TED (10)-4072 Reg. No. ………………………….

(REVISION-2010) Signature …….…………………

FIFTH SEMESTER DIPLOMA EXAMINATION IN COMPUTER ENGINEERING AND INFORMATIONTECHNOLIGY-MARCH, 2014

SOFTWARE ENGINEERING

[*Time:* 3 hours

(Maximum marks: 100)

Marks

PART –A
(Maximum marks: 10)

I. Answer all questions in two sentences. Each question carries 2 marks.

1. Define Risk.

A risk is any anticipated unfavorable event or circumstance that can occur while a project is underway.

2. List any two limitations of LOC.

- LOC gives a numerical value of problem size that can vary widely with individual coding style – different programmers lay out their code in different ways.
- LOC measure correlates poorly with the quality and efficiency of the code.

3. Write the outcome of requirement analysis.

The main purpose of the requirements analysis activity is to analyze the collected information to obtain a clear understanding of the product to be developed, with a view to removing all ambiguities, incompleteness, and inconsistencies from the initial customer perception of the problem.

4. Define MTTF.

Mean Time To Failure is the average time between two successive failures, observed over a large number of failures.

5. Define Data Dictionary Interface.

The data dictionary interface should provide view and update access to the entities and relations stored in it. It should have print facility to obtain hard copy of the viewed screens.

PART – B

II. Answer *any five* questions. Each question carries 6 marks

1. Summarize the scheduling task in project planning activity.

Project-task scheduling is an important project planning activity. It involves deciding which tasks would be taken up when. In order to schedule the project activities, a software project manager needs to do the following:

1. Identify all the tasks needed to complete the project.
2. Break down large tasks into small activities.
3. Determine the dependency among different activities.
4. Establish the most likely estimates for the time durations necessary to complete the activities.
5. Allocate resources to activities.
6. Plan the starting and ending dates for various activities.
7. Determine the critical path. A critical path is the chain of activities that determines the duration of the project.

The first step in scheduling a software project involves identifying all the tasks necessary to complete the project. A good knowledge of the intricacies of the project and the development process helps the managers to effectively identify the important tasks of the project. Next, the large tasks are broken down into a logical set of small activities which would be assigned to different engineers. The work breakdown structure formalism helps the manager to breakdown the tasks systematically.

After the project manager has broken down the tasks and created the work breakdown structure, he has to find the dependency among the activities. Dependency among the different activities determines the order in which the different activities would be carried out. If an activity A requires the results of another activity B, then activity A must be scheduled after activity B. In general, the task dependencies define a partial ordering among tasks, i.e. each tasks may precede a subset of other tasks, but some tasks might not have any precedence ordering defined between them (called concurrent task). The dependency among the activities is represented in the form of an activity network.

2. Explain the spiral life cycle model.

   - Spiral life cycle model

   The Spiral model of software development is shown in fig. 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. The following activities are carried out during each phase of a spiral model.

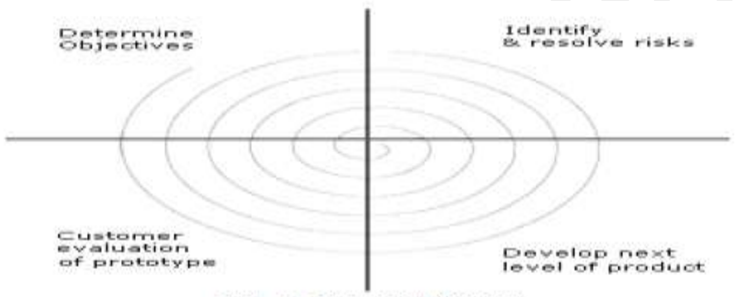

Determine Objectives
Identify & resolve risks
Customer evaluation of prototype
Develop next level of product

Fig. 2.2: 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.

❖ 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.

3. Compare Graphical user Interface and text based Interface.

   The following comparisons are based on various characteristics of a GUI with those of a text-based user interface.

   - In a GUI multiple windows with different information can simultaneously be displayed on the user screen. This is perhaps one of the biggest advantages of GUI over text-based interfaces since the user has the flexibility to simultaneously interact with several related items at any time and can have access to different system information displayed in different windows.
   - Iconic information representation and symbolic information manipulation is possible in a GUI. Symbolic information manipulation such as dragging an icon representing a file to a trash can be deleting is intuitively very appealing and the user can instantly remember it.
   - A GUI usually supports command selection using an attractive and user-friendly menu selection system.
   - In a GUI, a pointing device such as a mouse or a light pen can be used for issuing commands. The use of a pointing device increases the efficacy issue procedure.
   - On the flip side, a GUI requires special terminals with graphics capabilities for running and also requires special input devices such a mouse. On the other hand, a text-based user interface can be implemented even on a cheap alphanumeric display terminal. Graphics terminals are usually much more expensive than alphanumeric terminals. However, display terminals with graphics capability with bit-mapped high-resolution displays and significant amount of local processing power have become affordable and over the years have replaced text-based terminals on all desktops. Therefore, the emphasis of this lesson is on GUI design rather than text- based user interface design.

4. Write down the characteristics of good SRS.

   The important properties of a good SRS document are the following:

- Concise:-

The SRS document should be concise and at the same time unambiguous, consistent, and complete. Verbose and irrelevant descriptions reduce readability and also increase error possibilities.

- Structured:-

  It should be well-structured. A well-structured document is easy to understand and modify. In practice, the SRS document undergoes several revisions to cope up with the customer requirements. Often, the customer requirements evolve over a period of time. Therefore, in order to make the modifications to the SRS document easy, it is important to make the document well-structured.

- Black-box view:-

  It should only specify what the system should do and refrain from stating how to do these. This means that the SRS document should specify the external behavior of the system and not discuss the implementation issues. The SRS document should view the system to be developed as black box, and should specify the externally visible behavior of the system. For this reason, the SRS document is also called the black-box specification of a system.

- Conceptual integrity:-

  It should show conceptual integrity so that the reader can easily understand it.

- Response to undesired events:-

  It should characterize acceptable responses to undesired events. These are called system response to exceptional conditions.

- Traceable:-

  It should be possible to trace a specific requirement to the design elements that implement it and vice versa. Similarly, it should be possible to trace requirement to the code segments that implement it and the test cases that test the requirement and vice versa. Traceability is important to verify the results of a phase with the previous phase, to analyze the impact of a change, etc.

- Verifiable:-

  All requirements of the system as documented in the SRS document should verifiable. This means that it should be possible to determine whether or not requirements have been met in an implementation. Requirements such as the system should be user-friendly are not verifiable.

5. Explain Equivalence class partitioning with example.

❖ Equivalence Class Partitioning

In this approach, the domain of input values to a program is partitioned into a set of equivalence classes. This partitioning is done such that the behavior of the program is similar for every input data belonging to the same equivalence class. The main idea behind defining the equivalence classes is that testing the code with any one value belonging to an equivalence class is as good as testing the software with any other value belonging to that equivalence class. Equivalence classes for software can be designed by examining the input data and output data. The following are some general guidelines for designing the equivalence classes:

1. If the input data values to a system can be specified by a range of values, then one valid and two invalid equivalence classes should be defined.
2. If the input data assumes values from a set of discrete members of some domain, then one equivalence class for valid input values and another equivalence class for invalid input values should be defined.

Example:-

Design the black-box test suite for the following program. The program computes the intersection point of two straight lines and displays the result. It reads two integer pairs (m1, c1) and (m2, c2) defining the two straight lines of the form y=mx + c.

The equivalence classes are the following:

- Parallel lines (m1=m2, c1≠c2)
- Intersecting lines (m1≠m2)
- Coincident lines (m1=m2, c1=c2)

Now, selecting one representative value from each equivalence class, the test suit (2, 2) (2, 5), (5, 5) (7, 7), (10, 10) (10, 10) are obtained.

6. Outline the quality factors for a software product.

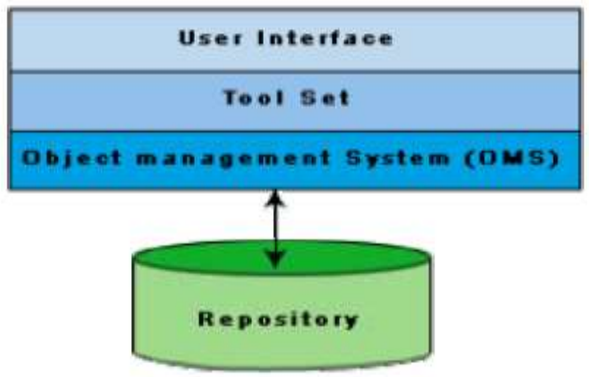
The modern view of a quality associates with a software product several quality factors such as the following:

- Portability: A software product is said to be portable, if it can be easily made to work in different operating system environments, in different machines, with other software products, etc.
- Usability: A software product has good usability, if different categories of users (i.e. both expert and novice users) can easily invoke the functions of the product.
- Reusability: A software product has good reusability, if different modules of the product can easily be reused to develop new products.
- Correctness: A software product is correct, if different requirements as specified in the SRS document have been correctly implemented.
- Maintainability: A software product is maintainable, if errors can be easily corrected as and when they show up, new functions can be easily added to the product, and the functionalities of the product can be easily modified, etc.

7. Explain the architecture of CASE environment with a diagram.

- Architecture of a CASE environment

The architecture of a typical modern CASE environment is shown diagrammatically in fig. 15.2. The important components of a modern CASE environment are user interface, tool set, object management system (OMS), and a repository.



- User interface

  The user interface provides a consistent framework for accessing the different tools thus making it easier for the users to interact with the different tools and reducing the overhead of learning how the different tools are used.

- Object-Management System(OMS) and repository

  Different case tools represent the software product as a set of entities such as specification, design, text data, project plan, etc. The object management system maps these logical entities such into the underlying storage management system (repository). The commercial relational database management systems are geared towards supporting large volumes of information structured as simple relatively short records. There are a few types of entities but large number of instances. By contrast, CASE tools create a large number of entity and relation types with perhaps a few instances of each. Thus the object management system takes care of appropriately mapping into the underlying storage management system.

PART – C

(Answer *one* full question from each unit. Each question carries 15 marks)

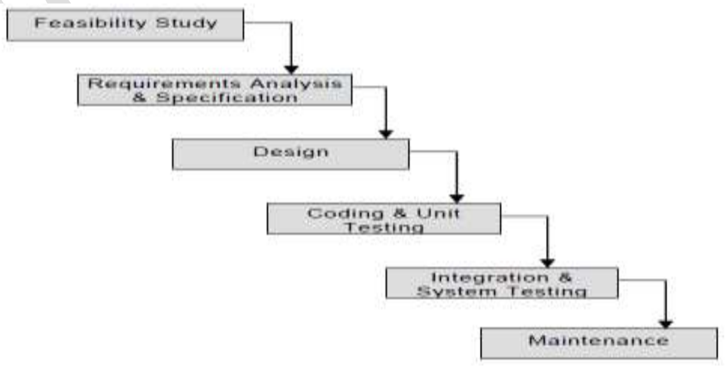
UNIT – I

III. (a) Illustrate waterfall model with a neat sketch.

❖ 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't 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:-


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

• Feasibility Study

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

- 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
- 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.
- 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.
- 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. Integration is normally carried out incrementally over a number of steps. During each integration step, the partially integrated system is tested anda 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.

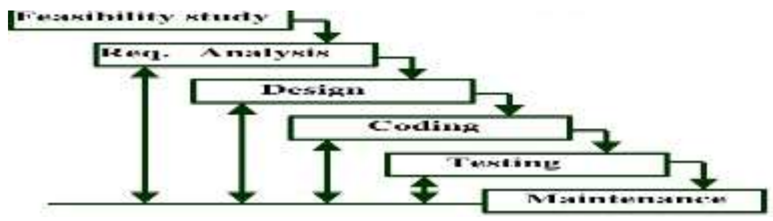
- <u>Maintenance</u>

  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.

❖ <u>Iterative Waterfall Model</u>

The classical waterfall model is not a practical model in the sense that it can't be used in actual software development projects. Thus, this model can be considered to be a theoretical way of developing software.The iterative waterfall model as making necessary changes to the classical waterfall model so that it becomes applicable to practical software development projects. Essentially the main change to the classical waterfall model is in the form of providing feedback paths from every phase to its preceding phases as shown in fig. The feedback paths allow for correction of the errors committed during a phase, as and when these are detected in a later phase. For example, if during testing a design error is identified, then the feedback path allows the design to be reworked and the changes to be reflected in design documents. However, observe that there is no feedback path to the feasibility stage. This means that the feasibility study errors cannot be corrected. Though errors are inevitable in almost every phase of development, it is desirable to detect these errors in the same phase, in which they occur.



- <u>Shortcomings of the Iterative Waterfall Model</u>
  1. The waterfall model cannot satisfactorily handle the different types of risks that a real life software project may suffer from.

To achieve better efficiency and higher productivity, most real life projects find it difficult to follow the rigid phase sequence prescribed by the waterfall model.

(b) Describe the contents of software project management plan (SPMP).

<u>Software Project Management Plan (SPMP)</u>

Once project planning is complete, project managers document their plans in a Software Project Management Plan (SPMP) document. The SPMP document should discuss a list of different items that have been discussed below. This list can be used as a possible organization of the SPMP document.

Organization of the Software Project Management Plan (SPMP) Document

1. Introduction
   a) Objectives
   b) Major Functions
   c) Performance Issues
   d) Management and Technical Constraints
2. Project Estimates
   a) Historical Data Used
   b) Estimation Techniques Used
   c) Effort, Resource, Cost, and Project Duration Estimates
3. Schedule
   a) Work Breakdown Structure
   b) Task Network Representation
   c) Gantt Chart Representation
   d) PERT Chart Representation

4. Project Resources
    a) People
    b) Hardware and Software
    c) Special Resources
5. Staff Organization
    a) Team Structure
    b) Management Reporting
6. Risk Management Plan
    a) Risk Analysis
    b) Risk Identification
    c) Risk Estimation
    d) Risk Abatement Procedures
7. Project Tracking and Control Plan
8. Miscellaneous Plans
    a) Process Tailoring
    b) Quality Assurance Plan
    c) Configuration Management Plan
    d) Validation and Verification
    e) System Testing Plan
    f) Delivery, Installation, and Maintenance Plan

OR

IV.

a) Explain function point metric and feature point metric.

- Function point (FP)

Function point metric was proposed by Albrecht [1983]. This metric overcomes many of the shortcomings of the LOC metric. The conceptual idea behind the function point metric is that the size of a software product is directly dependent on the number of different functions or features it supports. A software product supporting many features would certainly be of larger size than a product with less number of features. Each function when invoked reads some input data and transforms it to the corresponding output data.

Besides using the number of input and output data values, function point metric computes the size of a software product (in units of functions points or FPs) using three other characteristics of the product as shown in the following expression. The size of a product in function points (FP) can be expressed as the weighted sum of these five problem characteristics. The weights associated with the five characteristics were proposed empirically and validated by the observations over many projects. Function point is computed in two steps. The first step is to compute the unadjusted function point (UFP).

UFP = (Number of inputs)*4 + (Number of outputs)*5 + (Number of inquiries)*4 + (Number of files)*10 + (Number of interfaces)*10

Once the unadjusted function point (UFP) is computed, the technical complexity factor (TCF) is computed next. TCF refines the UFP measure by considering fourteen other factors such as high transaction rates, throughput, and response time requirements, etc. Each of these 14 factors is assigned from 0 (not present or no influence) to 6 (strong influence). The resulting numbers are summed, yielding the total degree of influence (DI). Now, TCF is computed as (0.65+0.01*DI). As DI can vary from 0 to 70, TCF can vary from 0.65 to 1.35. Finally, FP=UFP*TCF.

- Feature point metric

A major shortcoming of the function point measure is that it does not take into account the algorithmic complexity of a software. That is, the function point metric implicitly assumes that the effort required to design and develop any two functionalities of the system is the same. But, we know that this is normally not true, the effort required to develop any two functionalities may vary widely. It only takes the number of functions that the system supports into consideration without distinguishing the difficulty level of developing the various functionalities. To overcome this problem, an extension of the function point metric called feature point metric is proposed.

Feature point metric incorporates an extra parameter algorithm complexity. This parameter ensures that the computed size using the feature point metric reflects the fact that the more is the complexity of a function, the greater is the effort required to develop it and therefore its size should be larger compared to simpler functions.

b) Summarize data structure and object oriented design methods.

Data structure design method

The data structure-oriented design approach utilizes the data structures of the input data, internal data (for example databases) and output data to develop software. In the data structure-oriented approach, the emphasis is on the object, which is the data. The structure of information, called data structure, has an important impact on the complexity and efficiency of algorithms designed to process information.

Software design is closely related to the data structure of the system, for example, alternative data will require a conditional processing element, and repetitive data will require a control feature for repetition and hierarchical data structure will require a hierarchical software structure. Data structure-oriented design is best utilized in applications that are a well-defined, hierarchical structure of information.

Object oriented design

Object-oriented software design is a design strategy where system designers think in terms of 'things' instead of operations or functions. The executing system is made up of interacting objects that maintain their own local state and provide operations on that state information. They hide information about the representation of the state and hence limit access to it. An object-oriented design process involves designing the object classes and the relationships between these classes. When the design is realized as an executing program, the required objects are created dynamically using the class definitions.

UNIT – II

V.

a) Explain command language and menu base user interfaces

- Command language based interfaces

A command language-based interface as the name itself suggests, is based on designing a command language which the user can use to issue the commands. The user is expected to frame the appropriate commands in the language and type them in appropriately whenever required. A simple command language-based interface might simply assign unique names to the different commands. However, a more sophisticated command language-based interface may allow users to compose complex commands by using a set of primitive commands. Such a facility to compose commands dramatically reduces the number of command names one would have to remember. Thus, a command language-based interface can be made concise requiring minimal typing by the user. Command language-based interfaces allow fast interaction with the computer and simplify the input of complex commands.

Command language-based interfaces suffer from several drawbacks. Usually, command language-based interfaces are difficult to learn and require the user to memorize the set of primitive commands. Also, most users make errors while formulating commands in the command language and also while typing them in. Further, in a command language-based interface, all interactions with the system is through a key-board and can not take advantage of effective interaction devices such as a mouse. Obviously, for casual and inexperienced users, command language-based interfaces are not suitable.

- Issues in designing a command language-based interface:-
    - ❖ The designer has to decide what mnemonics are to be used for the different commands.
    - ❖ The designer has to decide whether the users will be allowed to redefine the command names to suit their own preferences.
    - ❖ The designer has to decide whether it should be possible to compose primitive commands to form more complex commands.

- Menu-based interfaces

An important advantage of a menu-based interface over a command language-based interface is that a menu-based interface does not require the users to remember the exact syntax of the commands. A menu-based interface is based on recognition of the command names, rather than recollection. Further, in a menu-based interface the typing effort is minimal as most interactions are carried out through menu selections using a pointing device. This factor is an important consideration for the occasional user who cannot type fast.

However, experienced users find a menu-based user interface to be slower than a command language-based interface because an experienced user can type fast and can get speed advantage by composing different primitive commands to express complex commands. Composing commands in a menu-based interface is not possible. This is because of the fact that actions involving logical connectives (and, or, etc.) are awkward to specify in a menu-based system. Also, if the number of choices is large, it is difficult to select from the menu. In fact, a major challenge in the design of a menu-based interface is to structure large number of menu choices into manageable forms.

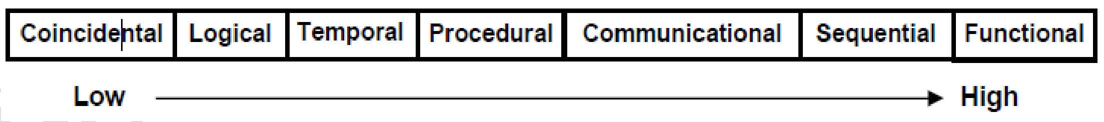
- ❖ Scrolling menu:-
- ❖ Walking menu:-
- ❖ Hierarchical menu:-

b) Compare cohesion and coupling.

- **Cohesion:-**

Cohesion is a measure of functional strength of a module. A module having high cohesion and low coupling is said to be functionally independent of other modules.

The different classes of cohesion that a module may possess are depicted in fig.



Coincidental cohesion:-A module is said to have coincidental cohesion, if it performs a set of tasks that relate to each other very loosely, if at all. In this case, the module contains a random collection of functions. It is likely that the functions have been put in the module out of pure coincidence without any thought or design.

Logical cohesion:-A module is said to be logically cohesive, if all elements of the module perform similar operations, e.g. error handling, data input, data output, etc.

Temporal cohesion:-When a module contains functions that are related by the fact that all the functions must be executed in the same time span, the module is said to exhibit temporal cohesion. The set of functions responsible for initialization, start-up, shutdown of some process, etc. exhibit temporal cohesion.

Procedural cohesion:-A module is said to possess procedural cohesion, if the set of functions of the module are all part of a procedure (algorithm) in which certain sequence of steps have to be carried out for achieving an objective, e.g. the algorithm for decoding a message.

Communicational cohesion:- A module is said to have communicational cohesion, if all functions of the module refer to or update the same data structure, e.g. the set of functions defined on an array or a stack.

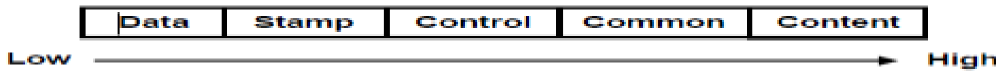
Sequential cohesion:-A module is said to possess sequential cohesion, if the elements of a module form the parts of sequence, where the output from one element of the sequence is input to the next.

Functional cohesion:-Functional cohesion is said to exist, if different elements of a module cooperate to achieve a single function.

- **Coupling:-**

Coupling between two modules is a measure of the degree of interdependence or interaction between the two modules. A module having high cohesion and low coupling is said to be functionally independent of other modules.

Even if there are no techniques to precisely and quantitatively estimate the coupling between two modules, classification of the different types of coupling will help to quantitatively estimate the degree of coupling between two modules. Five types of coupling can occur between any two modules. This is shown in fig.



Data coupling:-Two modules are data coupled, if they communicate through a parameter.

Stamp coupling:-Two modules are stamp coupled, if they communicate using a composite data item such as a record in PASCAL or a structure in C.

Control coupling:-Control coupling exists between two modules, if data from one module is used to direct the order of instructions execution in another.

Common coupling:-Two modules are common coupled, if they share data through some global data items.

Content coupling:- Content coupling exists between two modules, if they share code, e.g. a branch from one module into another module.

OR

VI. List the contents of a good SRS and prepare an SRS document for the following software product. Implement a Library Management System having the provision to Add, Delete, Modify Members and to perform transactions like Book issue and Book return.

The benefits of a good SRS are,

- A contract between the customer and the software vendor – A good SRS document specifies all the features required in the final system including technical requirements and interface requirements. SRS document is used by the customer to determine whether the software vendor has provided all the features in the delivered software system. To the Software vendor it provides a solid foundation to fix the scope of the software system.
- Enables costing and pricing of the project – A well defined SRS enables software developers to accurately estimate the amount of effort required to build the software product. Function point analysis and SMC are some the techniques adopted for estimating effort.
- Input for detailed design – A good SRS enables experienced developers to convert the requirements directly to a technical design. For example, a well defined data dictionary can be easily converted to a database specification.
- Management of customer expectations – Since SRS precisely defines project scope, it ensures that customer expectations don't change during software development. If they do, SRS can be modified and costing/pricing can be done again on the changes required.

1.Introduction

1.1. Purpose

The main objective of this document is to illustrate the requirements of the project Library Management system. The document gives the detailed description of the both functional and non functional requirements proposed by the client. The document is developed after a number of consultations with the client and considering the complete requirement specifications of the given Project. The final product of the team will be meeting the requirements of this document.

1.2. Scope of the project

The system accepts the General Library Transactions of book like issue, return and renewals for the members . The different areas where we can use this application are :

• Any education institute can make use of it for providing information about author, content of the available books.

• It can be used in offices and modifications can be easily done according to requirements.

1.3. Conventions Used

The following are the list of conventions and acronyms used in this document and the project as well:

• Administrator: A login id representing a user with user administration privileges to the software .

• User: A general login id assigned to most users

• Client: Intended users for the software

• SQL: Structured Query Language; used to retrieve information from a database .

• SQL Server: A server used to store data in an organized format .

• Unique Key: Used to differentiate entries in a database .

2. Overall Description

2.1. Product Perspective

The proposed Library Management System will provide a search functionality to facilitate the search of resources. This search will be based on various categories viz. book name . Also Advanced Search feature is provided in order to search various categories simultaneously. Further the library staff personnel can add/update/remove the resources and the resource users from the system.

2.2. Product Features

There are two different users who will be using this product:

• Librarian who will be acting as the administrator

• Student of the University who will be accessing the Library online.

The features that are available to the Librarian are:

- A librarian can issue a book to the student
- Can view The different categories of books available in the Library .
- Can view the List of books available in each category
- Can take the book returned from students
- Add books and their information of the books to the database
- Edit the information of the existing books.
- Can check the report of the issued Books.
- Can access all the accounts of the students.

The features available to the Students are:

- Can view The different categories of books available in the Library
- Can view the List of books available in each category
- Can own an account in the library
- Can view the books issued to him
- Can put a request for a new book
- Can view the history of books issued to him previously
- Can search for a particular book.

UNIT – III

VII.

a) Explain Integration testing and System testing.

- Integration Testing

The primary objective of integration testing is to test the module interfaces, i.e. there are no errors in the parameter passing, when one module invokes another module. During integration testing, different modules of a system are integrated in a planned manner using an integration plan. The integration plan specifies the steps and the order in which modules are combined to realize the full system. After each integration step, the partially integrated system is tested. An important factor that guides the integration plan is the module dependency graph. The structure chart (or module dependency graph) denotes the order in which different modules call each other. By examining the structure chart the integration plan can be developed.

There are four types of integration testing approaches. Any one (or a mixture) of the following approaches can be used to develop the integration test plan. Those approaches are the following:

- Big bang approach

  It is the simplest integration testing approach, where all the modules making up a system are integrated in a single step. In simple words, all the modules of the system are simply put together and tested. However, this technique is practicable only for very small systems.

- Top-down approach

  Top-down integration testing starts with the main routine and one or two subordinate routines in the system. After the top-level 'skeleton' has been tested, the immediately subroutines of the 'skeleton' are combined with it and tested. Top-down integration testing approach requires the use of program stubs to simulate the effect of lower-level routines that are called by the routines under test.

- Bottom-up approach

  In bottom-up testing, each subsystem is tested separately and then the full system is tested. A subsystem might consist of many modules which communicate among each other through well-defined interfaces. The primary purpose of testing each subsystem is to test the interfaces among various modules making up the subsystem. Both control and data interfaces are tested. The test cases must be carefully chosen to exercise the interfaces in all possible manners.

- Mixed-approach

  A mixed (also called sandwiched) integration testing follows a combination of top- down and bottom-up testing approaches. In top-down approach, testing can start only after the top-level modules have been coded and unit tested. Similarly, bottom-up testing can start only after the bottom level modules are ready. The mixed approach overcomes this shortcoming of the top-down and bottom-up approaches. In the mixed testing approaches, testing can start as and when modules become available. Therefore, this is one of the most commonly used integration testing approaches.

  - System testing

System tests are designed to validate a fully developed system to assure that it meets its requirements. There are essentially three main kinds of system testing:

- ❖ Alpha Testing. Alpha testing refers to the system testing carried out by the test team within the developing organization.
- ❖ Beta testing. Beta testing is the system testing performed by a select group of friendly customers.
- ❖ Acceptance Testing. Acceptance testing is the system testing performed by the customer to determine whether he should accept the delivery of the system.

In each of the above types of tests, various kinds of test cases are designed by referring to the SRS document. Broadly, these tests can be classified into functionality and performance tests. The functionality tests test the functionality of the software to check whether it satisfies the functional requirements as documented in the SRS document. The performance tests test the conformance of the system with the nonfunctional requirements of the system.

- ❖ <u>Performance testing</u>

Performance testing is an important type of system testing. Performance testing is carried out to check whether the system needs the non- functional requirements identified in the SRS document. There are several types of performance testing.

- o Stress testing:- Stress testing is also known as endurance testing. Stress testing evaluates system performance when it is stressed for short periods of time.
- o Volume Testing:- It is especially important to check whether the data structures (arrays, queues, stacks, etc.) have been designed to successfully extraordinary situations.
- o Configuration Testing:- This is used to analyze system behavior in various hardware and software configurations specified in the requirements.
- o Compatibility Testing:- This type of testing is required when the system interfaces with other types of systems.
- o Regression Testing:- This type of testing is required when the system being tested is an upgradation of an already existing system to fix some bugs or enhance functionality, performance, etc.

- Recovery Testing:- Recovery testing tests the response of the system to the presence of faults, or loss of power, devices, services, data, etc.
- Maintenance Testing:- This testing addresses the diagnostic programs, and other procedures that are required to be developed to help maintenance of the system. It is verified that the artifacts exist and they perform properly.
- Documentation Testing :- It is checked that the required user manual, maintenance manuals, and technical manuals exist and are consistent.
- Usability Testing:- Usability testing concerns checking the user interface to see if it meets all user requirements concerning the user interface.

b) Comment on terms:

i. Software reuse
ii. Software maintenance

i. <u>Software reuse</u>

Software products are expensive. Software project managers are worried about the high cost of software development and are desperately look for ways to cut development cost. A possible way to reduce development cost is to reuse parts from previously developed software. In addition to reduced development cost and time, reuse also leads to higher quality of the developed products since the reusable components are ensured to have high quality.

It is important to know about the kinds of the artifacts associated with software development that can be reused. Almost all artifacts associated with software development, including project plan and test plan can be reused. However, the prominent items that can be effectively reused are: Requirements specification, Design, Code, Test cases and Knowledge.

The following are some of the basic issues that must be clearly understood for starting any reuse program.

- Component creation
- Component indexing and storing
- Component search
- Component understanding

- Component adaptation
- Repository maintenance

ii. Software maintenance

Software maintenance is becoming an important activity of a large number of software organizations. This is no surprise, given the rate of hardware obsolescence, the immortality of a software product per se, and the demand of the user community to see the existing software products run on newer platforms, run in newer environments, and/or with enhanced features. When the hardware platform is changed, and a software product performs some low-level functions, maintenance is necessary. Also, whenever the support environment of a software product changes, the software product requires rework to cope up with the newer interface. For instance, a software product may need to be maintained when the operating system changes. Thus, every software product continues to evolve after its development through maintenance efforts. Therefore it can be stated that software maintenance is needed to correct errors, enhance features, port the software to new platforms, etc.

There are basically three types of software maintenance. These are:

- Corrective: Corrective maintenance of a software product is necessary to rectify the bugs observed while the system is in use.
- Adaptive: A software product might need maintenance when the customers need the product to run on new platforms, on new operating systems, or when they need the product to interface with new hardware or software.
- Perfective: A software product needs maintenance to support the new features that users want it to support, to change different functionalities of the system according to customer demands, or to enhance the performance of the system.

OR

VIII.

(a) Explain the Debugging approaches.

- Brute Force Method:

This is the most common method of debugging but is the least efficient method. In this approach, the program is loaded with print statements to print the intermediate values with the hope that some of the printed values will help to identify the statement in error. This approach becomes more systematic with the use of a symbolic debugger (also called a source code debugger), because values of different variables can be easily checked and break points and watch points can be easily set to test the values of variables effortlessly.

- Backtracking:

  This is also a fairly common approach. In this approach, beginning from the statement at which an error symptom has been observed, the source code is traced backwards until the error is discovered. Unfortunately, as the number of source lines to be traced back increases, the number of potential backward paths increases and may become unmanageably large thus limiting the use of this approach.

- Cause Elimination Method:

  In this approach, a list of causes which could possibly have contributed to the error symptom is developed and tests are conducted to eliminate each. A related technique of identification of the error from the error symptom is the software fault tree analysis.

- Program Slicing:

  This technique is similar to back tracking. Here the search space is reduced by defining slices. A slice of a program for a particular variable at a particular statement is the set of source lines preceding this statement that can influence the value of that variable [Mund2002].

(b) Discuss reliability and reliability metrics.

Reliability metrics

The reliability requirements for different categories of software products may be different. For this reason, it is necessary that the level of reliability required for a software product should be specified in the SRS (software requirements specification) document. In order to be able to do this, some metrics are needed to quantitatively express the reliability of a software product. A good reliability measure should be observer-dependent, so that different people can agree on the degree of reliability a system has. There are six reliability metrics which can be used to quantify the reliability of software products.

• Rate of occurrence of failure (ROCOF):- ROCOF measures the frequency of occurrence of unexpected behavior (i.e. failures). ROCOF measure of a software product can be obtained by observing the behavior of a software product in operation over a specified time interval and then recording the total number of failures occurring during the interval.

• Mean Time To Failure (MTTF):- MTTF is the average time between two successive failures, observed over a large number of failures. To measure MTTF, we can record the failure data for n failures. Let the failures occur at the time instants t1, t2, …, tn. Then, MTTF can be calculated as 1 1 (1) n ii itt n +− = − ∑ . It is important to note that only run time is considered in the time measurements, i.e. the time for which the system is down to fix the error, the boot time, etc are not taken into account in the time measurements and the clock is stopped at these times.

• Mean Time To Repair (MTTR):- Once failure occurs, some timeis required to fix the error. MTTR measures the average time it takes to track the errors causing the failure and to fix them.

• Mean Time Between Failure (MTBR):- MTTF and MTTR can be combined to get the MTBR metric: MTBF = MTTF + MTTR. Thus, MTBF of 300 hours indicates that once a failure occurs, the next failure is expected after 300 hours. In this case, time measurements are real time and not the execution time as in MTTF.

• Probability of Failure on Demand (POFOD):- Unlike the other metrics discussed, this metric does not explicitly involve time measurements. POFOD measures the likelihood of the system failing when a service request is made. For example, a POFOD of 0.001 would mean that 1 out of every 1000 service requests would result in a failure.

UNIT – IV

IX.

a) Summarize CASE support in software life cycle.

▪ Prototyping Support:

Prototyping is useful to understand the requirements of complex software products, to demonstrate a concept, to market new ideas, and so on. The important features of a prototyping CASE tool are as follows:

- Define user interaction
- Define the system control flow

- Store and retrieve data required by the system
- Incorporate some processing logic

A good prototyping tool should support the following features:

- Since one of the main uses of a prototyping CASE tool is graphical user interface (GUI) development, prototyping CASE tool should support theuser to create a GUI using a graphics editor. The user should be allowed to define all data entry forms, menus and controls.
- It should integrate with the data dictionary of a CASE environment.
- If possible, it should be able to integrate with external user defined modules written in C or some popular high level programming languages.
- The user should be able to define the sequence of states through which a created prototype can run. The user should also be allowed to control the running of the prototype.
- The run time system of prototype should support mock runs of the actual system and management of the input and output data.

- <u>Structured analysis and design</u>

Several diagramming techniques are used for structured analysis and structured design. A CASE tool should support one or more of the structured analysis and design techniques. It should support effortlessly drawing analysis and design diagrams.It should support drawing for fairly complex diagrams, preferably through a hierarchy of levels.The CASE tool should provide easy navigation through the different levels and through the design and analysis.The tool must support completeness and consistency checking across the design and analysis and through all levels of analysis hierarchy. Whenever it is possible, the system should disallow any inconsistent operation, but it may be very difficult to implement such a feature. Whenever there arises heavy computational load while consistency checking, it should be possible to temporarily disable consistency checking.

- <u>Code generation</u>

As far as code generation is concerned, the general expectation of a CASE tool is quite low. A reasonable requirement is traceability from source file to design data. The CASE tool should support generation of module skeletons or templates in one or more popular languages. It should be possible to include copyright message, brief description of the module, author name and the date of creation in some selectable format. The tool should generate records, structures, class definition automatically from the contents of the data dictionary in one or more popular languages. It should generate database tables for relational database management systems. The tool should generate code for user interface from prototype definition for X window and MS window based applications.

- Test case generation

  The CASE tool for test case generation should have the following features:

  - It should support both design and requirement testing.
  - It should generate test set reports in ASCII format which can be directly imported into the test plan document.

b) Explain the benefits of CASE.

Several benefits accrue from the use of a CASE environment or even isolated CASE tools. Some of those benefits are:

• A key benefit arising out of the use of a CASE environment is cost saving through all development phases. Different studies carry out to measure the impact of CASE put the effort reduction between 30% to 40%.

• Use of CASE tools leads to considerable improvements to quality. This is mainly due to the facts that one can effortlessly iterate through the different phases of software development and the chances of human error are considerably reduced.

• CASE tools help produce high quality and consistent documents. Since the important data relating to a software product are maintained in a central repository, redundancy in the stored data is reduced and therefore chances of inconsistent documentation is reduced to a great extent.

• CASE tools take out most of the drudgery in a software engineer's work. For example, they need not check meticulously the balancing of the DFDs but can do it effortlessly through the press of a button.

• CASE tools have led to revolutionary cost saving in software maintenance efforts. This arises not only due to the tremendous value of a CASE environment in traceability and consistency checks, but also due to the systematic information capture during the various phases of software development as a result of adhering to a CASE environment.

• Introduction of a CASE environment has an impact on the style of working of a company, and makes it oriented towards the structured and orderly approach.

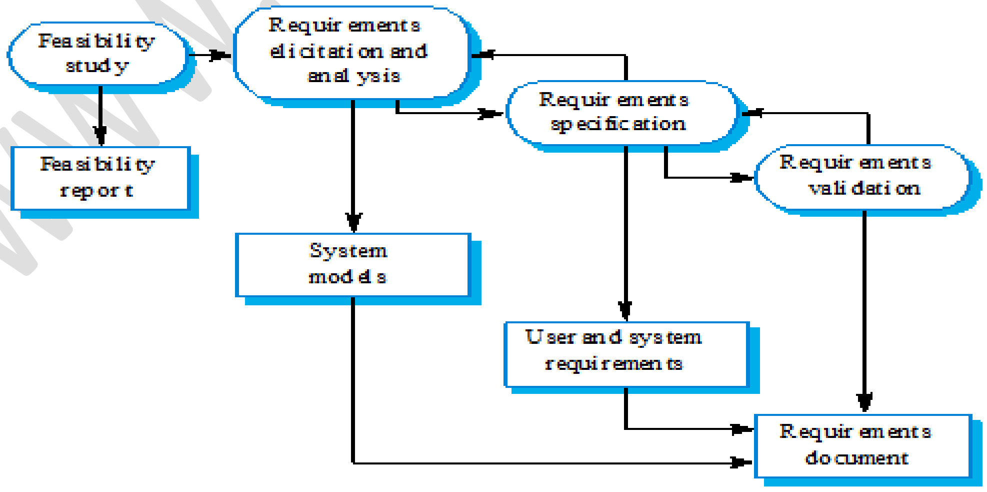
X. Illustrate a case study to build software that automates the Banking system including deposit and withdrawal. Highlights design and testing phases.

Objectives

- To describe the principal requirements engineering activities and their relationships
- To introduce techniques for requirements elicitation and analysis
- To describe requirements validation and the role of requirements reviews
- To discuss the role of requirements management in support of other requirements engineering processes

Requirements engineering processes

- The processes used for RE vary widely depending on the application domain, the people involved and the organisation developing the requirements.
- However, there are a number of generic activities common to all processes
    - Requirements elicitation;
    - Requirements analysis;
    - Requirements validation;
    - Requirements management.



Feasibility studies

    - A feasibility study decides whether or not the proposed system is worthwhile or doable.
    - A short focused study that checks
        - If the system contributes to organisational objectives;
        - If the system can be engineered using current technology and within budget;
        - If the system can be integrated with other systems that are used.

<u>Feasibility study implementation</u>

- Based on information assessment (what is required), information collection and report writing.
- Questions for people in the organisation
    - What if the system wasn’t implemented?
    - What are current process problems?
    - How will the proposed system help?
    - What will be the integration problems?
    - Is new technology needed? What skills?
    - What facilities must be supported by the proposed system?

<u>Requirements </u>

- Open an account for a customer (savings or chequing)
- Deposit Withdraw
- Display details of an account Change LOC
- Produce monthly statements Print a list of customers

<u> Ambiguities</u>

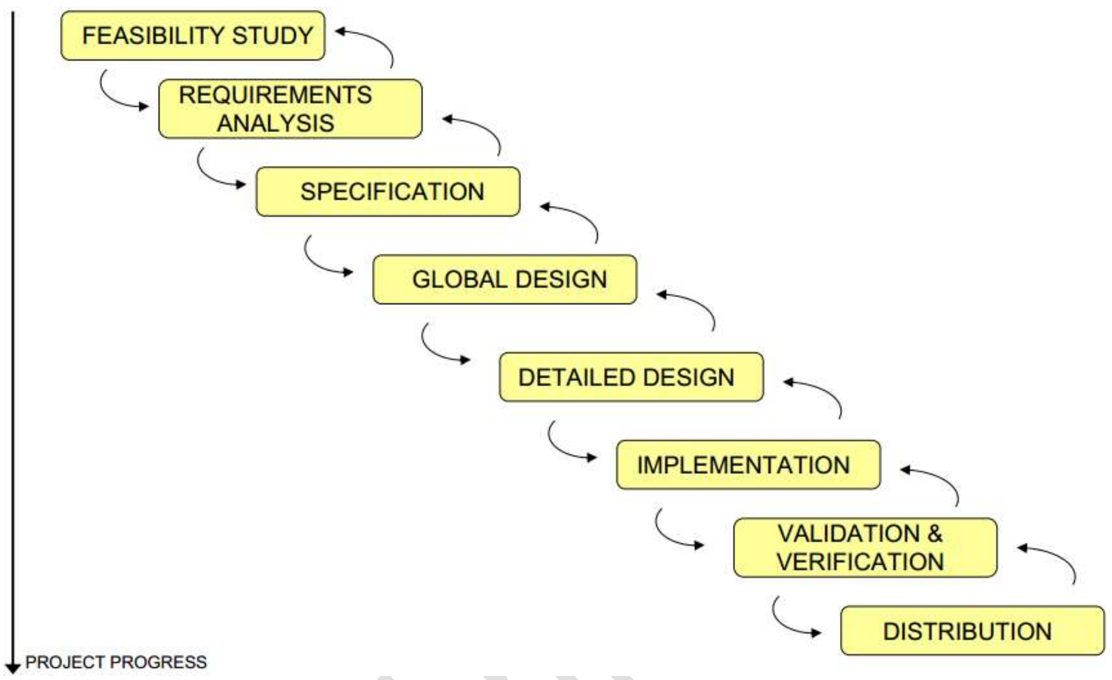
What is the difference between savings and chequing?

<u>CASE STUDY 1</u>

How should we go from Requirements to Code?

- Two basic approaches Plan-driven (waterfall type models)
- Agile (incremental approaches)

The waterfall model of the lifecycle



<u>Problem for the waterfall</u>

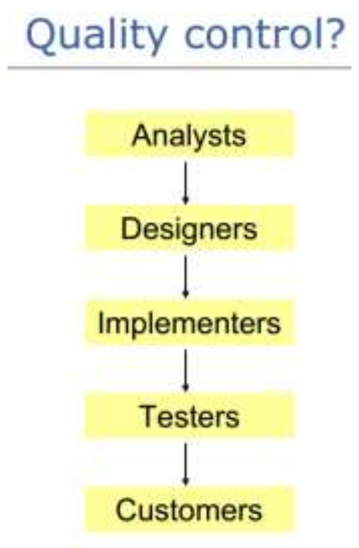
Late appearance of actual code.

Lack of support for requirements change — and more generally for extendibility and reusability. Lack of support for the maintenance activity (70% of software costs?).

Division of labor hampering Total Quality Management.

Impedance mismatches.

Highly synchronous model.



<u>Use Case – Open Account</u>

- Actors: Customer (initiator), Teller ⍰
- Overview: the customer applies for and is granted a new bank account with some initial approved line of credit. The teller logs on to the system, selects open account , and supplies the new account details. The system prints a copy of the contents of the new account with the date, which the teller hands to the customer.

<u>Test Driven Development</u>

1. Write a little Test ⍰
   a. Test will not work initially (there is no code) ⍰
   b. Might not even compile
2. Make the Test work quickly ⍰
   a. Commit whatever sins are necessary in the process (e.g. duplication in the code)
3. Refactor ⍰
   a. Eliminate the duplication created in merely getting the test to work

<u>Open Account Use case (revisited)</u>

- Actors: Customer (initiator), Teller ⍰
- Overview: the customer applies for and is granted a new bank account with some initial approved line of credit. The teller logs on to the system, selects open account , and supplies the new account details. The system prints a copy of the contents of the new account with the date, which the teller hands to the customer.

Withdraw Use Case

- Actors: Customer (initiator) and Teller
- Overview: The customer provides the teller with an account number and a withdrawal amount. The teller selects `withdraw-request’ from the system menu, and enters the data. The System debits the account by the requested withdrawal amount, and prints a receipt with the date and the amount, which the teller gives to the customer.

Bank Challenge question

1. Add a deposit feature
   a. How would you write the test
2. New Requirement: monthly statement
   a. The System must generate a monthly statement for the customer Each deposit or withdrawal must be date stamped
   b. Interest must be added or charged as applicable
3. New Requirement:
   a. Print all bank customers in alphabetical order
   b. Print customers with largest balances in order