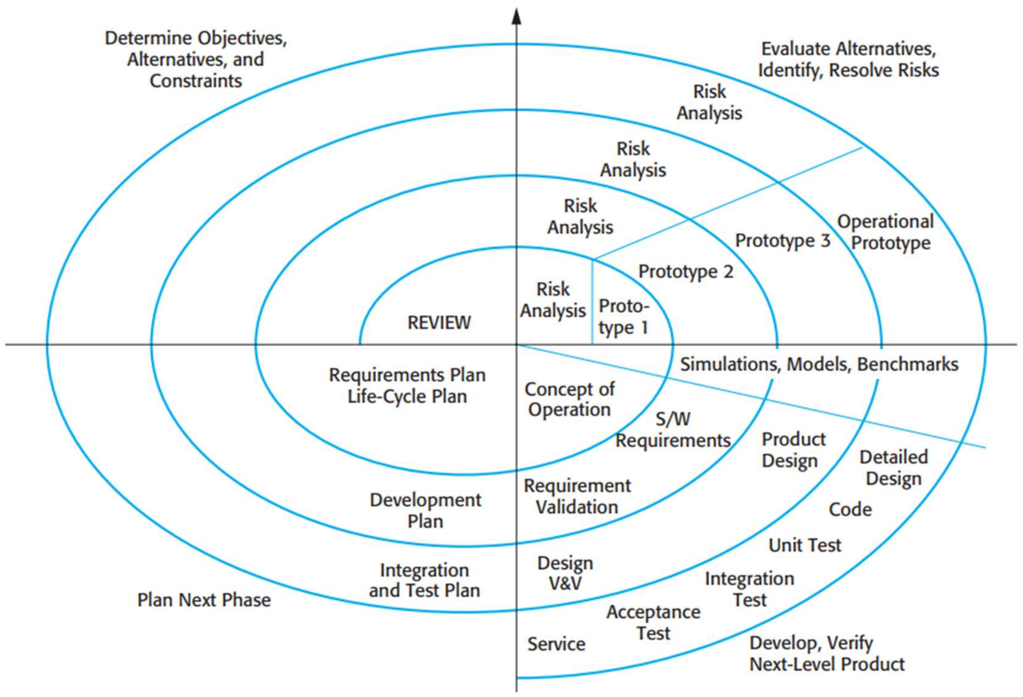
- ***However, there are problems with incremental delivery**:*
  1. Most systems require a set of basic facilities that are used by different parts of the system. As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.
  2. Iterative development can also be difficult when a replacement system is being developed. Users want all of the functionality of the old system and are often unwilling to experiment with an incomplete new system. Therefore, getting useful customer feedback is difficult.
  3. In the incremental approach, there is no complete system specification until the final increment is specified. This requires a new form of contract, which large customers such as government agencies may find difficult to accommodate.

3. Boehm's Spiral Model

- ➢ It is a risk-driven software process model, proposed by Boehm (1988).
- ➢ In this model, the software process is represented as a spiral, rather than a sequence of activities with some backtracking from one activity to another.
- ➢ 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 spiral model combines change avoidance with change tolerance.
- It assumes that changes are a result of project risks and includes explicit risk management activities to reduce these risks.
- Each loop in the spiral is split into four sectors:
  - **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.
  - **Risk assessment and reduction:**
    - For each of the identified project risks, a detailed analysis is carried out.
    - Steps are taken to reduce the risk.
    - If there is a risk that the requirements are inappropriate, a prototype system may be developed.
  - **Development and validation:**
    - After risk evaluation, a development model for the system is chosen.
  - **Planning:**
    - The project is reviewed, and a decision is 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.
- *The main difference between the spiral model and other software process models is its explicit recognition of risk.*
- *A cycle of the spiral begins by elaborating objectives such as performance and functionality. Alternative ways of achieving these objectives, and dealing with the constraints on each of them, are then enumerated. Each alternative is assessed against each objective and sources of project risk are identified.*
- *The next step is to resolve these risks by information-gathering activities such as more detailed analysis, prototyping, and simulation.*
- *Once risks have been assessed, some development is carried out, followed by a planning activity for the next phase of the process.*



- **When to use Spiral Model:**
  1. *When projects are large with complex requirements, where the risks and uncertainties are high* .
  2. *When there is a budget constraint and risk evaluation is important.*
  3. *When handling medium to high-risk projects.*
  4. *When handling long-term projects, because of potential changes to economic priorities as the requirements change with time.*
  5. *When the customer is not sure of their requirements.*
  6. *A new product line which should be released in phases to get enough customer feedback.*

- **Advantages of the Spiral Model:**
  1. *It helps in better **Risk management**.*
  2. *It provides **flexibility** by allowing iterative development.*
  3. It **continuously evaluates** and reviews the software throughout the development process.
  4. It helps in reducing the chances of expensive rework and contributes to **cost-effectiveness**.
  5. *It is suitable for handling **large** and **complex projects**.*
  6. It promotes **enhanced communication** with stakeholders.

- **Disadvantages of the Spiral Model:**
  1. *It is expensive and time consuming.*
  2. *It is not suitable for small or low risk projects.*
  3. *It can be complex to implement and understand.*
  4. The success of the spiral model heavily relies on the accuracy and effectiveness of risk analysis.
  5. *Spiral may go on indefinitely.*
  6. It may face difficulty when handling frequent and significant requirement changes.

RUP (The Rational Unified Process):

- The Rational Unified Process (RUP)is an example of a modern process model that has been derived from work on the UML and the associated Unified Software Development Process.
- It is a good example of a hybrid process model.
- It brings together elements from all of the generic process models , illustrates good practice in specification and design and supports prototyping and incremental delivery.
- The RUP recognizes that conventional process models present a single view of the process.
- In contrast, the RUP is normally described from three perspectives:

  1. **A dynamic perspective,** which shows the phases of the model over time.
  2. **A static perspective**, which shows the process activities that are enacted.
  3. **A practice perspective**, which suggests good practices to be used during the process.

1. **<u>A dynamic perspective:</u>**
   - The RUP is a phased model that identifies four discrete phases in the software process.
   - The phases in the RUP are more closely related to business rather than technical concerns. (In the waterfall model where phases are equated with process activities).
   - The four phases are:

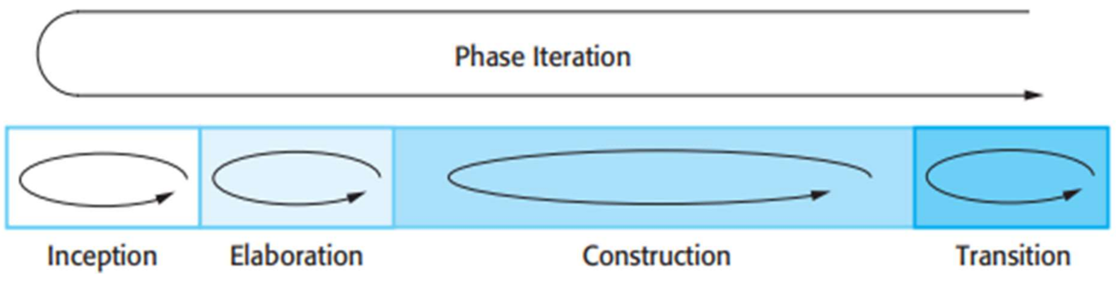
   (1) Inception:
   - The goal of the inception phase is to establish a business case for the system.
   - Identify all external entities (people and systems) that will interact with the system and define these interactions.

- Assess the contribution that the system makes to the business. If this contribution is minor, then the project may be cancelled after this phase.

(2) Elaboration:
- The goals of the elaboration phase are to develop an understanding of the problem domain, establish an architectural framework for the system, develop the project plan, and identify key project risks.
- On completion of this phase you should have a requirements model for the system, which may be a set of UML use-cases, an architectural description, and a development plan for the software.

(3) Construction:
- The construction phase involves system design, programming, and testing.
- Parts of the system are developed in parallel and integrated during this phase.
- On completion of this phase, a working software system and associated documentation is ready for delivery to users.

(4) Transition:
- The final phase of the RUP is concerned with moving the system from the development community to the user community and making it work in a real environment.
- On completion of this phase, a documented software system that is working correctly in its operational environment.



➢ Iteration within the RUP is supported in two ways.
1. Each phase may be carried out in an iterative way with the results developed incrementally.
2. The whole set of phases may also be carried out incrementally.

2. <u>A static perspective:</u>
- The static view of the RUP focuses on the activities that take place during the development process. These are called workflows in the RUP description.
- There are six core process workflows identified in the process and three core supporting workflows.
- **Workflow Descriptions:**

  1.Business modelling:
  - The business processes are modelled using business use cases.

  2.Requirements:
  - Actors who interact with the system are identified and use cases are developed to model the system requirements.

  3.Analysis and design:
  - A design model is created and documented using architectural models, component models, object models, and sequence models.

  4.Implementation:
  - The components in the system are implemented and structured into implementation sub-systems.
  - Automatic code generation from design models helps accelerate this process.

5.Testing:

- Testing is an iterative process that is carried out in conjunction with implementation.
- System testing follows the completion of the implementation.

6.Deployment:

- A product release is created, distributed to users, and installed in their workplace.

7.Configuration and change management:

- This supporting workflow manages changes to the system .

8.Project management:

- This supporting workflow manages the system development.

9.Environment:

- This workflow is concerned with making appropriate software tools available to the software development team.

➢ The RUP has been designed in conjunction with the UML, so the workflow description is oriented around associated UML models such as sequence models, object models, etc.

➢ The advantage in presenting dynamic and static views is that phases of the development process are not associated with specific workflows.

➢ In principle at least, all of the RUP workflows may be active at all stages of the process.

➢ In the early phases of the process, most effort will probably be spent on workflows such as business modelling and requirements and, in the later phases, in testing and deployment.

<u>**3. A practice perspective:**</u>

- The practice perspective on the RUP describes good software engineering practices that are recommended for use in systems development.
- Six fundamental best practices are recommended:
  1. **Develop software iteratively**:
     - Plan increments of the system based on customer priorities and develop the highest-priority system features early in the development process.
  2. **Manage requirements Explicitly:**
     - Document the customer's requirements and keep track of changes to these requirements. Analyze the impact of changes on the system before accepting them.
  3. **Use component-based architectures:**
     - 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 to software:**
     - Manage changes to the software using a change management system and configuration management procedures and tools.

➢ The RUP is not a suitable process for all types of development, e.g., embedded software development.

➢ However, it does represent an approach that potentially combines the three generic process models (Water fall model, Incremental development and Reuse-oriented software engineering).

➢ The most important innovations in the RUP are the separation of phases and workflows, and the recognition that deploying software in a user's environment is part of the process.

➢ Phases are dynamic and have goals.

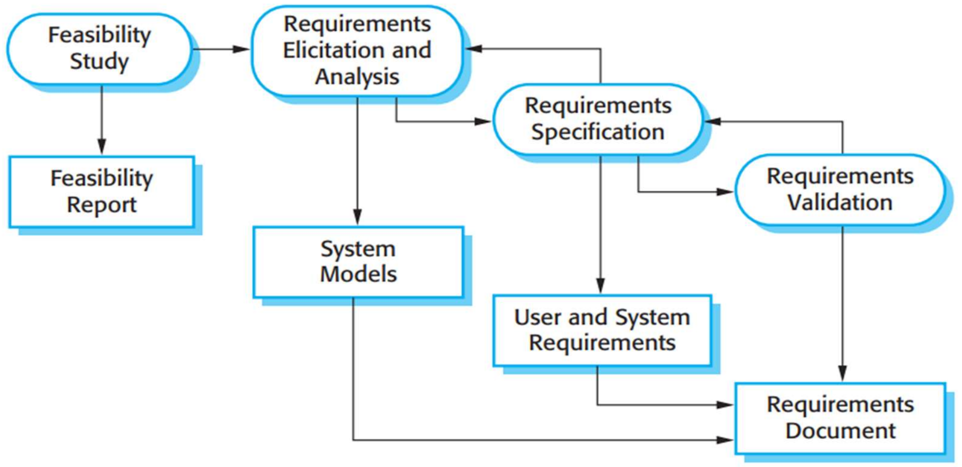
➢ Workflows are static and are technical activities that are not associated with a single phase but may be used throughout the development to achieve the goals of each phase.

Process activities

- Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.
- Software developers use a variety of different software tools in their work.
- Software tools provide process support by automating some process activities and by providing information about the software that is being developed.
- Software development tools (sometimes called Computer-Aided Software Engineering or CASE tools) are programs that are used to support software engineering process activities.
- These tools therefore include design editors, data dictionaries, compilers, debuggers, system building tools, etc.
- There are four basic process activities
  1. Software specification (Also called as Requirements engineering)
  2. Software design and implementation
  3. Software validation
  4. Software evolution

1. Software Specification or Requirements Engineering

- It 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 inevitably lead to later problems in the system design and implementation.
- The requirements engineering process aims to produce an agreed requirements document that specifies a system satisfying stakeholder requirements.
- Requirements are usually presented at two levels of detail.
  1. **End-users and customers** :
     - need a high-level statement of the requirements -- overview of the project focusing on goals, plans and outcomes.
  2. **system developers**:
     - need a more detailed system specification.
- There are four main activities in the requirements engineering process:
  1. Feasibility study
  2. Requirements elicitation and analysis
  3. Requirements specification
  4. Requirements validation



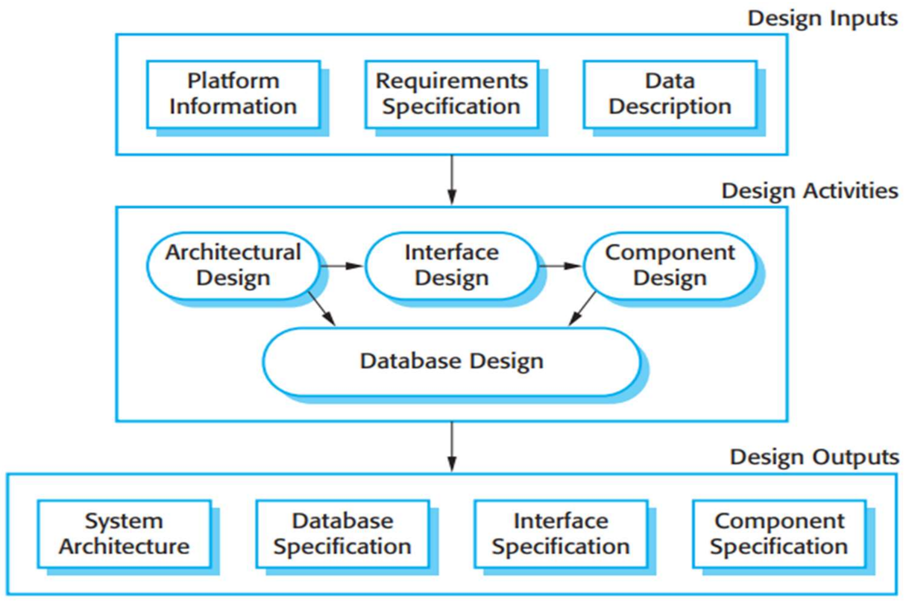
- The activities in the requirements process are not simply carried out in a strict sequence.
- Requirements analysis continues during definition and specification and new requirements come to light throughout the process.
- Here ,the activities of analysis, definition, and specification are interleaved.

1. **Feasibility study**
   - ➢ Identifying and estimating the user needs whether they may be satisfied using current software and hardware technologies.
   - ➢ Studying to check technically or financially possible to develop proposed system or not.
   - ➢ Taking decision to go ahead or not.

   Is it technically and financially feasible to build the system?
2. **Requirements elicitation and analysis**
   - ➢ Discovering or extracting the system requirements through
     - observation of existing systems,
     - discussions with potential users and procurers,
     - task analysis, and so on
   - ➢ Development of one or more system models and prototypes.
     - It helps us to understand the system to be specified.
3. **Requirements specification**
   - ➢ Translating the information gathered during the analysis activity into a document .
   - ➢ Document contains a set of requirements.
   - ➢ Two types of requirements may be included in this document.
     1. **User requirements**: abstract statements (i.e., summary) of the system requirements for the customer and end-user of the system.
     2. **System requirements:** more detailed description of the functionality to be provided.
4. **Requirements validation**
   - ➢ Checking the requirements for realism, consistency, and completeness.
   - ➢ Discovering and modifying the errors in the requirements document.

2.Software design and implementation

- ➢ It is the process of converting a system specification into an executable system.
- ➢ It always involves processes of
  - software design
  - programming (but, if an incremental approach to development is used, may also involve refinement of the software specification).
- ➢ A software design is a
  - ▪ description of the structure of the software to be implemented,
  - ▪ the data models and structures used by the system,
  - ▪ the interfaces between system components and,
  - ▪ sometimes, the algorithms used.



- Designers do not arrive at a finished design immediately but develop the design iteratively.
- They add formality and detail as they develop their design with constant backtracking to correct earlier designs.
- The diagram suggests that the stages of the design process are sequential.
- The design process activities are interleaved.
- Feedback from one stage to another and consequent design rework is inevitable in all design processes.

Design Inputs

- The platform information is an essential input to the design process, as designers must decide how best to integrate it with the software's environment.
- The 'software platform' is the environment in which the software will execute.
- Most software interfaces with other software systems. These include the operating system, database, middleware, and other application systems. These make up the 'software platform'.
- The requirements specification is a description of the functionality the software must provide and its performance and dependability requirements.
- If the system is to process existing data, then the description of that data may be included in the platform specification; otherwise, the data description must be an input to the design process so that the system data organization to be defined.

Design Activities

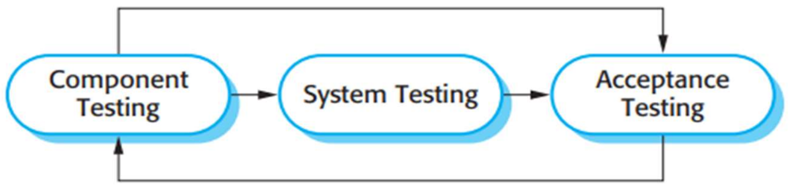
- The activities in the design process vary, depending on the type of system being developed.For example,
  - Real-time systems require timing design but may not include a database so there is no database design involved.
  - For information systems , four activities are used as a part of the design process .
- There are four activities that may be part of the design process for information systems:
  1. Architectural design
  2. Interface design
  3. Component design
  4. Database design

1. Architectural design, where you identify the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships and how they are distributed.
2. Interface design, where you define the interfaces between system components.
3. Component design, where you take each system component and design how it will operate.
4. Database design, where you design the system data structures and how these are to berepresented in a database.

3.Software validation

- Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.
- Involves checking and review processes and system testing.
- System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.
- Testing is the most commonly used V & V activity.

Stages of testing

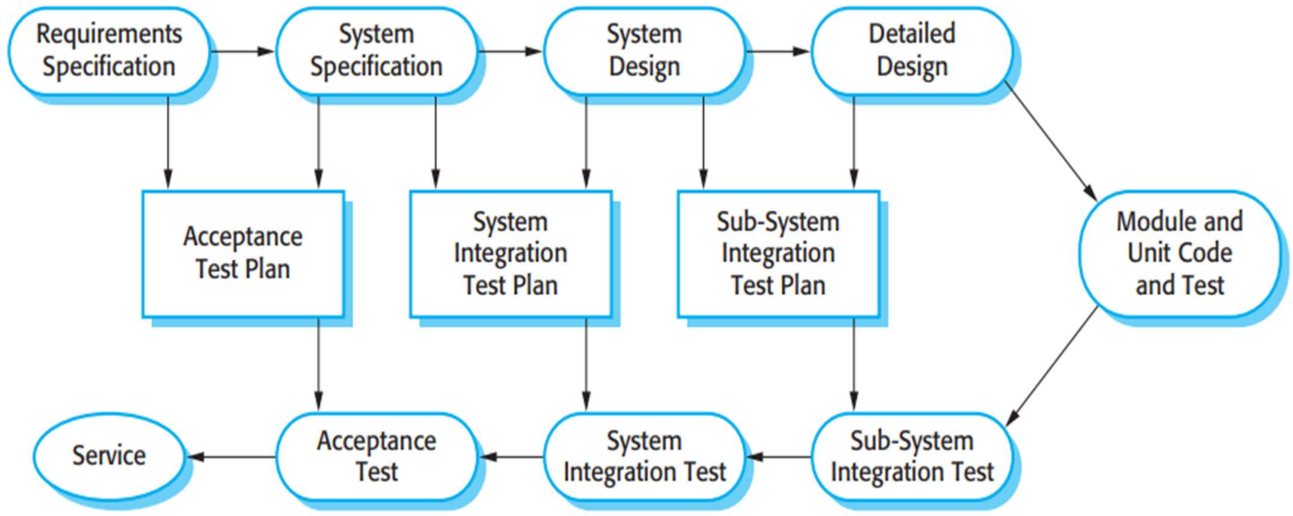


Testing stages:

- The stages in the testing process are:

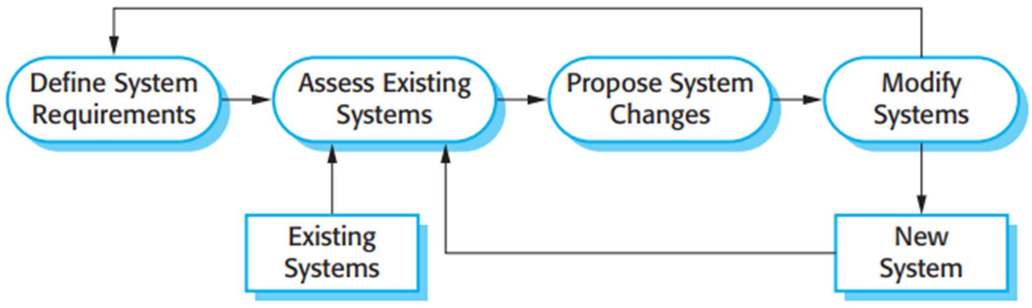
1. **Development or Component testing:**
   a. **In**dividual components are tested independently.
   b. Components may be functions or objects or coherent groupings of these entities.
2. **System testing:**
   a. Testing of the system as a whole.
   b. Testing of emergent properties is particularly important.
3. **Acceptance testing:**
   a. Testing with customer data to check that the system meets the customer's needs.

Testing phases in a plan-driven software process


Requirements Specification
System Specification
System Design
Detailed Design
Acceptance Test Plan
System Integration Test Plan
Sub-System Integration Test Plan
Module and Unit Code and Test
Service
Acceptance Test
System Integration Test
Sub-System Integration Test

4.Software evolution

- Software is inherently flexible and can change.
- As requirements change through changing business circumstances, the software that supportsthe business must also evolve and change.
- Although there has been a demarcation between development and evolution (maintenance) thisis increasingly irrelevant as fewer and fewer systems are completely new.



Agile Software Development:

Agile methods

→ *Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods.*

→ *These methods:*

1. *Focus on the code rather than the design.*
2. *Are based on an iterative approach to software development.*
3. *Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.*

→ *The aim of agile methods is to reduce overheads in the software process (by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.*

→ *Two most widely used agile methods are:* ***Extreme programming*** *and* ***Scrum****.*

→ *Agile methods have been very successful for some types of system development:*

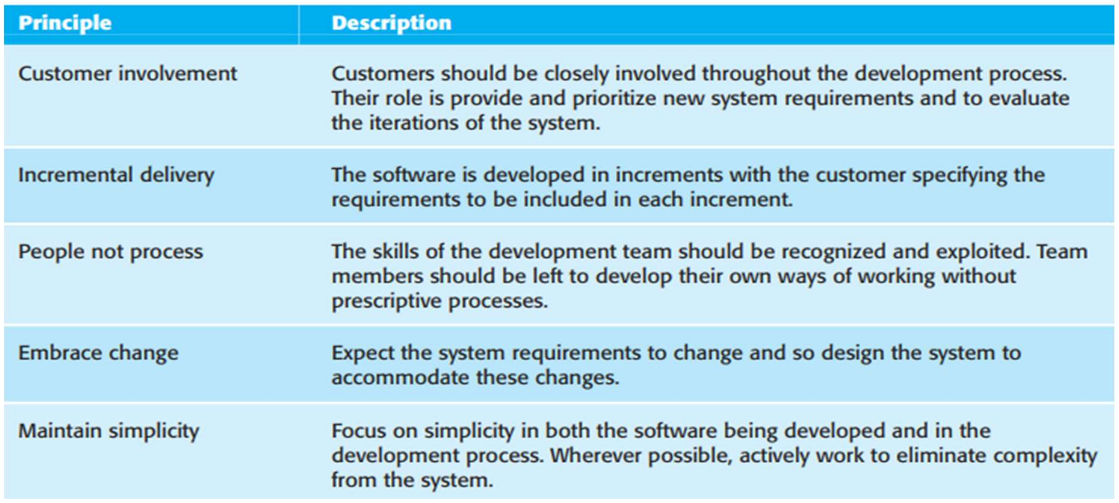
1. *Product development where a software company is developing a small or medium-sized product for sale.*
2. *Custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are not a lot of external rules and regulations that affect the software.*

1 The Agile manifesto: Values and Principles

- *The philosophy behind agile methods is reflected in the agile manifesto that was agreed on by many of the leading developers of these methods.*
- *This manifesto states that:*

  We are uncovering better ways of developing software by doing it and helping others do it.

- **Manifesto Values:**
  1. Individuals and interactions over processes and tools
  2. Working software over comprehensive documentation
  3. Customer collaboration over contract negotiation
  4. Responding to change over following a plan
- **Principles:**



Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and 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.

- ❖ Highest priority is to satisfy the customer through early and continuousdelivery of valuable software.
- ❖ Welcome changing requirements, even late in development. Agile processesharness change for the customers competitive advantage
- ❖ Deliver working software frequently, from a couple of weeks to couple ofmonths, with a preference to the shortest timescale
- ❖ Business people and developers must work together daily throughout theproject.
- ❖ Build projects around motivated individuals. Give them the environment andthe support they need, and trust them to get the job done
- ❖ The most efficient and effective method of conveying information to andwithin a development team is face-to-face conversation
- ❖ Working software is the primary measure of progress
- ❖ Agile processes promote sustainable development. The sponsors, developersand users should be able to maintain a constant pace indefinitely
- ❖ Continuous attention to technical excellence and good design enhances agility
- ❖ Simplicity – the art of maximizing the amount of work not done – is essential
- ❖ The best architectures, assignments and designs emerge from self organizingteams.
- ❖ At regular intervals the team reflects on how to become more effective, thentunes and adjusts its behavior accordingly.

Problems with agile methods

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 characterises agile methods.
3. Prioritising 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.

Agile methods and software maintenance

- → Most organizations spend more on maintaining existing software.
- → In agile methods two key issues are considered for maintenance:
  1. Are systems that are developed using an agile approach maintainable, even though its documentation minimized?
     - ❖ the lack of documentation should not be a problem in maintaining systems developed using an agile approach.
  2. Can agile methods be used effectively for evolving a system in response to customer change requests?
     - ❖ Yes. Agile practices are likely to be effective.
     - ❖ agile development process is a process of software evolution.
     - ❖ the main difficulty after software delivery is likely to be keeping customers involved in the process.
     - ❖ problem that is likely to arise is maintaining continuity of the development team.

2. *Plan-driven and Agile Development*

→ **Plan-driven development:**

- ❖ A plan-driven approach to software engineering is based around separate development stages with the outputs to be produced at each of these stages planned in advance.
- ❖ The outputs from one stage are used as a basis for planning the following process activity.
- ❖ *In a plan-driven approach, iteration occurs within activities with formal documentsused to communicate between stages of the process.*
- ❖ *For example, the requirements will evolve and, ultimately, a requirementsspecification will be produced.*
- ❖ *This is then an input to the design and implementation process.*
- ❖ *A plan-driven software process can support incremental development and delivery.*
- ❖ *It is perfectly feasible to allocate requirements and plan the design and developmentphase as a series of increments.*

→ **Agile development:**

- ❖ Specification, design, implementation, and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process.
- ❖ *Agile approaches to software development consider design and implementation to bethe central activities in the software process.*
- ❖ *They incorporate other activities, such as requirements elicitation and testing, intodesign and implementation.*
- ❖ *In an agile approach, iteration occurs across activities. Therefore, the requirementsand the design are developed together, rather than separately.*
- ❖ *An agile process is not inevitably code-focused, and it may produce some design documentation.*

→ *Fig 5.3 shows the distinctions between plan-driven and agile approaches to system*

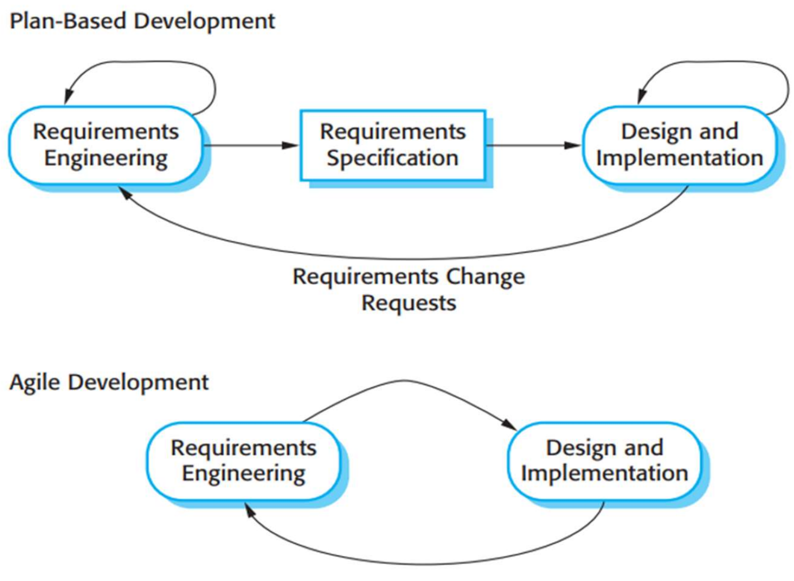

Plan-Based Development
Requirements Engineering
Requirements Specification
Design and Implementation
Requirements Change Requests
Agile Development
Requirements Engineering
Design and Implementation

Fig 5.3: Plan- driven and agile specification

3.Extreme Programming

 It is the best-known and most widely used agile method.

→ *Extreme Programming (XP) takes an 'extreme' approach to iterative development.*

- *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.*

→ *For example: In XP, several new versions of a system may be developed by different programmers, integrated and tested in a day.*

→ *In extreme programming, requirements are expressed as scenarios (called userstories), which are implemented directly as a series of tasks.*

 Programmers work in pairs and develop tests for each task before writing the code.

 All tests must be successfully executed when new code is integrated into the system.

→*There is a short time gap between releases of the system.*

 Fig 5.4 illustrates the XP process to produce an increment of the system that is beingdeveloped.

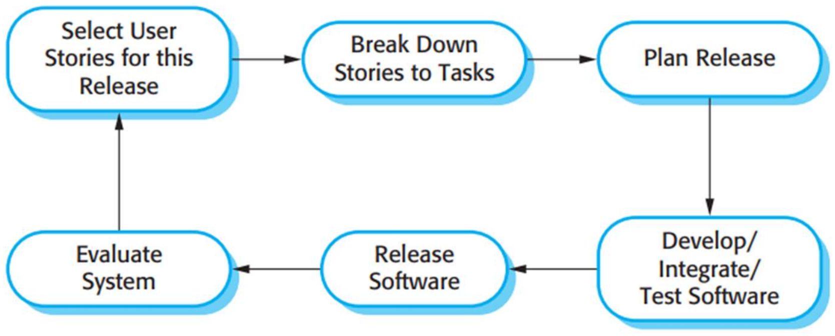
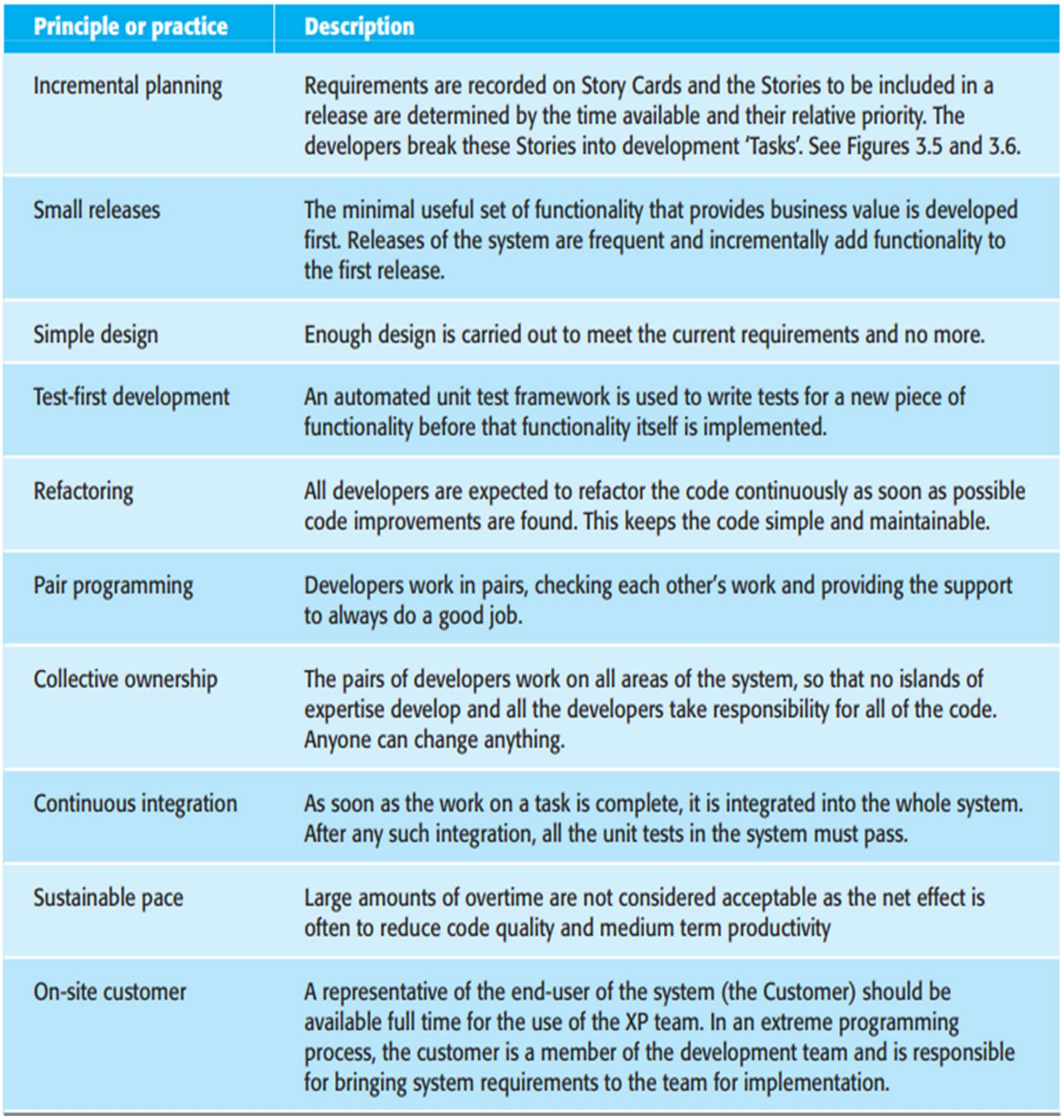

Select User Stories for this Release
Break Down Stories to Tasks
Plan Release
Develop/ Integrate/ Test Software
Release Software
Evaluate System

Fig 5.4: The extreme programming release cycle

 Extreme programming involves a number of ***practices*** *which reflect the principles of agile methods:*

1. Incremental development is supported through small, frequent releases of the system. Requirements are based on simple customer stories or scenarios that are used as a basis for deciding what functionality should be included in a system increment.
2. Customer involvement is supported through the continuous engagement of the customer in the development team. The customer representative takes part in the development and is responsible for defining acceptance tests for the system.
3. People are supported through pair programming, collective ownership of the system code, and a sustainable development process that does not involve excessively long working hours.
4. Change is embraced through regular system releases to customers, test-first development, refactoring to avoid code degeneration, and continuous integration of new functionality.
5. Maintaining simplicity is supported by constant refactoring that improves codequality and by using simple designs that do not unnecessarily anticipate future changes to the system.



Principle or practice	Description
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 'Tasks'. See Figures 3.5 and 3.6.
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 other's 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 take responsibility for all of the code. Anyone can change anything.
Continuous integration	As soon as the 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 overtime 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.

Fig 5.5: Extreme programming practices

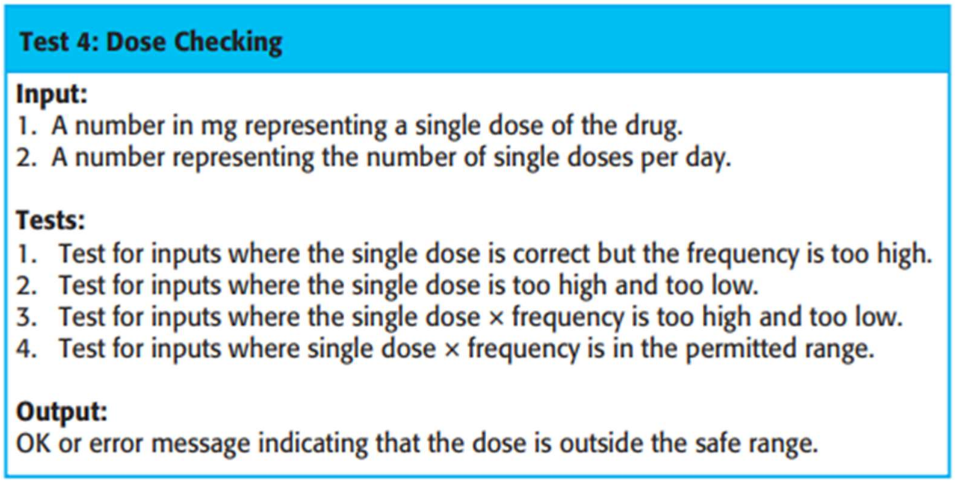
1.Testing in XP

→ *XP includes an approach to testing that reduces the chances of introducingundiscovered errors into the current version of the system.*

→ *The key features of testing in XP are:*

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

 Fig 5.6 is a shortened description of a test case that has been developed to check thatthe prescribed dose of a drug does not fall outside known safe limits.



Test 4: Dose Checking

Input:
1. A number in mg representing a single dose of the drug.
2. A number representing the number of single doses per day.

Tests:
1. Test for inputs where the single dose is correct but the frequency is too high.
2. Test for inputs where the single dose is too high and too low.
3. Test for inputs where the single dose × frequency is too high and too low.
4. Test for inputs where single dose × frequency is in the permitted range.

Output:
OK or error message indicating that the dose is outside the safe range.

Fig 5.6: Test case description for dose checking

- *The role of the customer in the testing process is to help develop acceptance tests forthe stories that are to be implemented in the next release of the system.*
- *Test automation is essential for test-first development.*
- *Tests are written as executable components before the task is implemented.*
- *These testing components should be standalone, should simulate the submission ofinput to be tested, and should check that the result meets the output specification.*
- *An automated test framework is a system that makes it easy to write executable testsand submit a set of tests for execution.*

→ ***XP testing difficulties:***

- ❖ *Test-first development and automated testing usually results in a large number of testsbeing written and executed.*
- ❖ *However, 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. For example, they may write incomplete tests that do not check for all possible exceptions that may occur.
  2. Some tests can be very difficult to write incrementally. For example, in a complex user interface, it is often difficult to write unit tests for the code that implements the 'display logic' and workflow between screens.
  3. It's difficult to judge the completeness of a set of tests. Although there are lot of system tests, the test set may not provide complete coverage. Crucial parts of the system may not be executed and so remain untested.

2. Pair Programming

- *Another innovative practice that has been introduced in XP is that programmers work in pairs to develop the software.*
- *They actually sit together at the same workstation to develop the software.*

→ *The same pairs do not always program together.*

→ *Pairs are created dynamically so that all team members work with each other during the development process.*

- ***The use of pair programming has a number of advantages:***

1. It supports the idea of collective ownership and responsibility for the system.
2. It acts as an informal review process because each line of code is looked at by at least two people. Code inspections and reviews are very successful in discovering a high percentage of software errors.
3. It helps support refactoring, which is a process of software improvement. The difficulty of implementing this in a normal development environment is that effort in refactoring is expended for long-term benefit. An individual who practices refactoring may be judged to be less efficient than one who simply carries on developing code. Where pair programming and collective ownership are used, others benefit immediately from the refactoring so they are likely to support the process.

4. Agile Project Management

- The principal responsibility of software project managers is to manage the project sothat the software is delivered on time and within the planned budget for the project.
- They supervise the work of software engineers and monitor how well the softwaredevelopment is progressing.
- The standard approach to project management is plan-driven.
- Agile project management requires a different approach, which is adapted to incremental development and the strengths of agile methods.
    - The Scrum approach is used

Fig 5.7 is a diagram of the Scrum management process.

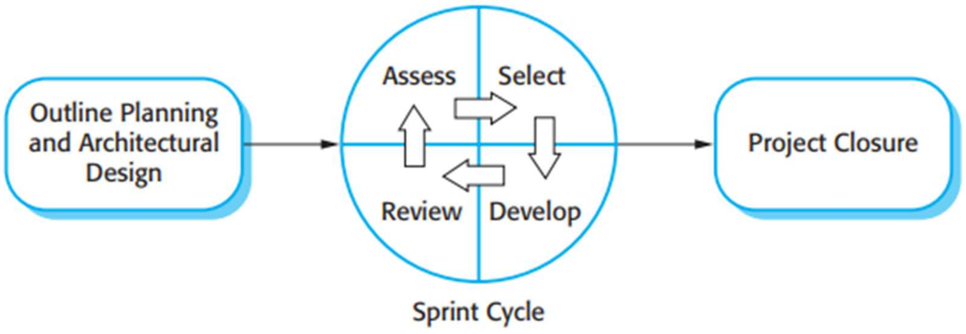


Fig 5.7: The Scrum process

 Scrum does not prescribe the use of programming practices such as pair programmingand test-first development.

 It can therefore be used with more technical agile approaches, such as XP, to providea management framework for the project.

 There are three phases in Scrum:

1. The first is an **outline planning phase** where you establish the general objectives for the project and design the software architecture.
2. This is followed by a **series of sprint cycles**, where each cycle develops an increment of the system.
3. Finally, the **project closure phase** wraps up the project, completes required documentation such as system help frames and user manuals, and assesses the lessonslearned from the project.

 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 a sprint, the completed functionality is delivered to stakeholders.

→ **Key characteristics of this Scrum process are as follows**:

1. Sprints are fixed length, normally 2–4 weeks. They correspond to the development of a release of the system in XP.
2. The starting point for planning is the product backlog, which is the list of workto be done on the project. During the assessment phase of the sprint, this is reviewed, and priorities and risks are assigned. The customer is closely involved in this process and can introduce new requirements or tasks at the beginning of each sprint.
3. The selection phase involves all the project team who work with the customer to select the features and functionality to be developed during the sprint.
4. 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.
5. At the end of the sprint, the work done is reviewed and presented to stakeholders. The next sprint cycle then begins.

 Advantages of Scrum:

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