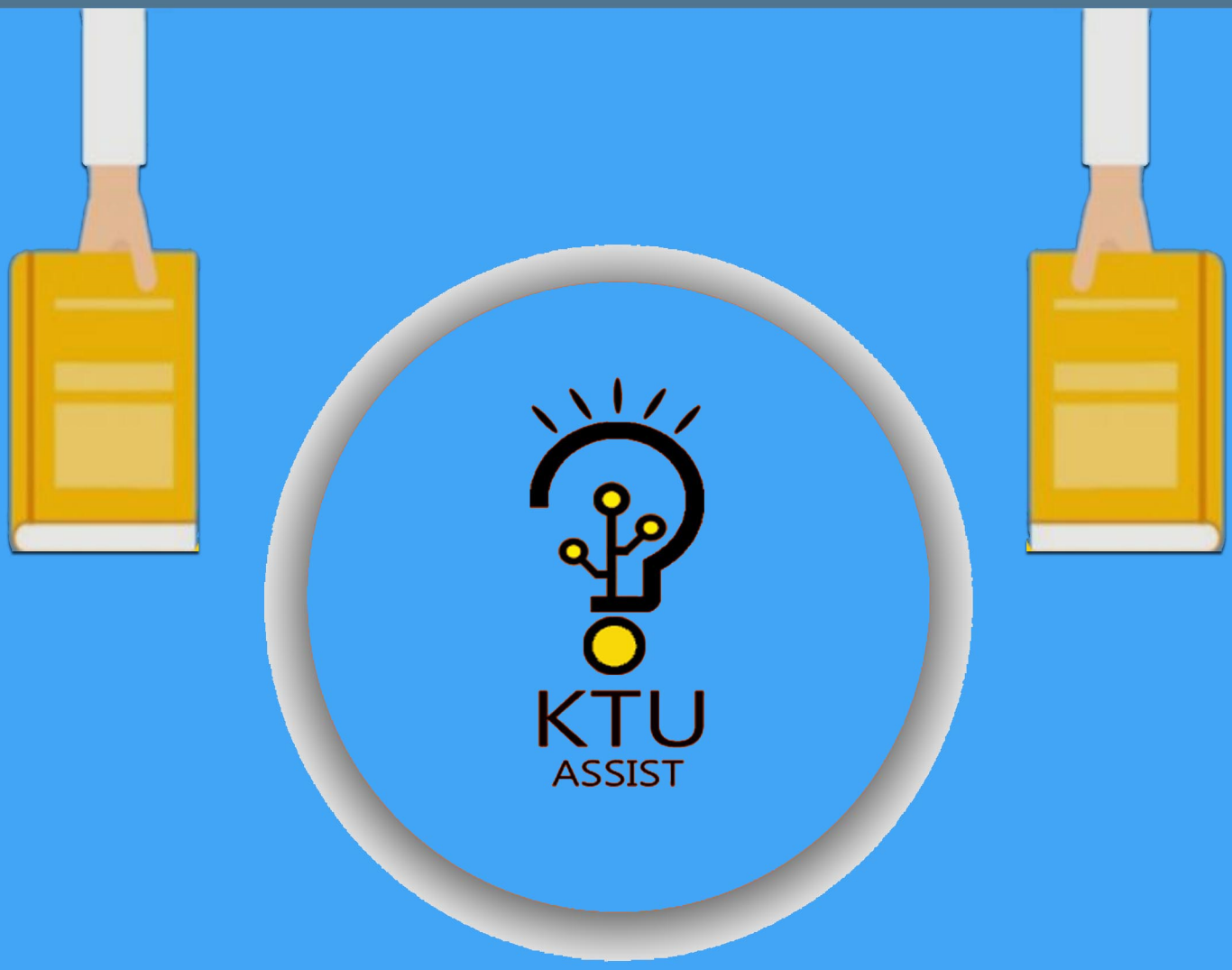
APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

STUDY MATERIALS



Module I

Introduction to software engineering- scope of software engineering – historical aspects, economic aspects, maintenance aspects, specification and design aspects, team programming aspects. Software engineering a layered technology – processes, methods and tools. Software process models – prototyping models, incremental models, spiral model, waterfall model.

Q1. Define software Engineering?

Software

- Computer programs and associated documentation
  - requirements, design models and user manuals.
- Software products may be
  - Generic - developed to be sold to a range of different customers
- e.g. PC software such as Excel or Word.
  - custom - developed for a single customer according to their specification.
- New software can be created by developing new programs, configuring generic software systems or reusing existing software.

What is computer Science

- Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
- Computer science theories are still insufficient to act as a complete underpinning for software engineering.

System engineering vs Software Engineering

- System engineering - computer-based systems development including hardware, software and process engineering.
- Software engineering is part of this process - developing the software infrastructure, control, applications and databases in the system.
- System engineers are involved in system specification, architectural design, integration and deployment.

Software Process

- A set of activities for the development or evolution of software.
- Generic activities in all software processes are:
  - Specification - what the system should do and its development constraints
  - Development - production of the software system
  - Validation - checking that the software is what the customer wants
  - Evolution - changing the software in response to changing demands.

1.1 Introduction to Software Engineering

- Software Engineering is an engineering discipline which is concerned with all aspects of software production from the early stages of system requirements through to maintaining the system after is has gone into use.
- Computer Science is concerned with the theories and methods which underlie computers and software systems.
- Software Engineering is concerned with the practical problem of producing software.
- "Engineering discipline" – Engineers make things work. They apply theories, methods and tools that are appropriate but use them selectively and always try to discover solutions to problems even when

there are not applicable theories and methods to support them. Engineers have to work to organizational and financial constraints.

- "All aspects of" - Software engineering is not just concerned with the technical process of software development but also with activities such as software project management and the development of tools, methods and theories to support software product

Q2. Explain the various scope of software engineering?

1.2 Scope of software engineering

The scope of software engineering is extremely broad,five aspects are involved:

- Historical Aspects
- Economic Aspects
- Maintenance Aspects
- Requirements, Analysis, and Design Aspects
- Team Development Aspects

These five aspects can be categorized in the fields of Mathematics, Computer Science Economics, Management, and Psychology

1.3 Historical Aspects

- In the belief that software could be engineered on the same footing as traditional engineering disciplines a NATO study group coined the term "Software Engineering" in 1967. This was endorsed by the NATO Software Engineering Conference in 1968.
- Software engineering cannot be considered as engineered since an unacceptably large proportion of software products still are being
  - Delivered late
  - Over budget
    - With residual faults.
  - Solution is that a software engineer has to acquire a broad range of skills, both technical managerial. These skills have to be applied to Programming and every step of software production, from requirements to maintenance.

1.4 Economic Aspects

- Applying economic principles to software engineering requires the client to choose techniques that reduce long-term costs in terms of the economic sense.
- The cost of introducing new technology into an organization may involve training cost , a steep learning curve , unable to do productive work when attending the class etc.

1.5 Maintenance Aspects

- Classical View of Maintenance: Development-then-maintenance model. However, this model is unrealistic due to: – During the development, the client's requirements may change. This leads to the changes in the specification and design. Developers try to reuse parts of existing software products in the software product to be constructed.
- Modern view of Maintenance: It is the process that occurs when "software undergoes modifications to code and associated documentation due to a problem or the need for improvement or adaptation". That is, maintenance occurs whenever a fault is fixed or the requirements change, irrespective of whether this takes place before or after installation of the product.
- Classical Postdelivery Maintenance: All changes to the product once the product has been delivered and installed on the client's computer and passes its acceptance test.
- Modern Maintenance (or just maintenance): Corrective, perfective, or adaptive activities performed at any time. Classical post delivery maintenance is a subset of modern maintenance
- The importance of Post delivery Maintenance: – A software product is a model of the real world, and the real world is perpetually changing. As a consequence, software has to be maintained constantly for it to remain an accurate reflection of the real world. A major aspect of software engineering consists of techniques, tools, and practices that lead to a reduction in post delivery maintenance cost.

1.6. Requirements, Analysis, and Design Aspects

- The earlier we correct a fault, the better. That is, the cost of correcting a fault increases steeply since it is directly related to what has to be done to correct a fault.
- If the mistake is made while eliciting the requirements, the resulting fault will probably also appear in the specifications, the design, and the code. We have to edit the code, recompile and relink the code, and test.

- It is crucial to check that making the change has not created a new problem elsewhere in the product. All the relevant documentation, including manuals, needs to be updated. The corrected product must be delivered and reinstalled.

1.7 Team Development Aspects

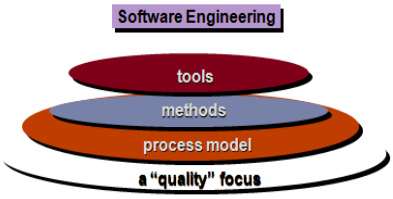
- Team development leads to interface problems among code components and communication problems among team members.
- Unless the team is properly organized, an inordinate amount of time can be wasted in conferences between team members.
- It also includes human aspects, such as team organization, economic aspects, and legal aspects, such as copyright law.

Q3. Explain software engineering as a layered technology?

1.8 Software Engineering a Layered Technology

Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines

A Layered Technology



a. Quality Focus

Continuous process improvement. Bedrock that supports software engineering.

b. Process

It is the foundation of software engineering. Defines a framework that must be established for

- ► Effective delivery of SE technology
- ► Management control of software project
- ► Context of technical methods applied
- ► Work products
- ► Milestones, Quality ensured
- ► Proper change management

c. Methods

- Provide technical how-to's for building software.
- Encompass a broad array of tasks that include requirements analysis, design modelling , program construction, testing, and support.
- Rely on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques

d. Tools

- Provide automated or semi-automated support for the process and the methods.
- CASE: computer-aided software engineering is a system for the support of software development.
  - Combines SW, HW, and a SE database (a repository containing important information about analysis, design, program construction, and testing)

Q4. Explain the various software process models?

1.9 Software Process Model

- A software engineer must incorporate a development strategy that encompasses
  - the process methods, and tools layers.
- This strategy is often referred to as a *process model* or a *software engineering paradigm*.
- A process model is chosen based on
  - the nature of project and application,

- the methods and tools to be used,
- and the controls and deliverables that are required

Q5. Explain about waterfall model

1.10 classical waterfall model

The classical waterfall model is intuitively the most obvious way to develop software. Though the classical waterfall model is elegant and intuitively obvious, it is not a practical model in the sense that it can not be used in actual software development projects. Thus, this model can be considered to be a *theoretical way of developing software*. But all other life cycle models are essentially derived from the classical waterfall model. So, in order to be able to appreciate other life cycle models it is necessary to learn the classical waterfall model.

Classical waterfall model divides the life cycle into the following phases as shown in fig

- Feasibility Study
- Requirements Analysis and Specification
- Design
- Coding and Unit Testing
- Integration and System Testing
- Maintenance

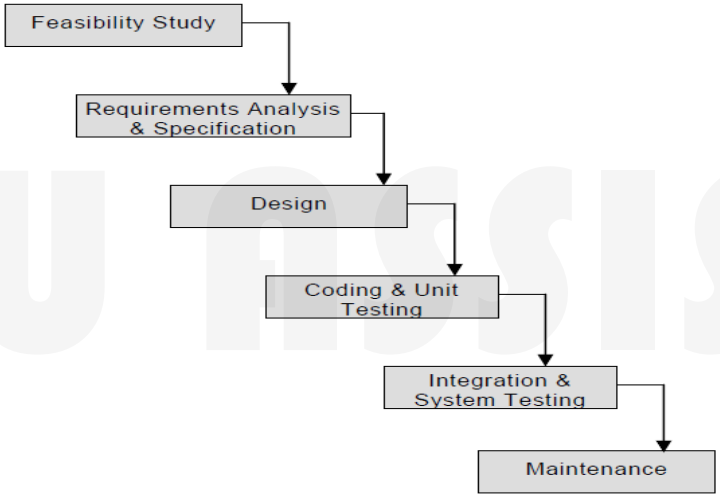


Fig Classical Waterfall Model

Activities in each phase of the life cycle

• **Activities undertaken during feasibility study: -**

The main aim of feasibility study is to determine whether it would be financially and technically feasible to develop the product.

- At first project managers or team leaders try to have a rough understanding of what is required to be done by visiting the client side. They study different input data to the system and output data to be produced by the system. They study what kind of processing is needed to be done on these data and they look at the various constraints on the behavior of the system.

- After they have an overall understanding of the problem they investigate the different solutions that are possible. Then they examine each of the solutions in

terms of what kind of resources required, what would be the cost of development and what would be the development time for each solution.

- Based on this analysis they pick the best solution and determine whether the solution is feasible financially and technically. They check whether the customer budget would meet the cost of the product and whether they have sufficient technical expertise in the area of development.

The following is an example of a feasibility study undertaken by an organization. It is intended to give you a feel of the activities and issues involved in the feasibility study phase of a typical software project.

handling communication with the mine sites. He arrived at a cost to develop from the analysis. He found that the solution involving maintenance of local databases at the mine sites and periodic updating of a central database was financially and technically feasible. The project manager discussed his solution with the GMC management and found that the solution was acceptable to them as well.

• **Activities undertaken during requirements analysis and specification: -**

The aim of the requirements analysis and specification phase is to understand the exact requirements of the customer and to document them properly. This phase consists of two distinct activities, namely

Requirements gathering and analysis, and

Requirements specification

The goal of the requirements gathering activity is to collect all relevant information from the customer regarding the product to be developed. This is done to clearly understand the customer requirements so that incompleteness and inconsistencies are removed.

The requirements analysis activity is begun by collecting all relevant data regarding the product to be developed from the users of the product and from the customer through interviews and discussions. For example, to perform the requirements analysis of a business accounting software required by an organization, the analyst might interview all the accountants of the organization to ascertain their requirements. The data collected from such a group of users usually contain several contradictions and ambiguities, since each user typically has only a partial and incomplete view of the system. Therefore it is necessary to identify all ambiguities and contradictions in the requirements and resolve them through further discussions with the customer. After all ambiguities, inconsistencies, and incompleteness have been resolved and all the requirements properly understood, the requirements specification activity can start. During this activity, the user requirements are systematically organized into a Software Requirements Specification (SRS) document.

The customer requirements identified during the requirements gathering and analysis activity are organized into a SRS document. The important components of this document are functional requirements, the nonfunctional requirements, and the goals of implementation.

• **Activities undertaken during design: -**

The goal of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the software architecture is derived from the SRS document. Two distinctly different approaches are available: the traditional design approach and the object-oriented design approach.

Traditional design approach

Traditional design consists of two different activities; first a structured analysis of the requirements specification is carried out where the detailed structure of the problem is examined. This is followed by a structured design activity. During structured design, the results of structured analysis are transformed into the software design.

Object-oriented design approach

In this technique, various objects that occur in the problem domain and the solution domain are first identified, and the different relationships that exist among these objects are identified. The object structure is further refined to obtain the detailed design.

• **Activities undertaken during coding and unit testing:-**

The purpose of the coding and unit testing phase (sometimes called the implementation phase) of software development is to translate the software design into source code. Each component of the design is implemented as a program module. The end-product of this phase is a set of program modules that have been individually tested.

During this phase, each module is unit tested to determine the correct working of all the individual modules. It involves testing each module in isolation as this is the most efficient way to debug the errors identified at this stage.

• **Activities undertaken during integration and system testing: -**

Integration of different modules is undertaken once they have been coded and unit tested. During the integration and system testing phase, the modules are integrated in a planned manner. The different modules making up a software product are almost never integrated in one shot. Integration is normally carried out incrementally over a number of steps. During each integration step, the partially integrated system is tested and a set of previously planned modules are added to it. Finally, when all the modules have been successfully integrated and tested, system testing is carried out. The goal of system testing is to ensure that the developed system conforms to its requirements laid out in the SRS document. System testing usually consists of three different kinds of testing activities:

- α – testing: It is the system testing performed by the development team.
- β – testing: It is the system testing performed by a friendly set of customers.
- acceptance testing: It is the system testing performed by the customer himself after the product delivery to determine whether to accept or reject the delivered product.

System testing is normally carried out in a planned manner according to the system test plan document. The system test plan identifies all testing-related activities that must be performed, specifies the schedule of testing, and allocates resources. It also lists all the test cases and the expected outputs for each test case.

• **Activities undertaken during maintenance: -**

Maintenance of a typical software product requires much more than the effort necessary to develop the product itself. Many studies carried out in the past confirm this and indicate that the relative effort of development of a typical software product to its maintenance effort is roughly in the 40:60 ratio. Maintenance involves performing any one or more of the following three kinds of activities:

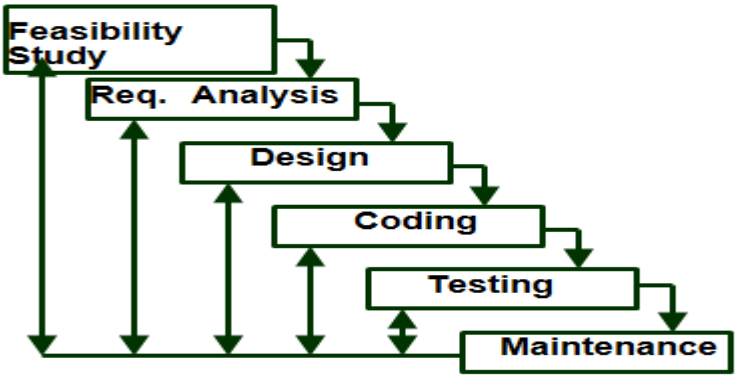
- Correcting errors that were not discovered during the product development phase. This is called corrective maintenance.
- Improving the implementation of the system, and enhancing the functionalities of the system according to the customer's requirements. This is called perfective maintenance.
- Porting the software to work in a new environment. For example, porting may be required to get the software to work on a new computer platform or with a new operating system. This is called adaptive maintenance.

Shortcomings of the classical waterfall model

- The classical waterfall model is an idealistic one since it assumes that no development error is ever committed by the engineers during any of the life cycle phases. However, in practical development environments, the engineers do commit a large number of errors in almost every phase of the life cycle. The source of the defects can be many: oversight, wrong assumptions, use of inappropriate technology, communication gap among the project engineers, etc.
- . These defects usually get detected much later in the life cycle. For example, a design defect might go unnoticed till we reach the coding or testing phase. Once a defect is detected, the engineers need to go back to the phase where the defect had occurred and redo some of the work done during that phase and the subsequent phases to correct the defect and its effect on the later phases.
- Phase-entry and phase-exit criteria of each phase
  - At the starting of the feasibility study, project managers or team leaders try to understand what is the actual problem by visiting the client side. At the end of that phase they pick the best solution and determine whether the solution is feasible financially and technically.
  - At the starting of requirements analysis and specification phase the required data is collected. After that requirement specification is carried out. Finally, SRS document is produced.
  - At the starting of design phase, context diagram and different levels of DFDs are produced according to the SRS document. At the end of this phase module structure (structure chart) is produced.
  - During the coding phase each module (independently compilation unit) of the design is coded. Then each module is tested independently as a stand-alone unit and debugged separately. After this each module is documented individually. The end product of the implementation phase is a set of program modules that have been tested individually but not tested together.
  - After the implementation phase, different modules which have been tested individually are integrated in a planned manner. After all the modules have been successfully integrated and tested, system testing is carried out.
  - Software maintenance denotes any changes made to a software product after it has been delivered to the customer. Maintenance is inevitable for almost any kind of product. However, most products need maintenance due to the wear and tear caused by use.

Iterative Waterfall Model

- Classical waterfall model is idealistic-assumes that no defect is introduced during any development activity.In practice defects do get introduced in almost every phase of the life cycle. Defects usually get detected much later in the life cycle:
  - For example, a design defect might go unnoticed till the coding or testing phase.
- Once a defect is detected:
  - we need to go back to the phase where it was introduced
  - redo some of the work done during that and all subsequent phases.
- **Therefore we need feedback paths in the classical waterfall model.**



- **Errors should be detected**
  - in the same phase in which they are introduced.
- **For example:**
  - **if a design problem is detected in the design phase itself,**
    - the problem can be taken care of much more easily
    - than say if it is identified at the end of the integration and system testing phase.

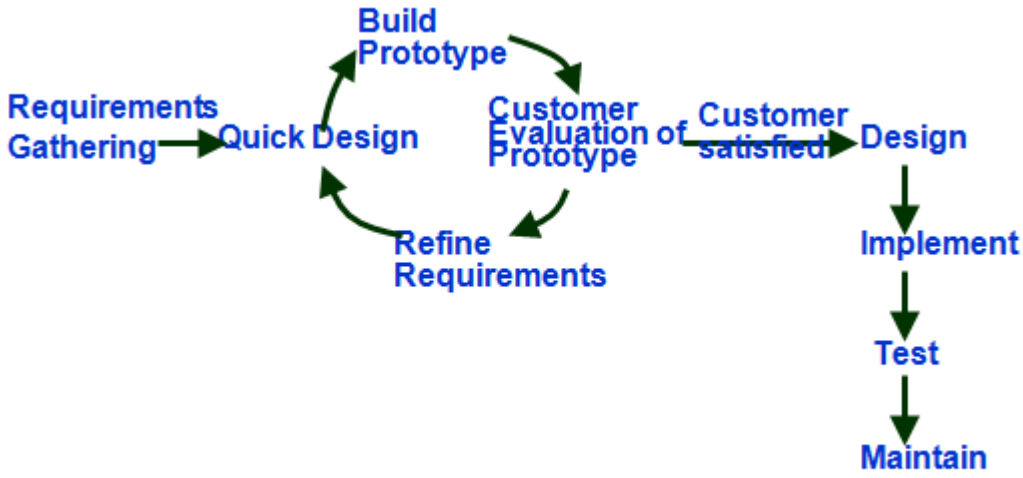
Q6. Summarize prototyping model with a neat diagram?

Prototyping Model

A prototype is a toy implementation of the system. A prototype usually exhibits limited functional capabilities, low reliability, and inefficient performance compared to the actual software. A prototype is usually built using several shortcuts. The shortcuts might involve using inefficient, inaccurate, or dummy functions. The shortcut implementation of a function, for example, may produce the desired results by using a table look-up instead of performing the actual computations. A prototype usually turns out to be a very crude version of the actual system.

- **Start with approximate requirements.**
- **Carry out a quick design.**
- **The developed prototype is submitted to the customer for his evaluation:**
  - Based on the user feedback, requirements are refined.
  - This cycle continues until the user approves the prototype.

The actual system is developed using the classical waterfall approach



Need for a prototype in software development

There are several uses of a prototype. An important purpose is to illustrate the input data formats, messages, reports, and the interactive dialogues to the customer. This is a valuable mechanism for gaining better understanding of the customer's needs:

• how the screens might look like

• how the user interface would behave

• how the system would produce outputs

- This is something similar to what the architectural designers of a building do; they show a prototype of the building to their customer. The customer can evaluate whether he likes it or not and the changes that he would need in the actual product. A similar thing happens in the case of a software product and its prototyping model.

- Another reason for developing a prototype is that it is impossible to get the perfect product in the first attempt. Many researchers and engineers advocate that if you want to develop a good product you must plan to throw away the first version. The experience gained in developing the prototype can be used to develop the final product.

- A prototyping model can be used when technical solutions are unclear to the development team. A developed prototype can help engineers to critically examine the technical issues associated with the product development.

- Often, major design decisions depend on issues like the response time of a hardware controller, or the efficiency of a sorting algorithm, etc. In such circumstances, a prototype may be the best or the only way to resolve the technical issues.

Examples for prototype model

A prototype of the actual product is preferred in situations such as:

• user requirements are not complete
• technical issues are not clear

Let's see an example for each of the above category.

Example 1: User requirements are not complete

In any application software like billing in a retail shop, accounting in a firm, etc the users of the software are not clear about the different functionalities required. Once they are provided with the prototype implementation, they can try to use it and find out the missing functionalities.

Example 2: Technical issues are not clear

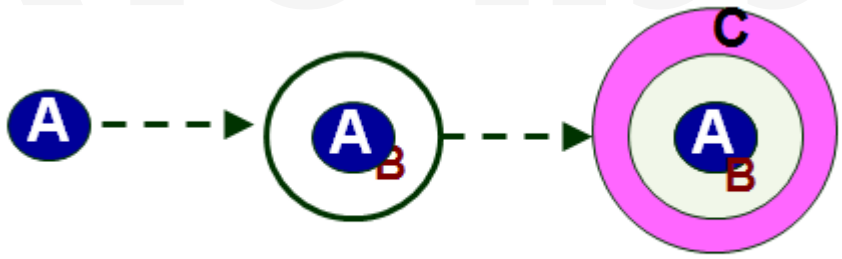
Suppose a project involves writing a compiler and the development team has never written a compiler.

In such a case, the team can consider a simple language, try to build a compiler in order to check the issues that arise in the process and resolve them. After successfully building a small compiler (prototype), they would extend it to one that supports a complete language.

Q7. Describe evolutionary or incremental model?

Evolutionary Model(Incremental Model)

- Evolutionary model (aka successive versions or incremental model):
  - The system is broken down into several modules which can be incrementally implemented and delivered.
- First develop the core modules of the system.
- The initial product skeleton is refined into increasing levels of capability:by adding new functionalities in successive versions
- Successive version of the product:
- functioning systems capable of performing some useful work.
- A new release may include new functionality:
- also existing functionality in the current release might have been enhanced



Advantages of Evolutionary Model

- Users get a chance to experiment with a partially developed system:
  - much before the full working version is released,
- Helps finding exact user requirements:
  - much before fully working system is developed.
- Core modules get tested thoroughly:
  - reduces chances of errors in final product.

Disadvantages of Evolutionary Model

- Often, difficult to subdivide problems into functional units:which can be incrementally implemented and delivered.

- evolutionary model is useful for very large problems, where it is easier to find modules for incremental implementation.

Q8. Explain Spiral model?

Spiral model

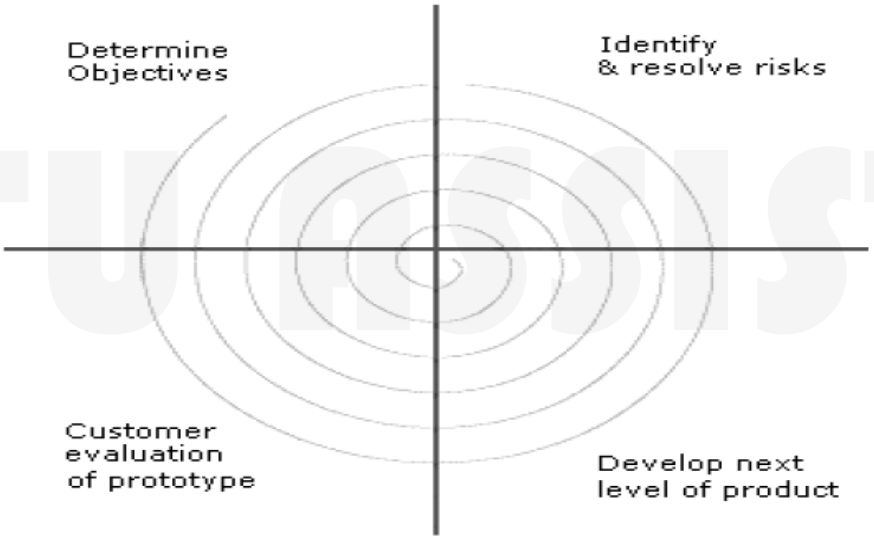
The Spiral model of software development is shown in fig. 2.2. The diagrammatic representation of this model appears like a spiral with many loops. The exact number of loops in the spiral is not fixed. Each loop of the spiral represents a phase of the software process. For example, the innermost loop might be concerned with feasibility study. The next loop with requirements specification, the next one with design, and so on. Each phase in this model is split into four sectors (or quadrants) as shown in fig. below The following activities are carried out during each phase of a spiral model.

- First quadrant (Objective Setting)

- During the first quadrant, it is needed to identify the objectives of the phase.
- Examine the risks associated with these objectives.

- Second Quadrant (Risk Assessment and Reduction)

- A detailed analysis is carried out for each identified project risk.
- Steps are taken to reduce the risks. For example, if there is a risk that the requirements are inappropriate, a prototype system may be developed.



Spiral Model

- Third Quadrant (Development and Validation)

- Develop and validate the next level of the product after resolving the identified risks.

- Fourth Quadrant (Review and Planning)

- Review the results achieved so far with the customer and plan the next iteration around the spiral.
- Progressively more complete version of the software gets built with each iteration around the spiral.

Circumstances to use spiral model

- The spiral model is called a meta model since it encompasses all other life cycle models. Risk handling is inherently built into this model. The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks. However,

this model is much more complex than the other models – this is probably a factor deterring its use in ordinary projects.

Q9. Com[pare various process models?

Comparison of different life-cycle models

- The classical waterfall model can be considered as the basic model and all other life cycle models as embellishments of this model. However, the classical waterfall model can not be used in practical development projects, since this model supports no mechanism to handle the errors committed during any of the phases.
- This problem is overcome in the iterative waterfall model. The iterative waterfall model is probably the most widely used software development model evolved so far. This model is simple to understand and use. However, this model is suitable only for well-understood problems; it is not suitable for very large projects and for projects that are subject to many risks.
- The prototyping model is suitable for projects for which either the user requirements or the underlying technical aspects are not well understood. This model is especially popular for development of the user-interface part of the projects.
- The evolutionary approach is suitable for large problems which can be decomposed into a set of modules for incremental development and delivery. This model is also widely used for object-oriented development projects. Of course, this model can only be used if the incremental delivery of the system is acceptable to the customer.
- The spiral model is called a meta model since it encompasses all other life cycle models. Risk handling is inherently built into this model. The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks. However, this model is much more complex than the other models – this is probably a factor deterring its use in ordinary projects.
- The different software life cycle models can be compared from the viewpoint of the customer. Initially, customer confidence in the development team is usually high irrespective of the development model followed. During the lengthy development process, customer confidence normally drops off, as no working product is immediately visible. Developers answer customer queries using technical slang, and delays are announced. This gives rise to customer resentment.
- On the other hand, an evolutionary approach lets the customer experiment with a working product much earlier than the monolithic approaches. Another important advantage of the incremental model is that it reduces the customer's trauma of getting used to an entirely new system. The gradual introduction of the product via incremental phases provides time to the customer to adjust to the new product. Also, from the customer's financial viewpoint, incremental development does not require a large upfront

END