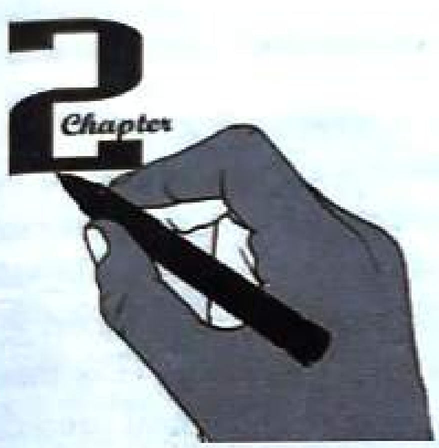

SOFTWARE PROCESS AND PROCESS MODEL

CHAPTER OUTLINE

After studying this chapter, the reader will be able to understand the

- Software Process
- Software Development Process Models
- Computer Aided Software Engineering (CASE)

Software Process

Software process also known as software methodology is the set of related activities that leads to cost effective and timeliness production of software. It is the set of activities that leads to the production of a software system from scratch or modifying existing software in a standard language like c, c++, java etc. so, a structured set of activities required to develop a software system like: specification, design, validation and evolution are called software process activities. These process activities are organized differently in different development model. How these activities are carried out depends on the type of software, people, and organizational structure involved. Following are major s/w process activities:

- Software specification
- Software design and implementation
- Software validation
- Software evolution

Software Specification

Software specification sometimes called requirement engineering is the process of understanding and defining what services are required from the system and identifying the constraints on the system's operation and development. In addition it also contains non functional requirements. Non-functional requirements impose constraints on the design or implementation (such as performance engineering requirements, quality standards, or design constraints). Software specification leads to production of requirement document that specifies a software satisfying stakeholder's requirement. It involves the activities like: feasibility study, requirement elicitation and analysis, requirement specification and requirement validation.

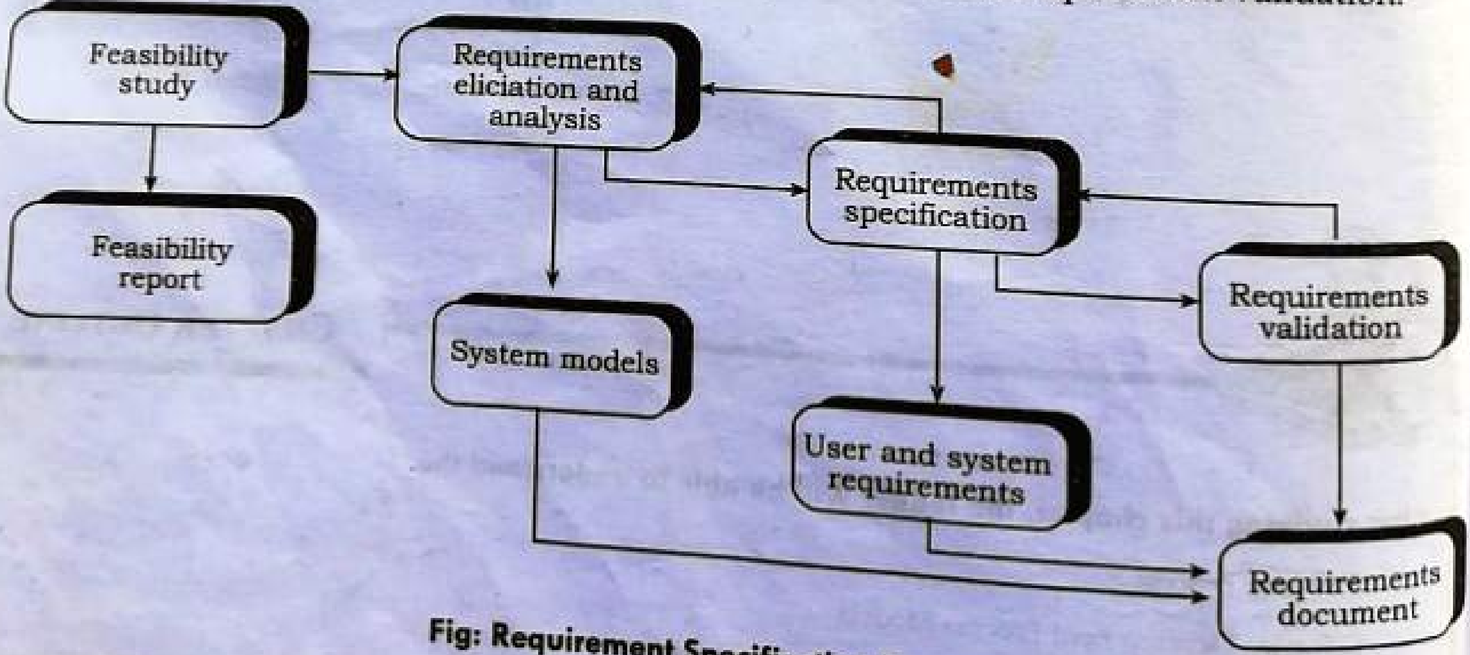

Fig: Requirement Specification Process

Software Design and Implementation

Software design and implementation is the stage in the software engineering process at which an executable software system is developed. Software design and implementation activities are invariably inter-leaved. - Software design is a creative activity in which you identify software components and their relationships, based on a customer's requirements. Implementation is the process of realizing the design as a program. A software design is a description of the structure of the software to be implemented, the data which is part of system, the interfaces between system component, sometimes algorithm used, logical structure of database etc. The implementation of the software development is the process of converting system specification and design into an executable system. The specific design process activities include:

- Architectural design
- Abstract specification
- Interface design
- Component design
- Data structure design
- Algorithm design

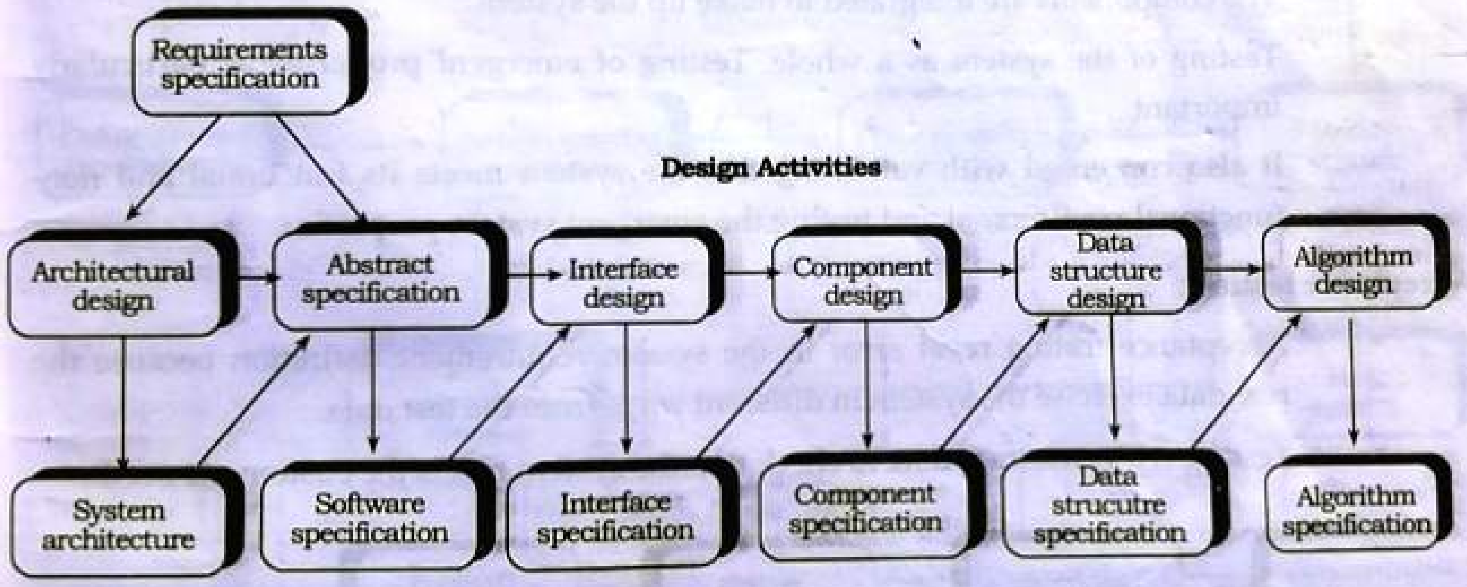

Fig: Design Products

Software Validation

Software Validation is a process of evaluating software product, so as to ensure that the software meets the pre-defined and specified business requirements as well as the end users/customers' demands and expectations. It is basically, performed with the intent to check that whether the developed software is built as per pre-decided software requirement specifications (SRS) and if it caters to fulfil the customers' actual needs in the real environment. Software validation generally means Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer. It is intended to show that the software does what the customer wants. i.e. Are we building right product? It involves checking process such as, inspection and reviews, at each stage of software process from user requirement to software development. It 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.

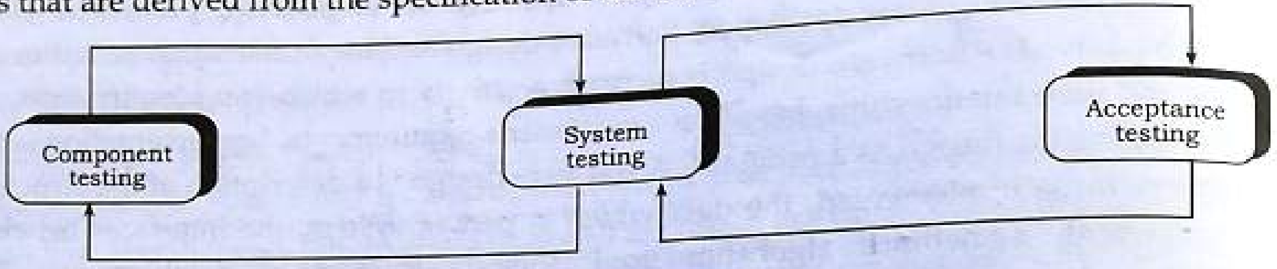

Fig: System Testing Process

Testing Stages

Component or unit testing

- Individual components are tested independently to ensure that they operate correctly.
- Components may be functions or objects or coherent groupings of these entities.

System testing

- The components are integrated to make up the system.
- Testing of the system as a whole. Testing of emergent properties is particularly important.
- It also concerned with validating that the system meets its functional and non-functional requirement and testing the emergent system properties.

Acceptance testing

- Acceptance testing revel error in the system requirement definition because the real data exercise the system in different ways from the test data.
- Testing with customer data to check that the system meets the customer's needs.

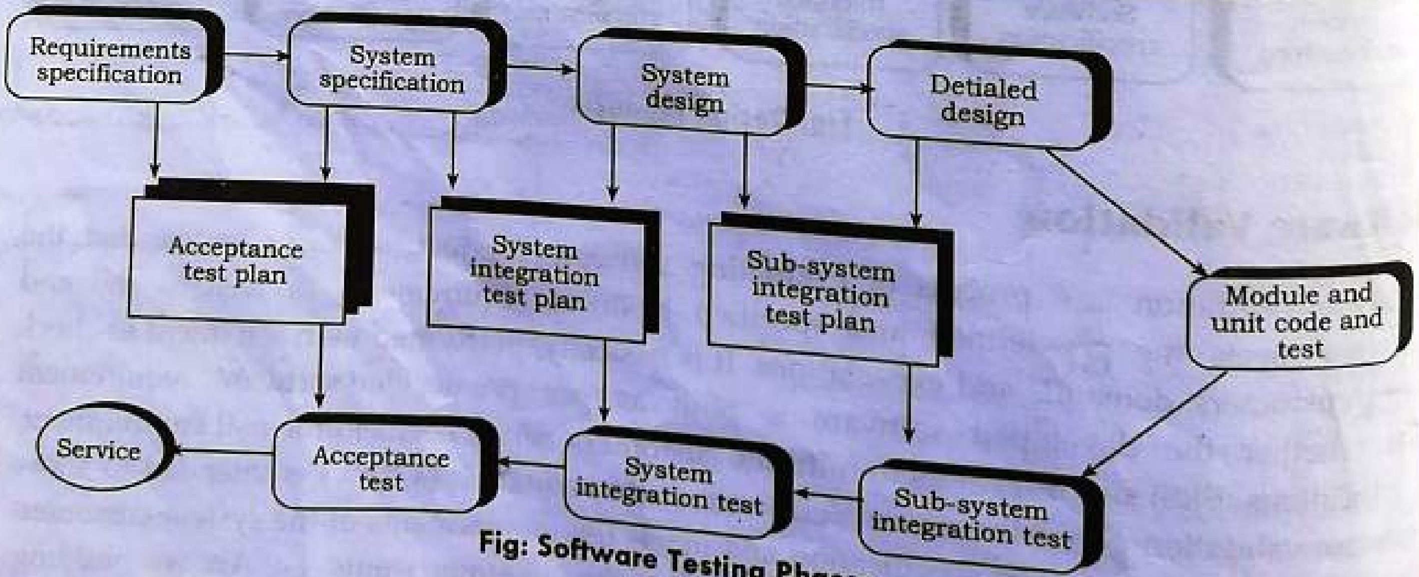
Requirements specification
System specification
System design
Detialed design
Acceptance test plan
System integration test plan
Sub-system integration test plan
Module and unit code and test
Service
Acceptance test
System integration test
Sub-system integration test

Fig: Software Testing Phases

Software Evolution

In software engineering, software evolution is referred to as the process of developing, maintaining and updating software for various reasons. Software changes are inevitable because there are many factors that change during the life cycle of a piece of software. It is a term which refers to the process of developing software initially, then timely updating it for various reasons, i.e., to add new features or to remove obsolete functionalities etc. The evolution process includes fundamental activities of change analysis, release planning, system implementation and releasing a system to customers. Some of these factors include:

- Requirement changes
- Environment changes
- Errors or security breaches
- New equipment added or removed
- Improvements to the system

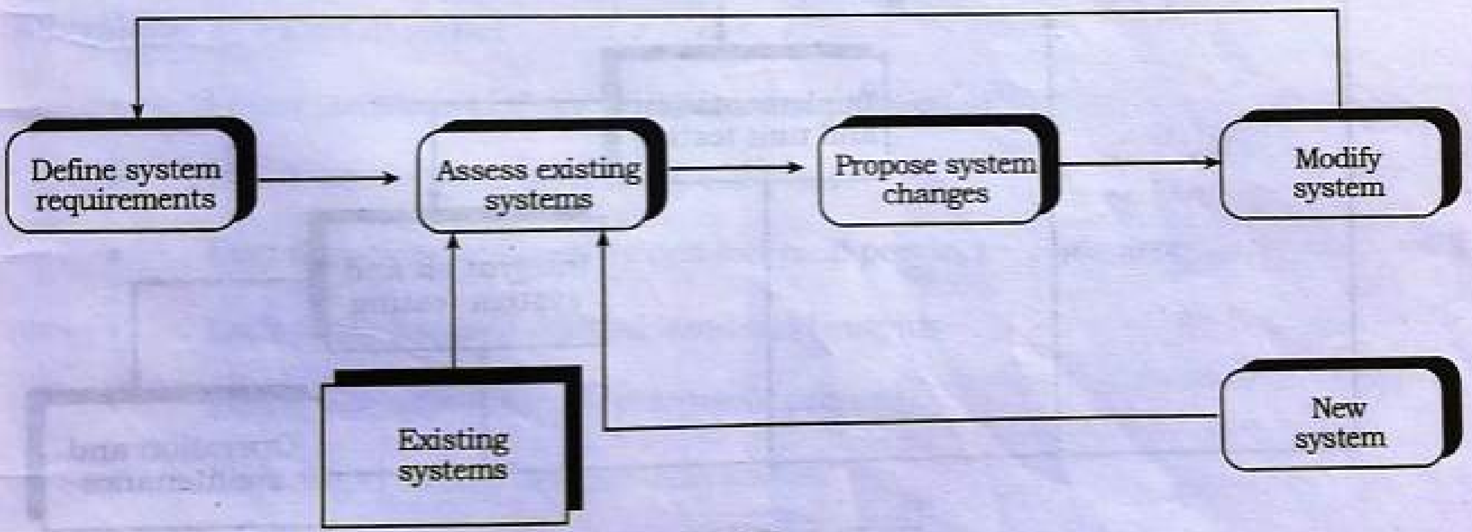

Fig: System Evolution

SOFTWARE DEVELOPMENT PROCESS MODELS

A software process model is an abstract diagrammatic representation of a software process in a simplified form. It represents the order in which the activities of software development will be undertaken. Each model represents a process from a specific perspective. Depending on the size and purpose of an organization, the systems do differ in terms of their technological complexity and organizational problems they are meant to solve. As there are different kinds of systems, a number of models are in existence that can be used in the development of an information system.

9866223214

Waterfall Model

This is the simplest software development life cycle model, which states that the phases are organized in a linear order. This model takes the fundamental process activities of specification, development, validation and evolution and represents them as separate process phases such as requirement specification, design, implementation, testing and so on.

Because of cascade from one phase to another, this model is known as waterfall model or software life cycle or linear sequential model. In this model the result of each phase is one or more documents that are approved and following phase should not start until the previous phase has finished these stages and feed information to each other. The principal stages or waterfall model are depicted in the following figure.

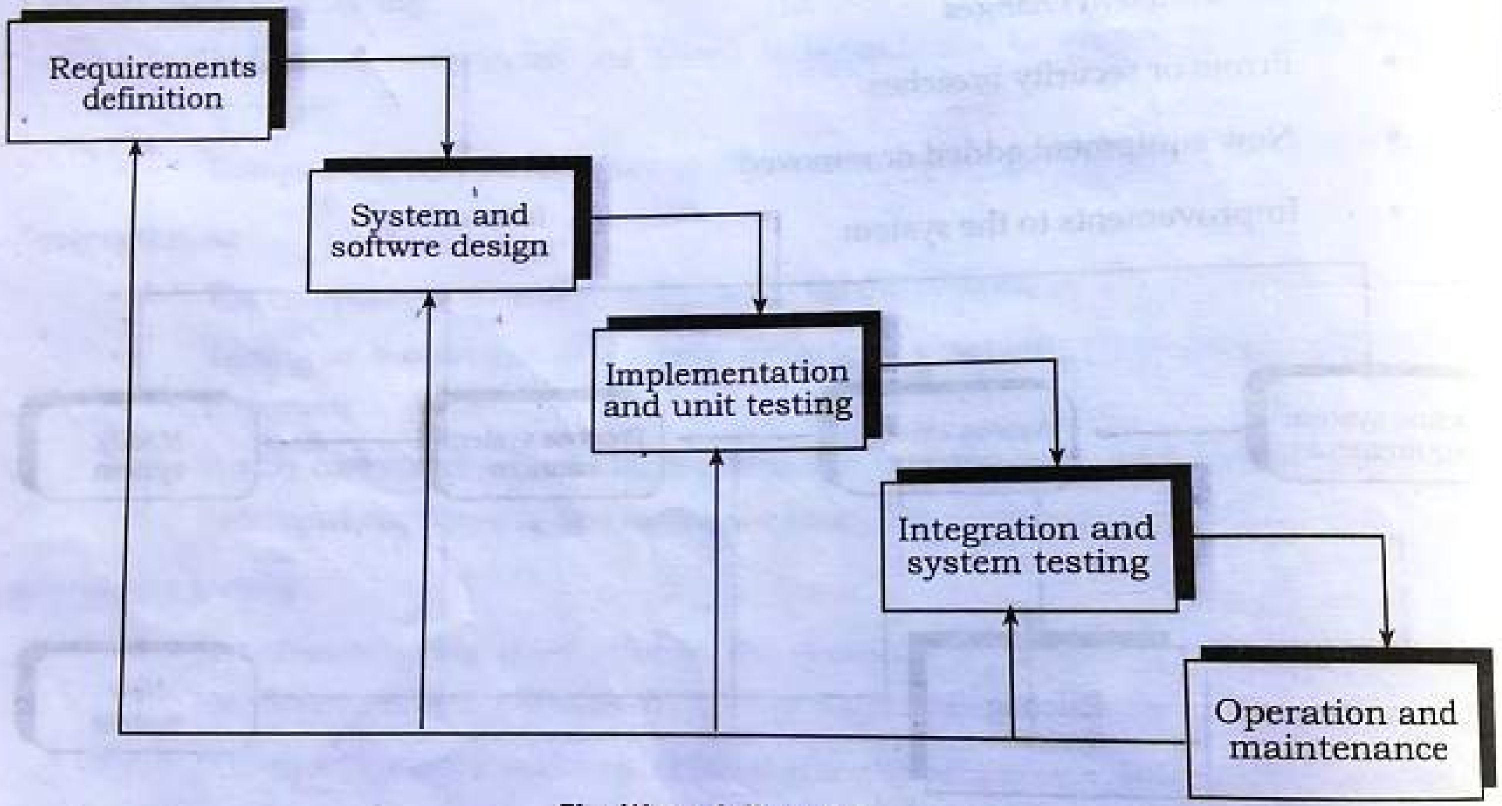
Requirements definition
System and softwre design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Fig: Water fall model

- **Requirements analysis and definition:** In this phase the complete service documentation will be prepared by the developers who will consult with the client. The system's service, constraints and goals are established by consultation with system users. They are often defined in detail and serve as a system specification. The developer will analyze the collected requirement to create the documentation both for developer and clients.
- **System and software design:** The software design based on approved documentation defines the software and hardware requirements for the software. The software architecture will be built which will include the module and their relationship, database and their relatives. Software design involves identifying and describing the fundamental software system abstractions and their relationship. At

the end of this phase developer will exactly know what they are going to develop and what the problems to be solved are.

- **Implementation and unit testing:** In this stage, requirement document and software design is transformed into set of programs and program units. Then individually developed units are tested to verify that each unit meets its specification.
- **Integration and system testing:** In this phase all the modules are integrated together to form a system 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.
- **Operation and maintenance:** This is the longest life-cycle phase. The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered earlier stages of the life-cycle, improving the implementation of system units and enhancing the system's services as new requirements are discovered.

Advantages of waterfall model

- Linear structure of this model is easy to understand.
- Development progress is easily estimated.
- Easy to understand even by non-technical person, i.e. customers.
- Each phase has well defined inputs and outputs.
- Easy to use as software development proceeds,
- Each stage has well defined deliverables.
- Helps the project manager in proper planning of the project.

Disadvantages of waterfall model

- The drawback of the model is the difficulty of accommodating change after the process is underway.
- Real projects rarely follow the waterfall model because changes can cause confusion.
- It is often difficult for the user to state all requirements explicitly.
- It has unnecessary delays; sometimes it may lead to "blocking states".
- It has inflexible partitioning of the project into distinct stages
- This model is only appropriate when the requirements are well-understood.

When to use the waterfall Model

- This model is used only when the requirements are very well known, clear and fixed.
- Product definition is stable.
- Technology is understood.
- If there are no ambiguous requirements
- Ample resources with required expertise are available freely
- When the project is of small size.

Evolutionary Development Model/Incremental Development Model

There is growing recognition that software, like all complex systems, evolves over a period of time. Business and product requirements often change as development proceeds, making a straight path to an end product unrealistic; tight market deadlines make completion of a comprehensive software product impossible, but a limited version must be introduced to meet competitive or business pressure; a set of core product or system requirements is well understood, but the details of the product or system extensions have yet to be defined. In these similar situations, software engineers need a process model that has been explicitly designed to accommodate a product that evolves over time. Evolutionary model is a version based software model in which iterative process starts with a simple implementation of small set of software requirements and enhances an evolving version until the complete software is implemented and deployed. Evolutionary model is iterative in nature but unlike prototyping this model focuses on delivery of operational product with each increment.

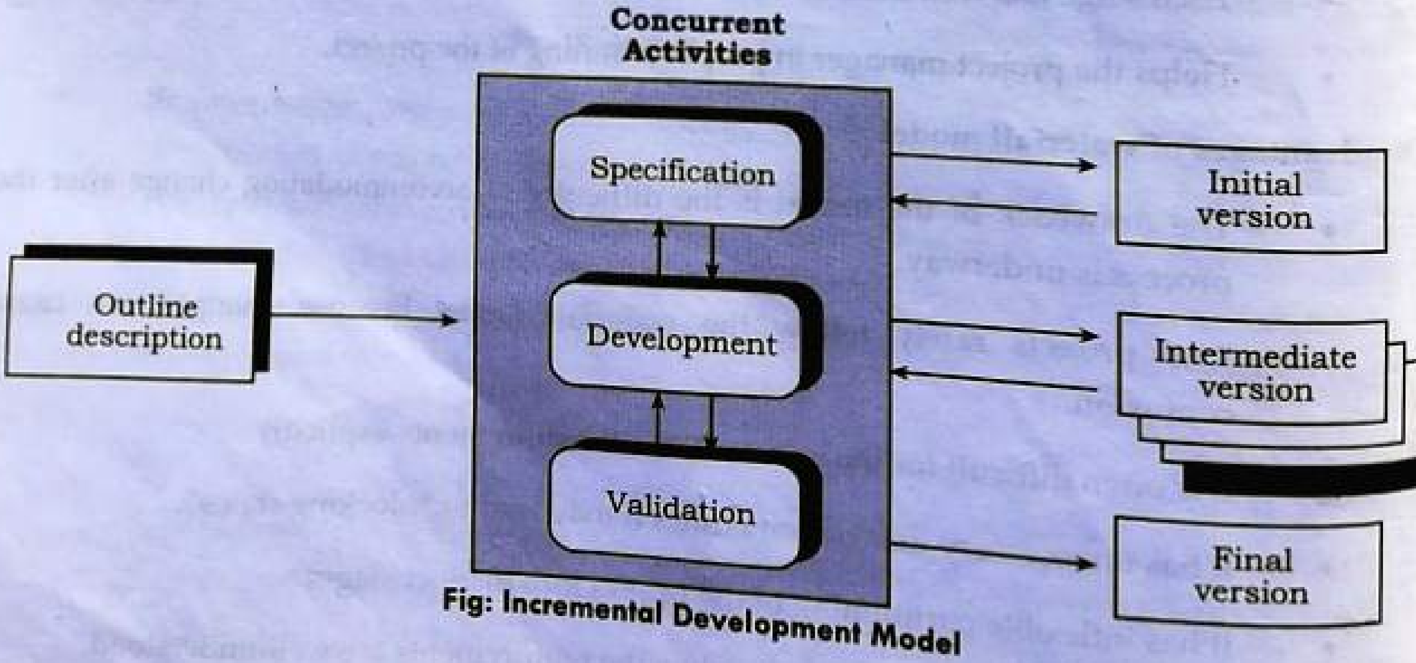

Fig: Incremental Development Model

Main characteristics

- The phases of the software construction are interleaved.
- Feedback from the user is used as throughout to the entire process.
- The software product is refined through many versions.

Types of Evolutionary Development

- Exploratory development: This evolutionary development starts with those requirements of the software which are clearly understood and software evolves by adding new requirements as they are proposed by the stakeholders of the software system.
- Throw-away prototyping: This evolutionary development starts with those requirements of the software which are poorly understood and develop a better understanding of the requirements of the software system.

Advantages

- Deals constantly with changes Provides quickly an initial version of the system
- Involves all development teams

Disadvantages

- Quick fixes may be involved,
- "Invisible" process, not well-supported by documentation,
- The system's structure can be corrupted by continuous change,
- Special tools and techniques may be necessary,
- The client may have the impression the first version is very close to the final product and thus be less patient.

Applicability

- This model can be used when the requirements of the complete system are not clearly defined and understood.
- Major requirements must be defined; however, some details can evolve with time.
- There is a need to get a product to the market early.
- When a new technology is being used
- When resources with needed skill set are not available
- When there are some high risk features and goals.

- It is applicable when the client and the developer agree on a "rapid prototype" tha will be thrown away,
- This model is good for small and medium-sized software systems.

Component Based Software Engineering (CBSE)

Component-based software engineering is a procedure that accentuates the design and development of computer-based systems with the help of existing reusable softwar components. It is also known as reuse oriented model. In this rather than developing system from scratch, we search for existing reusable components and design the desired system in terms of those components. The advantage of the component-based approach is providing the reusability of these components. With component-based software engineering (CBSE), it may be argued that software development risk is reduced, as one is reusing existing tried-and-testec software rather than software developed from scratch. Component-based developmen techniques involve procedures for developing software systems by choosing ideal off-the-shel components and then assembling them using well-defined software architecture. Componen based software engineering is a approach which mainly depends on building systems from the existing components and, providing support for the development of systems as assemblies of components.

Basic Principles of the Component-based Software Engineering

1. **Reusability:** It means that the same component can be used in many systems. The desire to reuse a component leads to some technical constraints such as: good documentation should be available to be able to reuse a component as well as a well organized reuse process and the similar architecture of the components should be provided to ensure the consistency and the efficiency of the system.
2. **Substitutability:** The overall system should work in spite of which component is used. There are some limitations in this area such as: the runtime replacement of the components.
3. **Extensibility:** Extensibility can take one of two shapes either extending components that are part of a system or increase the functionality of individual components.

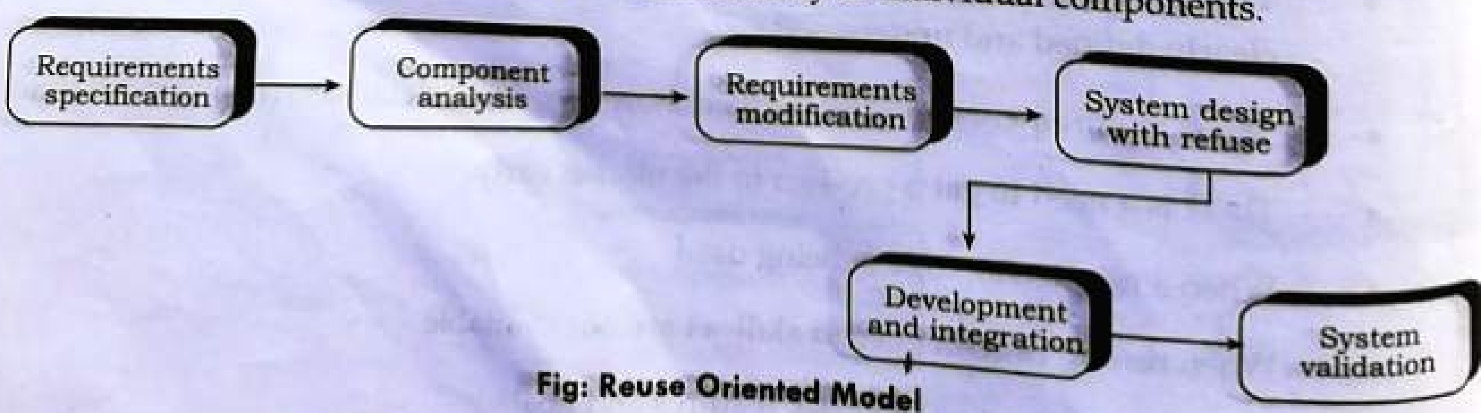
Requirements specification
Component analysis
Requirements modification
System design with refuse
Development and integration
System validation

Fig: Reuse Oriented Model

The stages of component-based software process which are different to other processes are the followings:

1. **Requirement Specification:** In this step, by consulting with the stakeholders of the system, requirements are collected and transform the requirements in to specification document.

2. **Component Analysis:** Based on the requirements specification, a search is made for components that can implement the given specification. Usually, there is no exact match, and the components that may be used only provide some of the functionality required.

3. **Requirements Modification:** During this stage, the requirements are analysed using information about the new components. Requirements are then modified to reflect the services of available components.

4. **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. Some new software may have to be designed if reusable components are not available.

5. **Development and Integration:** In this step, adapt the selected components so that they suit the existing component model or requirement specification. Some components would be possible to directly integrate in to the system, some would be modified through parameterization process, some would need wrapping code for adaptation, etc. Software that cannot be externally procured is developed, and reusable components are integrated to create the new system.

6. **System Validation:** In this step, developed system is tested to ensure that software system does exactly what the customer wants and software is defect free.

Advantages

- Software development cost and risk is reduced.
- Allows faster development and delivery of software.
- In principle, more reliable systems can be developed due to using previously tested components.

Disadvantages

- Compromises in requirements are needed,
- Less control over the system's evolution.

Applicability

- When there is a pool of existing components that could satisfy the requirements of the new product,
- Emerging trend: integration of web services from a range of suppliers.

Prototyping Model

In traditional waterfall model the intermediate changes in user's requirements cannot be accommodated once the system development process begins. So, an alternative method to traditional method has been developed, called prototyping model is based on the assumption that it is difficult to know all the requirement of user in advance. A prototyping is an initial version of software that is used to demonstrate the concept, tryout design options and find out more about problem domain and its possible solutions.

The basic idea in **Prototype model** is that instead of freezing the requirements before a design or coding can proceed, a throwaway prototype is built to understand the requirements. This prototype is developed based on the currently known requirements. By using this prototype, the client can get an "actual feel" of the system, since the interactions with prototype can enable the client to better understand the requirements of the desired system. It is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determining the requirements. In many instances the user only has general view of what is expected from the software product. In such a scenario where there is absence of detailed information regarding the input to the system, the processing needs and the output requirement, the prototyping model may be employed. This model reflects an attempt to increase the flexibility of the development process by allowing the client to interact and experiment with a working representation of the product.

This type of System development method is employed when it is very difficult to obtain exact requirements from the customer (unlike waterfall model, where requirements are clear). While making the model, user keeps giving feedbacks from time to time and based on it, a prototype is made. Completely built sample model is shown to user and based on his feedback; the SRS (System Requirements Specifications) document is prepared. After completion of this, a more accurate SRS is prepared, and now development work can start using Waterfall Model.

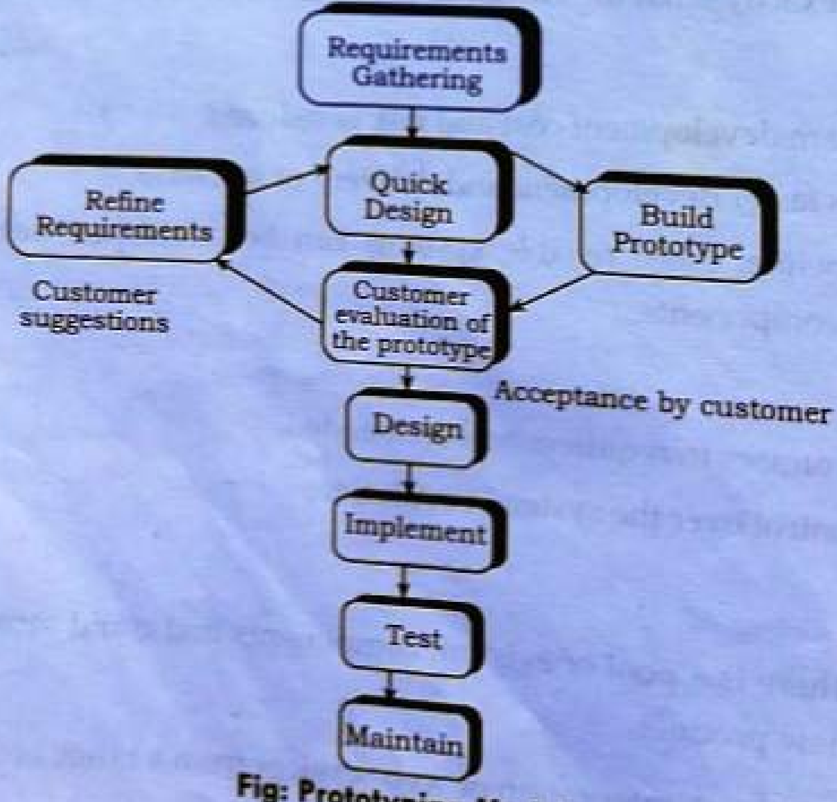

Fig: Prototyping Model

The prototyping model consists of following steps:

1. **Determine the Requirements:** This is the first phase of prototyping model. In this, the system's service, constraints and goals are established by consultation with system users.
2. **Quick Design:** On the basis of known requirement, rough design of system is drawn for discussion and decision.
3. **Build/create Prototype:** In this phase, the information from the design is rolled into prototype. Prototype is the blueprint of the system.
4. **Use and Evaluate Prototyping:** In this phase, the prototype is presented to the user of the system foe use and review. User comments and suggestions are collected and areas of refinements are determined.
5. **Refine the Prototype:** According to suggestions from the user, the system developer revises and enhances the prototype to make it more effective and efficient to meet customer requirements.
6. **Design:** Once the prototype is refined and accepted by the users, actual system is designed to satisfy customer.
7. **Implementation:** During this stage, the software design is realized as a set of programs and program units
8. **Testing:** Once the program modules are ready, each of the program modules is tested independently as per the specifications of the users and debugged.
9. **Operation and Maintenance:** In this stage, the system is installed and put into practical use. Maintenance involves correcting and improving the implementation of system units and enhancing the system's services as new requirements are discovered.

Advantages of Prototyping Model

- When prototype is shown to the user, he/she gets a proper clarity and 'feel' of the functionality of the software and he can suggest changes and modifications.
- Feedbacks from customer are received periodically and the changes don't come as a last minute surprise.
- When client is not confident about the developer's capabilities, he/she asks for a small prototype to be built. Based on this model, he/she judges capabilities of developer.
- Sometimes it helps to demonstrate the concept to prospective investors to get funding for project.
- It reduces risk of failure, as potential risks can be identified early and mitigation steps can be taken
- Iteration between development team and client provides a very good and conductive environment during project.
- Time required to complete the project after getting final the SRS reduces, since the developer has a better idea about how he should approach the project.

Disadvantages of Prototyping Model

- Prototyping is usually done at the cost of the developer. So it should be done using minimal resources.
- Once we get proper requirements from client after showing prototype model, it may be of no use. That is why; sometimes we refer to the prototype as "Throw-away" prototype.
- Too much involvement of client is not always preferred by the developer.
- Too many changes can disturb the rhythm of the development team.

When to use Prototype Model

- Prototype model should be used when the desired system needs to have a lot of interaction with the end users.
- Typically, online systems, web interfaces have a very high amount of interaction with end users, are best suited for Prototype model. It might take a while for a system to be built that allows ease of use and needs minimal training for the end user.
- Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in a useable system. They are excellent for designing good human computer interface systems.

Process Iteration

Changes are usually unavoidable in all large software projects. The system requirements change as organization continuously responds to the changing environment and conditions. Management priorities may change. Due to the quick progress in technologies, designs and implementation will change. This means that the process activities are regularly repeated as the system is reworked in response to change requirements. The following two process models have been designed to support process iteration:

1. Incremental delivery. The software specification, design and implementation are broken down into a series of increments that are each developed in turn.
2. Spiral development. The development of the system spirals outwards from an initial outline through to the final developed system.

Incremental Delivery

The waterfall model of development requires defining the requirements for a system before design begins. On contrary, an evolutionary development allows requirements to change but it leads to software that may be poorly structured and difficult to understand and maintain.

Incremental delivery is an approach that combines the advantages of these two models. In an incremental development process, customers identify the services to be provided by the software system. They decide which subset of the services is 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 priority of service. The highest priority services are delivered first.

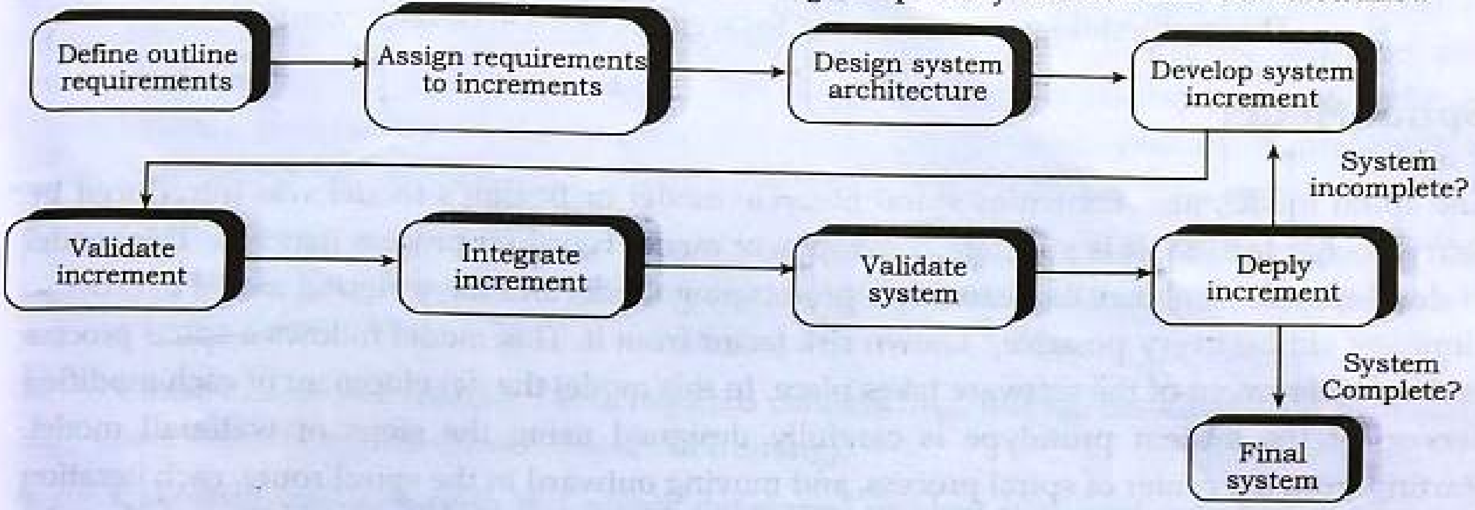

Fig: Incremental delivery

Once the system increments have been identified, the requirements for first increment are defined in detail, and that increment is developed using the best suited software process. As new increments are completed, they are integrated with existing increments and the system functionality improves with each delivered increment.

Advantages of Incremental Delivery Model

- 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.
- Customers can use the early increments as prototypes and gain experience that informs their requirements for later system increments.
- There is a lower risk of overall project failure. Although problems may be encountered in some increments, it is likely that these will be solved in later versions.
- As the highest priority services are delivered first, and later increments are integrated with them, it is unavoidable that the most important system services receive the most testing. This means that software failures are less likely to occur in the most important parts of the system.

Disadvantage of Incremental Delivery Model

- For incremental model, required good designing and well planning.
- The complete cost of this model is higher than waterfall model.

Applicability of Incremental Delivery Model

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

Spiral Model

The spiral model, also known as spiral lifecycle model or Boehm's model was introduced by Barry Boehm in 1988. It is a system development model based on process iteration. This model of development combines the features of prototyping model and the waterfall model in order to eliminate almost every possible/ known risk factor from it. This model follows a spiral process as the development of the software takes place. In this model the development of each modified version of the system prototype is carefully designed using the steps of waterfall model. Starting from the center of spiral process, and moving outward in the spiral route, each iteration helps to build a more complete version of the system in a progressive manner. The spiral model is similar to the incremental model, with more emphasis placed on risk analysis. The spiral model has following major phases:

1. Customer Communication
2. Planning
3. Risk Analysis
4. Engineering
5. Construction and Release
6. Customer Evaluation.

These phases can be depicted in the following figure:

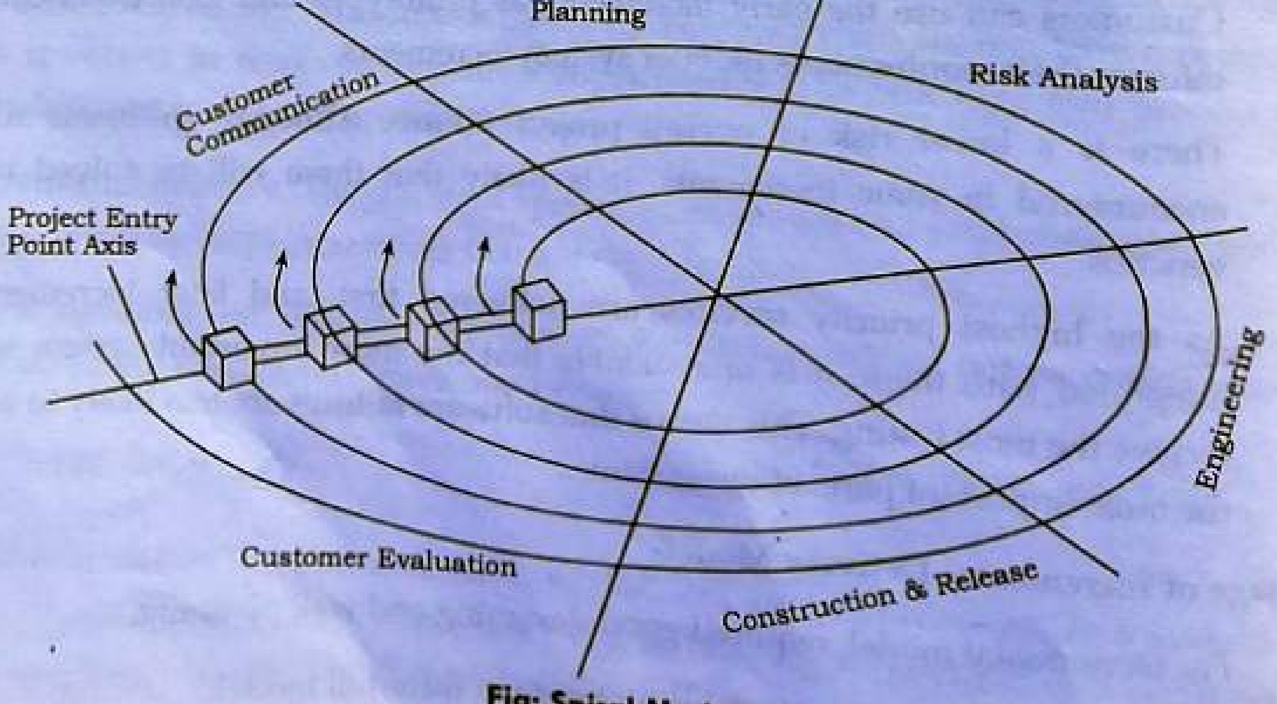

Fig: Spiral Model

1. **Customer Communication:** In this step, effective communication is established between developer and customer to identify the customer requirements.
2. **Planning:** In this phase, the objectives, alternatives and constraints of the projects are determined and are documented. The objectives and other specifications are fixed in order to decide which strategies to follow during project life cycle.
3. **Risk analysis:** This is the most important phase of spiral model. in this phase all possible alternatives, which can help in developing a cost effective project are analyzed and strategies are decided to use them. This phase has been added specially in order to identify and resolve all the possible risk in the project development. A prototype is produced at the end of the risk analysis phase to proceed with available data.
4. **Engineering:** In this phase, the actual development of the project is carried out. The output of this phase is passed through all the phases iteratively in order to obtain improvements in same.
5. **Construction and release:** Tasks required constructing, testing, installing, and providing user support (e.g., documentation,and training).
6. **Customer evaluation:** In this phase, developed product is passed onto the customer in order to receive customer comments and suggestions which can help in identifying and resolving potential in the developed software. This phase is much similar to testing phase. This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

Advantages of Spiral Model

- High amount of risk analysis hence, avoidance of Risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the software life cycle.

Disadvantages of Spiral Model

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.

When to use Spiral model

- When costs and risk evaluation is important
- It is used for medium to high-risk projects
- When users are unsure of their needs
- When requirements are complex
- It is used when significant changes are expected (research and exploration)

Rapid Software Development

RAD model is Rapid Application Development model based on incremental development. In this model the components or functions are developed in parallel as well as rapid development is achieved by using component based construction. This model prioritizes rapid prototyping and quick feedback over long drawn out development and testing cycles. The developments are time boxed, delivered and then assembled into a working prototype. This can quickly give the customer something to see and use and to provide feedback regarding the delivery and their requirements. With rapid application development, developers can make multiple iterations and updates to software rapidly without needing to start a development schedule from scratch each time. It emphasizes the fast and iterative release of prototypes and applications. In today's more competitive market, low-code rapid application development tools allow business and IT teams to effectively collaborate and deliver new applications faster, whether to innovate business practices, differentiate within the market, or streamline costs.

Diagram of RAD-Model

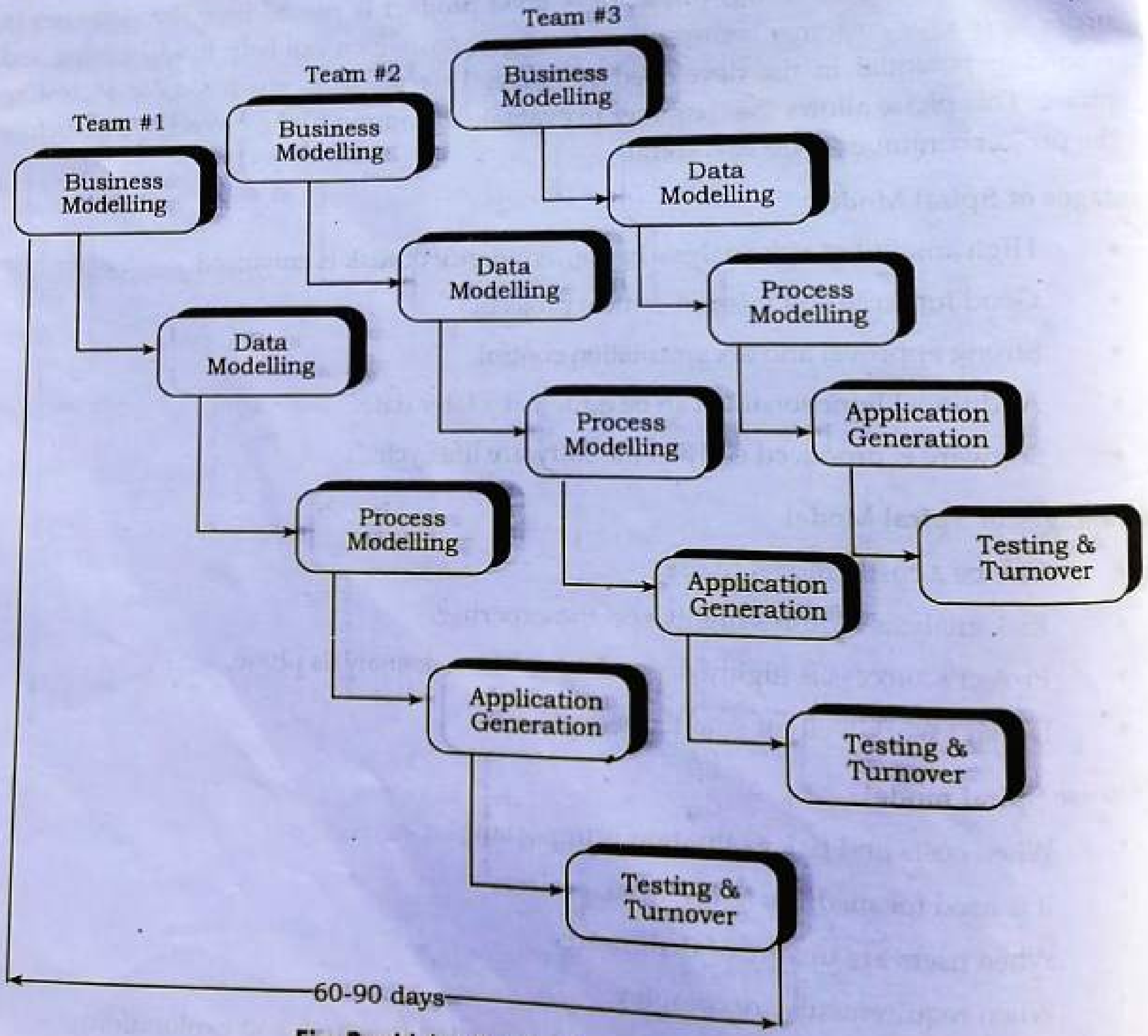

Fig: Rapid Application Development Model

The phases in the rapid application development (RAD) model are:

1 • **Business Modelling:** The information flow is identified between various business functions.

2 • **Data Modelling:** Information gathered from business modelling is used to define data objects that are needed for the business.

3 • **Process Modelling:** Data objects defined in data modelling are converted to achieve the business information flow to achieve some specific business objective. Description are identified and created for CRUD of data objects. Application generation: Automated tools are used to convert process models into code and the actual system.

4 • **Testing and turnover:** Test new components and all the interfaces.

Advantages of the RAD Model

1 • Reduced development time.

2 • Increases reusability of components

2 • Quick initial reviews occur

4 • Encourages customer feedback

- Integration from very beginning solves a lot of integration issues.

Disadvantages of RAD Model

- This model depends on strong team and individual performances for identifying business requirements.
- The system that can be modularized can be built using RAD

1 • It requires highly skilled developers/designers.

2 • High dependency on modelling skills

- Inapplicable to cheaper projects as cost of modelling and automated code generation is very high.

When to use RAD Model

- RAD should be used when there is a need to create a system that can be modularized in 2-3 months of time.
- It should be used if there's high availability of designers for modeling and the budget is high enough to afford their cost along with the cost of automated code generating tools.
- RAD SDLC model should be chosen only if resources with high business knowledge are available and there is a need to produce the system in a short span of time (2-3 months).

Rational Unified Process (RUP)

Rational Unified Process (RUP) is an iterative software development process framework created by the rational software corporation, a division of IBM. It is an object oriented and web enabled program development methodology. It has been derived from work on UML and the associated unified software development process. It brings together elements from all the generic process models, support iteration and illustrates good practice in specification and design. RUP is agile software development method, in which the life cycle of a project, or the development of software, is divided into four phases. It is an iterative and agile focused because all of the process's core activities repeat throughout the project. The process is agile because various components can be adjusted, and phases of the cycle can be repeated until the software meets requirements and objectives. RUP is normally described from three perspectives. A static perspective that show process activities that are enacted. A dynamic perspective that shows the phases of model over time and a practice perspective that suggests good practice to be used during the process.

Iteration in RUP is supported in to two ways: each phase may be enacted in an iterative way with the result developed incrementally and whole set of phases may also be enacted incrementally by the looping from transition to inception. The major phases of RUP are as follows:

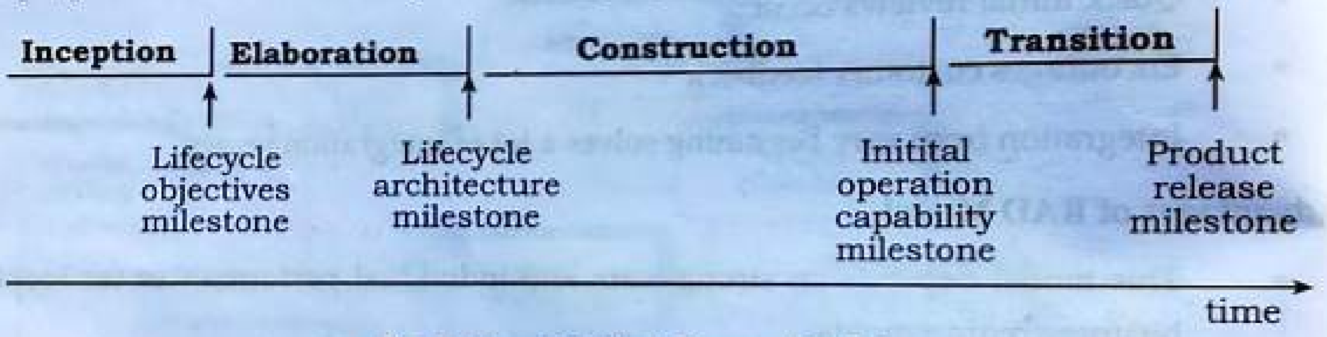
Inception
Elaboration
Construction
Transition
Lifecycle objectives milestone
Lifecycle architecture milestone
Initital operation capability milestone
Product release milestone
time

Fig: Rational Unified Process Model

Inception Phase

During the inception phase, the basic idea and structure of the project is determined. The team will sit down and determine if the project is worth pursuing at all. It establishes business cases for the system, all the external entities people and system that will interact with the system are identified and define these interactions. Based on the proposed purpose of the system, the estimated costs (monetary and time), and what resources will be required to complete the project once the green light is given. The conclusion of the inception phase is the Lifecycle objectives Milestone, which consists of the following evaluation criteria:

- Stakeholder concurrence on scope definition and cost/schedule estimates.
- Requirements understanding as evidenced by the fidelity of the primary use cases.
- Credibility of the cost/schedule estimates, priorities, risks, and development process.
- Depth and breadth of any architectural prototype that was developed.
- Actual expenditures versus planned expenditures.

Elaboration Phase

This phase develop an understanding of the problem domain, establish an architectural framework for the system develop the project plan and identify key risks. The purpose of the elaboration phase is to analyze the requirements and necessary architecture of the system. The success of this phase is particularly critical, as the final milestone of this phase signifies the transition of the project from low-risk to high-risk, since the actual development and coding will take place in the following phase.

The Lifecycle Architecture Milestone signifies the end of the elaboration phase, and is evaluated using these criteria:

- Is the vision of the product stable?
- Is the architecture stable?
- Does the executable demonstration show that the major risk elements have been addressed and credibly resolved?
- Is the plan for the construction phase sufficiently detailed and accurate? Is it backed up with a credible basis of estimates?
- Do all stakeholders agree that the current vision can be achieved if the current plan is executed to develop the complete system, in the context of the current architecture?
- Is the actual resource expenditure versus planned expenditure acceptable?

Construction Phase

The construction phase is essentially concerned when the design, coding and testing of all application features will take place. This period is also where integrations with other services or existing software should occur. The end of the construction phase is measured by the completion of the Initial Operational Capability Milestone, which is based on these criteria:

- Is this product release stable and mature enough to be deployed in the user community?
- Are all stakeholders ready for the transition into the user community?
- Are the actual resource expenditures versus planned expenditures still acceptable?

Transition Phase

This phase is concerned with moving the system from development community to user community and making it work in real environment. However, the transition phase is more than just the process of deployment; it must also handle all post-release support, bug fixes, patches, and so forth. The Product Release Milestone signals the end of the transition phase, and is based on a few simple questions:

- Is the user satisfied?
- Are the actual resources expenditures versus planned expenditures stil acceptable?

Advantages of Rational Unified Process

- Allows for the adaptive capability to deal with changing requirements throughou the development life cycle, whether they be from customers or from within th project itself.
- Emphasizes the need (and proper implementation of) accurate documentation.
- Diffuses potential integration headaches by forcing integration to occur throughout development, specifically within the construction phase where all other coding and development is taking place.

Disadvantages of Rational Unified Process

- Heavily relies on proficient and expert team members, since assignment of activities to individual workers should produce tangible, pre-planned results in the form of artifacts.
- Given the emphasis on integration throughout the development process, this can also be detrimental during testing or other phases, where integrations are conflicting and getting in the way of other, more fundamental activities.
- It is a fairly complicated model. Given the assortment of the components involved including best practices, phases, building blocks, milestone criteria, iterations and workflows, often proper implementation and use of the Rational Unified Process can be challenging for many organizations, particularly for smaller teams or projects.

COMPUTER AIDED SOFTWARE ENGINEERING (CASE)

A CASE (Computer Aided Software Engineering) tool is a standard term used to indicate an form of automated support for software engineering. CASE tools are set of software applicatio programs, which are used to automate SDLC activities. These are the tool used to automat some activity associated with software development. CASE is used to ensure a high-quality an defect-free software. It ensures a check-pointed and disciplined approach and helps designers developers, testers, managers and others to see the project milestones during development CASE tools are used by software project managers, analysts and engineers to develop softwar system. Use of CASE tools accelerates the development of project to produce desired result an helps to uncover flaws before moving ahead with next stage in software development.

There are number of CASE tools available to simplify various stages of Software Development Life Cycle such as Analysis tools, Design tools, Project management tools, Database Management tools, Documentation tools are to name a few.

Reasons for Using CASE Tools

- Savings in resources required for software development – with less.
- Quick development phase
- Reduction of generation of errors.
- Easier recognition of bugs during development.
- Savings in maintenance resources required.
- To increase productivity
- To help produce enhanced quality software at lower cost

Benefits of Using CASE Tools

Several benefits increase from the use of a CASE environment or even isolated CASE tools. Some of those pros are as follows:

- The major benefit of a CASE environment is cost saving through all development stages.
- Use of CASE tools leads to considerable improvement to quality.
- Different phases of Software Development and the chances of human error are significantly reduced.
- CASE tools help to produce high quality and consistent documents
- CASE tools take out most of the work in a software engineer's work.

Classification of CASE Tools

- **Upper Case Tools**: Upper CASE tools describe the tools that automate or support upper or earliest phases of system development. These tools are considered as front end tools and assist the developer during planning, requirement analysis and design stages of SDLC.
- **Lower Case Tools -:** Lower CASE tools are the tools that automate or support lower or later phases of system development. These tools are considered as back end tools and assist the developer with implementation, testing and maintenance activities. These tools aim to increase the reliability, adaptability and productivity of the code as well as support programming and integration tasks.
- **Integrated Case Tools:** Integrated CASE tools are the tools that automate or support Upper as well as lower phases of system development. These CASE tools are helpful in all the stages of SDLC, from Requirement gathering to Testing and documentation. They are complex and expensive too.

Case Tools Types

CASE tools are used in different phases of system development life cycle for different purpose, which is explain as below

Analysis Tools

These tools help to gather requirements, automatically check for any inconsistency, inaccuracy in the diagrams, data redundancies or erroneous omissions. For example, Accept 360, Accompa, Case Complete for requirement analysis, Visible Analyst for total analysis.

Design Tools

These tools help software designers to design the block structure of the software, which may further be broken down in smaller modules using refinement techniques. These tools provides detailing of each module and interconnections among modules. For example, Animated Software Design.

Programming Tools

These tools consist of programming environments like IDE (Integrated Development Environment), in-built modules library and simulation tools. These tools provide comprehensive aid in building software product and include features for simulation and testing. For example, Cscope to search code in C, Eclipse.

Prototyping Tools

Software prototype is simulated version of the intended software product. Prototype provides initial look and feel of the product and simulates few aspect of actual product.

Prototyping CASE tools essentially come with graphical libraries. They can create hardware independent user interfaces and design. These tools help us to build rapid prototypes based on existing information. In addition, they provide simulation of software prototype. For example, Serena prototype composer, Mockup Builder.

Maintenance Tools

Software maintenance includes modifications in the software product after it is delivered. Automatic logging and error reporting techniques, automatic error ticket generation and root cause Analysis are few CASE tools, which help software organization in maintenance phase of SDLC. For example, Bugzilla for defect tracking, HP Quality Center.

Diagram tools

These tools are used to represent system components, data and control flow among various software components and system structure in a graphical form. For example, Flow Chart Maker tool for creating state-of-the-art flowcharts.

Process Modeling Tools

Process modeling is method to create software process model, which is used to develop the software. Process modeling tools help the managers to choose a process model or modify it as per the requirement of software product. For example, EPF Composer

Project Management Tools

These tools are used for project planning, cost and effort estimation, project scheduling and resource planning. Managers have to strictly comply project execution with every mentioned step in software project management. Project management tools help in storing and sharing project information in real-time throughout the organization. For example, Creative Pro Office, Trac Project, Basecamp.

Documentation Tools

Documentation in a software project starts prior to the software process, goes throughout all phases of SDLC and after the completion of the project. Documentation tools generate documents for technical users and end users. Technical users are mostly in-house professionals of the development team who refer to system manual, reference manual, training manual, installation manuals etc. The end user documents describe the functioning and how-to of the system such as user manual. For example, Doxygen, Adobe RoboHelp.

Configuration Management Tools

An instance of software is released under one version. Configuration Management tools deal with –

- Version and revision management
- Baseline configuration management
- Change control management

CASE tools help in this by automatic tracking, version management and release management. For example, Fossil, Git, Accu REV.

Change Control Tools

These tools are considered as a part of configuration management tools. They deal with changes made to the software after its baseline is fixed or when the software is first released. CASE tools automate change tracking, file management, code management and more. It also helps in enforcing change policy of the organization.

Web Development Tools

These tools assist in designing web pages with all allied elements like forms, text, script, graphic and so on. Web tools also provide live preview of what is being developed and how will it look after completion. For example, Fontello, Adobe Edge Inspect, Foundation 3, Brackets.

Quality Assurance Tools

Quality assurance in a software organization is monitoring the engineering process and methods adopted to develop the software product in order to ensure conformance of quality as per organization standards. QA tools consist of configuration and change control tools and software testing tools. For example, SoapTest, AppsWatch, JMeter etc.

EXERCISE

1. What do you mean by software process model? Explain the fundamental software process activities.
2. Explain waterfall model of software development along with advantages, disadvantages and applicability.
3. What is incremental development model? Explain with block diagram. Why it is difficult to maintain software developed using incremental model?
4. What is spiral model? How does spiral model accommodate best features of waterfall model and prototyping model?
5. Explain reuse oriented (CBSE) model with its advantages and disadvantages.
6. Define RUP along with its different phases. What are advantages of static and dynamic views in RUP?

❑❑❑