Software Engineering End Semester solutions

- Swadesh Nayak
  1729085
- Tanaya Jaiswal
  1729088
- Tanisha Bhardwaj
  1729089
- Tuneer Bhattacharya
  1729091
- Muskan Mohanty
  1729139
- Nibedita Misra
  1729141
- Nilanjan Roy
  1729143
- Saagnik Sarbadhikari
  1729151
- Sagnik Misra
  1729152

10)Sayanha Sengupta

1729158

Date : 05.11.2019

2013

1. a) What is 99% complete syndrome? How does it affect the development process? [2 × 10

   b) What is the impact of developing prototypes on overall cost of development?

   c) List the activities and outcomes of different phases of SDLC.

   d) How is the *project organization* different from the *functional organization*?

   e) When an application was tested in developer's site with 100 machines in a network it was working fine with good performance. But when it was deployed on client's site, it didn't work. Which testing the testing team had missed? Justify your answer.

   f) List various reasons for software crisis.

   g) What is system testing? Discuss about alpha, beta and acceptance testing.

   h) Differentiate between the *Brute Force* method and the *Back Tracking* method used for debugging.

   i) Distinguish between *code walkthrough* and *code inspection.*

   j) "If the size of the software increases two times, the development time doesn't double". (True/False) Justify your answer.

[Swadesh Nayak 1729085]

a) In software development if you do not follow SDLC (Software Development Life Cycle) 99% complete syndrome happens. In this, the last 1 percent of anything takes longer than the other 99 percent. Most oftenly,this percent drags on to days, weeks, and even few months thereby delaying inspections and prohibiting our move.

b) By realizing additional requirements and constraints early, as well as receiving user feedback early, we can make better complexity and time estimates. Thus developing prototypes results in better cost and time estimates. These estimates impact a range of activities when rolling out a solution to production.

c) Given below are the various phases with outcomes:

- Requirement gathering and analysis: The relevant information is gathered and all the ambiguities are resolved.

- Design: Software architecture that is used for implementing system development is derived.
- Implementation or coding: The Software design is translated into source code. All the components of the software are implemented in this phase.
- Testing: Any defects found are fixed and made sure that the software is as per the customer's standard.
- Deployment: Customer along with the developers does the testing. If the customer finds the application as expected, then sign off is provided by the customer to go live.
- Maintenance: if any issue comes up and needs to be fixed or any enhancement is to be done is taken care by the developers.

d) A functional organization is a traditional structure where the organization is divided based on the functions performed by that particular group of people, such as Human Resources, Information Technology, Marketing, Service, etc.In projectized organizations, the majority of the organization's resources are involved in project work and the project work is generally completed for the benefit of an external customer.

e) Testing team must have missed the User Acceptance Testing (UAT). It is also known as beta or end-user testing, and is defined as testing the software by the user or client to determine whether it can be accepted or not. This is the final testing performed once the functional, system and regression testing are completed.

f) Reasons for Software Crisis:1-Lack of communication between software developers and user. 2-Due to increase in size of softwares. 3-Due to increase in complexity of the problem. 4-Lack of understanding of the problem. 5-Due to poor estimation of cost, time, size and planning.

g) System Testing is a type of software testing that is performed on a complete integrated system to evaluate the compliance of the system with the corresponding requirements.

Alpha Testing is like performing usability testing, which is normally done by the in-house developers. Beta Testing is done by the number of the end users before delivery, the change request would be fixed if user gives the feedback or report defects. Acceptance Testing is done after the system testing. It is used to check whether the software meets the customer requirements or not.

h) Brute-force search (BFS) is a foremost common technique of debugging however is that the least economical method. during this approach, the program is loaded with print statements to print the intermediate values with the hope that a number of the written values can facilitate to spot the statement in error whereas Backtracking is also a reasonably common approach but during this approach, starting from the

statement at which an error symptom has been discovered, the source code is derived backward till the error is discovered.

i) Code Walkthrough is a form of peer review in which a programmer leads the review process and the other team members ask questions and spot possible errors against development standards and other issues whereas

Code Inspection is the most formal type of review, which is a kind of static testing to avoid the defect multiplication at a later stage.

j) A task may also need to be broken into many smaller tasks that must each be estimated as well. The estimation process must be applied to each subtask, and then accumulated to get the overall task estimate.Also the ability to accurately estimate time and delivery of a given software development task is a function of the relative size of the task, and is highly exponential. In other words, the bigger the relative size, the more inaccurate the estimate becomes at an exponential rate.Thus, if size of S/W increases two times,development time doesn't double.

2. a) Identify the underlying life cycle model for the following case and discuss this life cycle model.

   A large application has to be developed and the developing organization does not have a well defined process model for testing activity.

b) Define testing. Discuss about the importance of stub and driver module in case of unit testing with suitable example.

(Tanya Jaiswal 1729088)

The underlying life cycle model is agile.

Agile Development Models

In earlier days Iterative Waterfall model was very popular to complete a project. But nowadays developers face various problems while using it to develop a software. The main difficulties included handling change requests from customers during project development and the high cost and time required to incorporate these changes. To overcome these drawbacks of Waterfall model, in the mid-1990s the Agile Software Development model was proposed.

The Agile model was primarily designed to help a project to adapt to change requests quickly. So, the main aim of the Agile model is to facilitate quick project completion. To accomplish this task agility is required. Agility is achieved by fitting the process to the

project, removing activities that may not be essential for a specific project. Also, anything that is wastage of time and effort is avoided.

Actually Agile model refers to a group of development processes. These processes share some basic characteristics but do have certain subtle differences among themselves. A few Agile SDLC models are given below:

- Crystal
- Atern
- Feature-driven development
- Scrum
- Extreme programming (XP)
- Lean development
- Unified process

In the Agile model, the requirements are decomposed into many small parts that can be incrementally developed. The Agile model adopts Iterative development. Each incremental part is developed over an iteration. Each iteration is intended to be small and easily manageable and that can be completed within a couple of weeks only. At a time one iteration is planned, developed and deployed to the customers. Long-term plans are not made.
Agile model is the combination of iterative and incremental process models. Steps involve in agile SDLC models are:

- Requirement gathering
- Requirement Analysis
- Design
- Coding
- Unit testing
- Acceptance testing

The time to complete an iteration is known as a Time Box. Time-box refers to the maximum amount of time needed to deliver an iteration to customers. So, the end date for an iteration does not change. Though the development team can decide to reduce the delivered functionality during a Time-box if necessary to deliver it on time. The central principle of the Agile model is the delivery of an increment to the customer after each Time-box.

Principles of Agile model:

- To establish close contact with the customer during development and to gain a clear understanding of various requirements, each Agile project usually includes a customer representative on the team. At the end of each iteration stakeholders and the customer representative review, the progress made and re-evaluate the requirements.
- Agile model relies on working software deployment rather than comprehensive documentation.
- Frequent delivery of incremental versions of the software to the customer representative in intervals of few weeks.
- Requirement change requests from the customer are encouraged and efficiently incorporated.
- It emphasizes on having efficient team members and enhancing communications among them is given more importance. It is realized that enhanced communication among the development team members can be achieved through face-to-face communication rather than through the exchange of formal documents.
- It is recommended that the development team size should be kept small (5 to 9 peoples) to help the team members meaningfully engage in face-to-face communication and have collaborative work environment.

- Agile development process usually deploy Pair Programming. In Pair programming, two programmers work together at one work-station. One does coding while the other reviews the code as it is typed in. The two programmers switch their roles every hour or so.

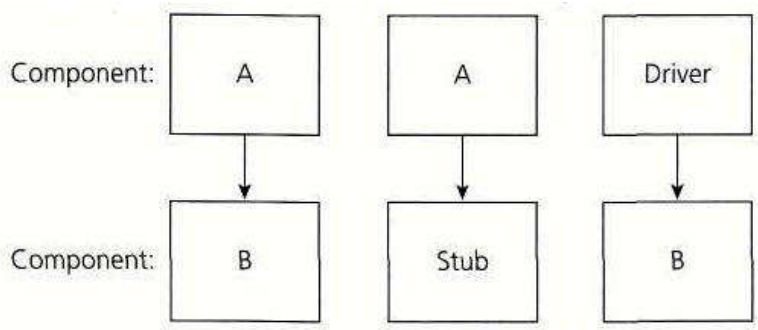
Advantages:

- Working through Pair programming produce well written compact programs which has fewer errors as compared to programmers working alone.
- It reduces total development time of the whole project.
- Customer representative get the idea of updated software products after each iretation. So, it is easy for him to change any requirement if needed.

Disadvantages:

- Due to lack of formal documents, it creates confusion and important decisions taken during different phases can be misinterpreted at any time by different team members.
- Due to absence of proper documentation, when the project completes and the developers are assigned to another project, maintenance of the developed project can become a problem.

B) Software testing is a process, to evaluate the functionality of a software application with an intent to find whether the developed software met the specified requirements or not and to identify the defects to ensure that the product is defect free in order to produce the quality product



The concept of Stubs and Drivers are mostly used in the case of component testing. Component testing may be done in isolation with the rest of the system depending upon the context of the development cycle.

Stubs and drivers are used to replace the missing software and simulate the interface between the software components in a simple manner.

Suppose you have a function (Function A) that calculates the total marks obtained by a student in a particular academic year. Suppose this function derives its values from another function (Function b) which calculates the marks obtained in a particular subject.

You have finished working on Function A and wants to test it. But the problem you face here is that you can't seem to run the Function A without input from Function B; Function B is still under development. In this case, you create a dummy function to act in place of Function B to test your function. This dummy function gets called by another function. Such a dummy is called a Stub.

To understand what a driver is, suppose you have finished Function B and is waiting for Function A to be developed. In this case you create a dummy to call the Function B. This dummy is called the driver.

3. a) Define cohesion and coupling. What is its impact on good design? Discuss about various types of coupling possible with suitable example. [4

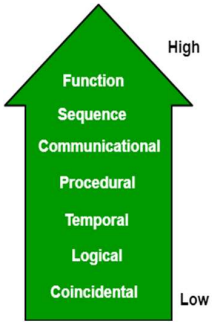
   b) Consider the following code snippet: [4

```
Function( int first[],int second[])
{  //m, q, p are globally declared.
      int c, d, k;
      for ( c = 0 ; c < m ; c++ ){
        for ( d = 0 ; d < q ; d++ ){
  for ( k = 0 ; k < p ; k++ )
 {sum = sum + first[c][k]*second[k][d];}
multiply[c][d] = sum;
sum = 0;}}
```

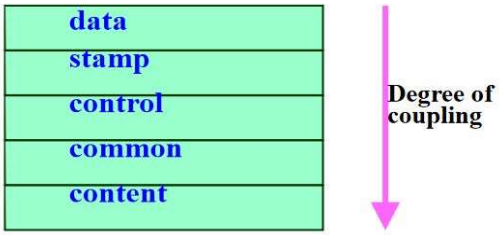
   Draw a CFG. Find out the number of linearly independent paths as well as the test data required to have 100% path coverage.

(Tanisha Bharadwaj-172929089)

Ans: a) **Cohesion** is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. A good software design will have high cohesion.

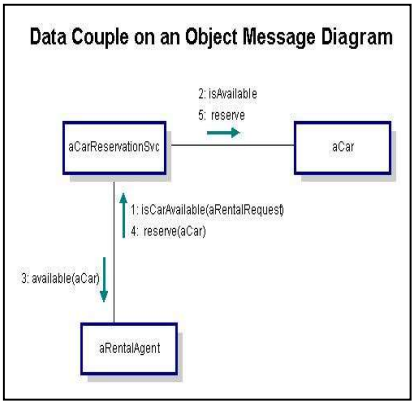


Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling.



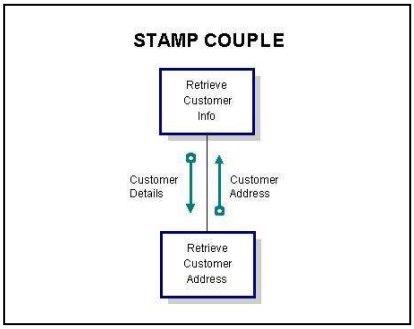
Types of Coupling:

Data Coupling: If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent to each other and communicating through data. Module communications don't contain tramp data.
Example-customer billing system.Or an integer, a float, a character, etc.



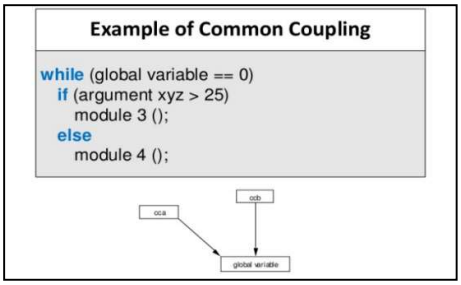
Stamp Coupling In stamp coupling, the complete data structure is passed from one module to another module. Therefore, it involves tramp data. It may be necessary due

to efficiency factors- this choice made by the insightful designer, not a lazy programmer.

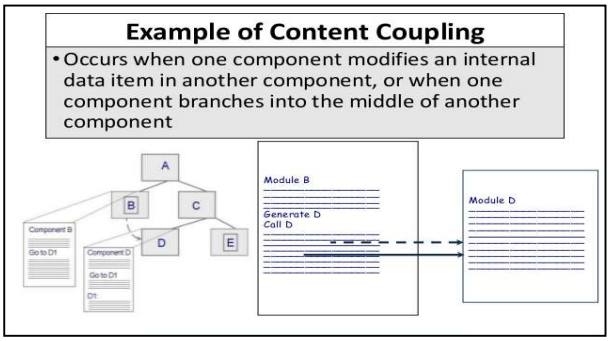


Control Coupling: If the modules communicate by passing control information, then they are said to be control coupled. It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality. Example- sort function that takes comparison function as an argument.

Common Coupling: The modules have shared data such as global data structures.The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change. So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses and reduced maintainability.



Content Coupling: In a content coupling, one module can modify the data of another module or control flow is passed from one module to the other module. This is the worst form of coupling and should be avoided.



7. a) What is the difference between black-box and white-box testing? Discuss about various approaches for test case design in black-box approach. [4

b) What are the different product categories according to Boehm? Explain in detail the COCOMO. [4

(Tuneer Bhattacharya 1729091)

Ans. A)

Black box	White box
It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.	It is a way of testing the software in which the tester has knowledge about the internal structure r the code or the program of the software.
It is mostly done by software testers.	It is mostly done by software developers.
It can be referred as outer or external software testing.	It is the inner or the internal software testing.
This testing can be initiated on the basis of requirement specifications document.	This type of testing of software is started after detail design document.
It is not suitable or preferred for algorithm testing.	It is suitable for algorithm testing.
Example: search something on google by using keywords	**Example:** by input to check and verify loops

Two main approaches to design black box test cases:

- Equivalence class partitioning
- Boundary value analysis

Equivalence partitioning or equivalence class partitioning is a software testing technique that divides the input data of a software unit into partitions of equivalent data from which test cases can be derived. In principle, test cases are designed to cover each partition at least once.

The main idea is that testing the code with any one value belonging to an equivalence class is as good as testing the code with any other value in that class.

The input domain data is divided into different equivalence data classes.It reduce the total number of test cases to a finite set of testable test cases.

Boundary value analysis is a software testing technique in which tests are designed to include representatives of boundary values in a range. The idea comes from the boundary.

The test cases use the values at the boundaries of different equivalence class.

It focuses on the boundary of the input space to identify test cases.The error tend to occur near the extreme values of an input variable.

Ans. B) Boehm postulated that any software development project can be classified into one of the following three categories based on the development complexity: organic, semidetached, and embedded. In order to classify a product into the identified categories, Boehm considered the characteristics of the product and that of the development team and development environment. Boehm's definition of organic, semidetached, and embedded systems are elaborated below.

Organic: A development project can be considered of organic type, if the project deals with developing a well understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar types of projects.

Semidetached: A development project can be considered of semidetached type, if the development consists of a mixture of experienced and inexperienced staff. Team members may have limited experience on related systems but may be unfamiliar with some aspects of the system being developed.

Embedded: A development project is considered to be of embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational procedures exist.

COCOMO (Constructive Cost Estimation Model) was proposed by Boehm.According to Boehm, software cost estimation should be done through three stages: Basic COCOMO, Intermediate COCOMO, and Complete COCOMO.

Basic COCOMO Model:The basic COCOMO model gives an approximate estimate of the project parameters. The basic COCOMO estimation model is given by the following expressions:

Effort = a1 x (KLOC)a 2 PM

Tdev = b1 x (Effort)b 2 Months

Where • KLOC is the estimated size of the software product expressed in Kilo Lines of Code,

• a1, a2, b1, b2 are constants for each category of software products,

• Tdev is the estimated time to develop the software, expressed in months,

• Effort is the total effort required to develop the software product, expressed in person months (PMs).

Estimation of development effort

Organic : Effort = 2.4(KLOC) 1.05 PM

Semi-detached : Effort = 3.0(KLOC) 1.12 PM

Embedded : Effort = 3.6(KLOC) 1.20 PM

Estimation of development time

Organic : Tdev = 2.5(Effort) 0.38 Months

Semi-detached : Tdev = 2.5(Effort) 0.35 Months

Embedded : Tdev = 2.5(Effort) 0.32 Months

Intermediate COCOMO model: The basic COCOMO model assumes that effort and development time are functions of the product size alone.In order to obtain an accurate estimation of the effort and project duration, the effect of all relevant parameters must be taken into account. The intermediate COCOMO model recognizes this fact and refines the initial estimate obtained using the basic COCOMO expressions by using a set of 15 cost drivers (multipliers) based on various attributes of software development.Depending on these ratings,appropriate cost driver values which should be multiplied with the initial estimate obtained using the basic COCOMO. In general, the cost drivers can be classified as being attributes of the following items:

Product: The characteristics of the product that are considered include the inherent complexity of the product, reliability requirements of the product, etc. **Computer**: Characteristics of the computer that are considered include the execution speed required, storage space required etc.

Personnel: The attributes of development personnel that are considered include the experience level of personnel, programming capability, analysis capability, etc.

Development Environment: Development environment attributes capture the development facilities available to the developers.

Complete COCOMO model:A major shortcoming of both the basic and intermediate COCOMO models is that they consider a software product as a single homogeneous entity. However, most large systems are made up several smaller sub-systems. These subsystems may have widely different characteristics.The complete COCOMO model considers these differences in characteristics of the subsystems and estimates the effort and development time as the sum of the estimates for the individual subsystems. The cost of each subsystem is estimated separately. This approach reduces the margin of error in the final estimate.

8. Write short notes (any two) [4 × 2

a) Software Configuration Management

b) Integration Testing

c) Reverse Engineering

d) Software Reliability

(Muskan Mohanty 1729139)

c) Reverse Engineering:-

Software Reverse Engineering is a process of recovering the design, requirement specifications and functions of a product from an analysis of its code. It builds a program database and generates information from this.

The purpose of reverse engineering is to facilitate the maintenance work by improving the understandability of a system and to produce the necessary documents for a legacy system.

Reverse Engineering Goals:

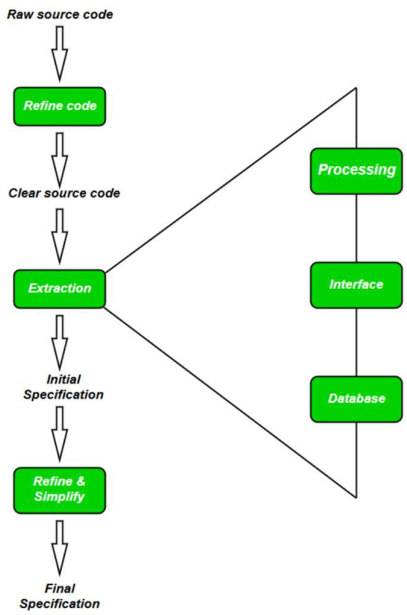
Cope with Complexity.

Recover lost information.

Detect side effects.

Synthesise higher abstraction.

Facilitate Reuse.



d) Software Reliability:-

Software reliability is defined as: The probability of failure-free software operation for a specified period of time in a specified environment.

• It can be an important factor affecting system reliability.

• Different from hardware reliability since it doesn't age, wear out, rust, deform or crack!

• Software usually stays in the same condition as when it was created and unless there are changes caused by hardware - like changes in the storage content or data path – we could assume software doesn't really "break".

• Software Reliability is an important to attribute of software quality, together with functionality, usability, performance, serviceability, capability, installability, maintainability, and documentation.

• Software Reliability is hard to achieve, as the complexity of software tends to be high:

– While any system with a high degree of complexity, including software, will be hard to reach a certain level of reliability, system developers tend to push complexity into the software layer, with the rapid growth of system size and ease of doing so by upgrading the software.

– For example

• Next-generation aircraft will have over one million source lines of software on-board; next-generation air traffic control systems will contain between one and two million lines; the upcoming international Space Station will have over two million lines on-board and over ten million lines of ground support software; several major life-critical defence systems will have over five million source lines of software.

2014

1. a) What are the solutions to the software crisis? Explain. [2 × 10
   b) What is function point metric? What factors we consider for this metric?
   c) Differentiate between architectural design and detailed design.
   d) Explain the different categories of users who use SRS.
   e) What is 99% complete syndrome? Explain.
   f) Explain any two debugging approaches.
   g) Differentiate between code walk-through and code inspection.
   h) Why spiral model is known as Meta model?
   i) Differentiate between alpha and beta testing.
   j) What is the purpose of using UML class diagram? Give example with the help of a diagram.

[Nibedita Misra 1729141]

a) Solutions of Software Crisis are as follows:

- Reduction in software over-budget.
- The quality of software must be high.
- Less time needed for software project.
- Experience working team member on software project.
- Software must be delivered.

b) Function Point (FP)is an element of software development which helps to approximate the cost of development early in the process. It may measures functionality from user's point of view.Factors affecting it are as follows :

- External Inputs (EI)
- External Outputs (EO)
- External Inquiry (EQ)
- Internal Logical Files (ILF's)
- External Interface Files (EIF's)

c) Architecture design is more abstract than detailed design and it specifies the fundamental structure and patterns of the system under development whereas Detailed design focuses on all of the implementation details necessary to implement the architecture that is specified.

d) Different categories of user of SRS documents are:-

1. user, customer & marketing personnel
2. s/w developers
3. User documentation writers
4. Test engineers
5. project managers
6. Maintenance engineers

e) 99% complete syndrome is the syndrome in which the last 1 percent of anything takes longer than the other 99 percent. Most oftenly,this percent drags on to days, weeks, and even few months thereby delaying inspections and prohibiting our move.

It happens when we do not follow SDLC (Software Development Life Cycle)in the software development .

f) Cause Elimination Method:

In this approach, a listing of causes that may presumably have contributed to the error symptom is developed and tests are conducted to eliminate every. A connected technique of identification of the error from the error symptom is that the package fault tree analysis.

Program Slicing:

In this technique the search house is reduced by process slices. A slice of a program for a specific variable at a particular statement is that the set of supply lines preceding this statement which will influence the worth of that variable.

g) Code Walkthrough is a form of peer review in which a programmer leads the review process and the other team members ask questions and spot possible errors against development standards and other issues whereas

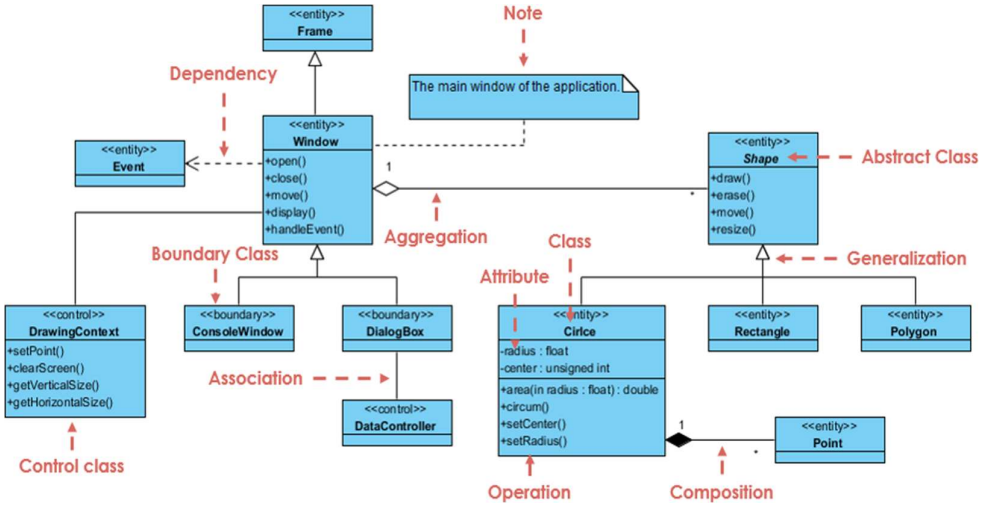
Code Inspection is the most formal type of review, which is a kind of static testing to avoid the defect multiplication at a later stage.

h) The Spiral model is called as a Meta Model because it subsumes all the other SDLC models. For example, a single loop spiral actually represents the Iterative Waterfall Model. The spiral model incorporates the stepwise approach of the Classical Waterfall Model. The spiral model uses the approach of Prototyping Model by building a prototype at the start of each phase as a risk handling technique. Also, the spiral model can be considered as supporting the evolutionary model – the iterations along the spiral can be considered as evolutionary levels through which the complete system is built.

i) Alpha testing is a type of acceptance testing; performed to identify all possible issues/bugs before releasing the product to everyday users or the public whereas Beta Testing of a product is performed by "real users" of the software application in a "real environment" and can be considered as a form of external User Acceptance Testing.

Beta version of the software is released to a limited number of end-users of the product to obtain feedback on the product quality.

j) In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.



3. a) What is LOC? How we estimate LOC of a software product? List three shortcomings of LOC. [5

b) List down at least six responsibilities of a software manager. [3

(Nilanjan Roy 1729143))

Ans. A)Source lines of code (SLOC), also known as lines of code (**LOC**), is a **software** metric used to measure the size of a computer program by counting the number of lines in the text of the program's source code.

Lines of Code (LOC): As the name suggest, LOC count the total number of lines of source code in a project. The units of LOC are:

- KLOC- Thousand lines of code
- NLOC- Non comment lines of code
- KDSI- Thousands of delivered source instruction

The size is estimated by comparing it with the existing systems of same kind. The experts use it to predict the required size of various components of software and then add them to get the total size.

Advantages:

- Universally accepted and is used in many models like COCOMO.
- Estimation is closer to developer's perspective.
- Simple to use.

Disadvantages or Shortcoming:

- Different programming languages contains different number of lines.
- No proper industry standard exist for this technique.
- It is difficult to estimate the size using this technique in early stages of project.

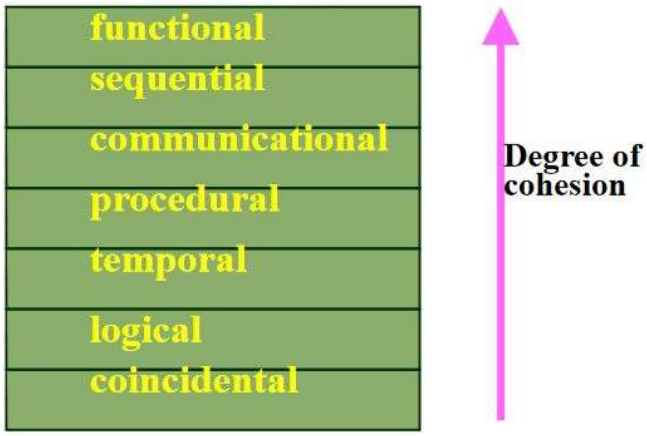
(b) The six responsibilities of a software manager are:-

1. **Project Estimation:** Project Size Estimation is the most important parameter based on which all other estimations like cost, duration and effort are made.
    - **Cost Estimation:** Total expenses to develop the software product is estimated.
    - **Time Estimation:** The total time required to complete the project.
    - **Effort Estimation:** The effort needed to complete the project is estimated.
2. **Scheduling:** After completion of estimation of all the project parameters, scheduling for manpower and other resources are done.
3. **Staffing:** Team structure and staffing plans are made.
4. **Risk Management:** The project manager should identify the unanticipated risks that may occur during project development risk, analysis the damage might cause these risks and take risk reduction plan to cope up with these risks.
5. **Miscellaneous plans:** This includes making several other plans such as quality assurance plan, configuration management plan, etc.
6. **Tracking the progress:** The project manager should keep an eye on the progress of the project. A project manager must track whether the project is going as per plan or not. If any problem arises, then take necessary action to solve the problem.

6. What is cohesion and coupling? Discuss about various types of cohesion and coupling with suitable examples. [2+6

(sagnik sarbadhikari 1729151)

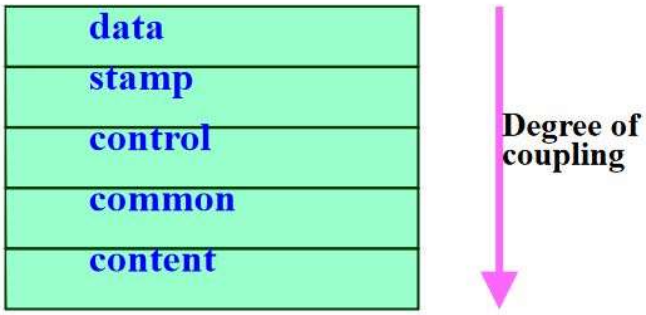
Ans. **Cohesion** is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. A good software design will have high cohesion.



Types of Cohesion:

- **Functional Cohesion:** Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation.

- **Sequential Cohesion:** An element outputs some data that becomes the input for other element, i.e., data flow between the parts. It occurs naturally in functional programming languages.
- **Communicational Cohesion:** Two elements operate on the same input data or contribute towards the same output data. Example- update record int the database and send it to the printer.
- **Procedural Cohesion:** Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable. Ex- calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA.
- **Temporal Cohesion:** The elements are related by their timing involved. A module connected with temporal cohesion all the tasks must be executed in the same time-span. This cohesion contains the code for initializing all the parts of the system. Lots of different activities occur, all at init time.
- **Logical Cohesion:** The elements are logically related and not functionally. Ex- A component reads inputs from tape, disk, and network. All the code for these functions is in the same component. Operations are related, but the functions are significantly different.
- **Coincidental Cohesion:** The elements are not related(unrelated). The elements have no conceptual relationship other than location in source code. It is accidental and the worst form of cohesion. Ex- print next line and reverse the characters of a string in a single component.



Types of Coupling:

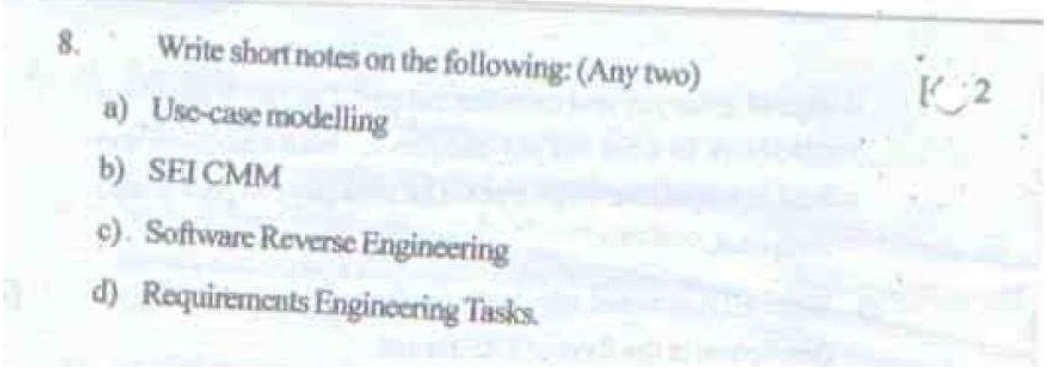
Data Coupling: If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent to each other and communicating through data. Module communications don't contain tramp data.
Example-customer billing system.Or an integer, a float, a character, etc.

Stamp Coupling In stamp coupling, the complete data structure is passed from one module to another module. Therefore, it involves tramp data. It may be necessary due to efficiency factors- this choice made by the insightful designer, not a lazy programmer.

Control Coupling: If the modules communicate by passing control information, then they are said to be control coupled. It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality. Example- sort function that takes comparison function as an argument.

Common Coupling: The modules have shared data such as global data structures.The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change. So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses and reduced maintainability.

Content Coupling: In a content coupling, one module can modify the data of another module or control flow is passed from one module to the other module. This is the worst form of coupling and should be avoided.


8. Write short notes on the following: (Any two) [2

a) Use-case modelling

b) SEI CMM

c) Software Reverse Engineering

d) Requirements Engineering Tasks.

[Sagnik Misra 1729152]

a) A use-case model is a model of how different types of users interact with the system to solve a problem. As such, it describes the goals of the users, the interactions between the users and the system, and the required behavior of the system in satisfying these goals.

A use-case model consists of a number of model elements.The most important model elements are: use cases, actors and the relationships between them.

A use-case diagram is used to graphically depict a subset of the model to simplify communications.There will typically be several use-case diagrams associated with a given model, each showing a subset of the model elements relevant for a particular purpose.The same model element may be shown on several use-case diagrams, but each instance must be consistent. If tools are used to maintain the use-case model, this consistency constraint is automated so that any changes to the model element (changing the name for example) will be automatically reflected on every use-case diagram that shows that element.

b) CMM (Capability Maturity Model) was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University in 1987.

- It is not a software process model. It is a framework which is used to analyse the approach and techniques followed by any organization to develop a software product.
- It also provides guidelines to further enhance the maturity of those software products.
- It is based on profound feedback and development practices adopted by the most successful organizations worldwide.

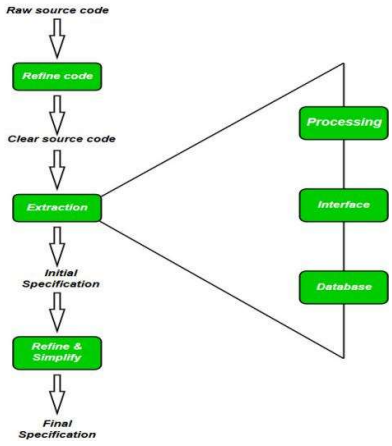
- This model describes a strategy that should be followed by moving through 5 different levels.
- Each level of maturity shows a process capability level. All the levels except level-1 are further described by Key Process Areas (KPA's).

c) Software Reverse Engineering is a process of recovering the design, requirement specifications and functions of a product from an analysis of its code. It builds a program database and generates information from this.

The purpose of reverse engineering is to facilitate the maintenance work by improving the understandability of a system and to produce the necessary documents for a legacy system.

Goals of software reverse engineering:

- Cope with Complexity.
- Recover lost information.
- Detect side effects.
- Synthesise higher abstraction.
- Facilitate Reuse.



d) The process of collecting the software requirement from the client then understand, evaluate and document it is called as requirement engineering. Requirement engineering constructs a bridge for design and construction.

Requirement engineering consists of seven different tasks as follows:

1. Inception

Inception is a task where the requirement engineering asks a set of questions to establish a software process.

2. Elicitation

Elicitation means to find the requirements from anybody.

3. Elaboration

In this task, the information taken from user during inception and elaboration and are expanded and refined in elaboration.

4. Negotiation

In negotiation task, a software engineer decides the how will the project be achieved with limited business resources.

5. Specification

In this task, the requirement engineer constructs a final work product.

6. Validation

The work product is built as an output of the requirement engineering and that is accessed for the quality through a validation step.

7. Requirement management

It is a set of activities that help the project team to identify, control and track the requirements and changes can be made to the requirements at any time of the ongoing project.

2016

1. Answer all parts of the following question. [2 × 10

(a) Which life cycle model would you follow for developing software for the following application:
A software that would function as the controller of a telephone switching system.
Give the justification for your answer.

(b) Name the activities those are performed in project planning.

(c) What do you mean by a person-month?

(d) What is a critical path in an activity network?

(e) What is the purpose of requirements analysis activity?

(f) What do you mean by balancing a DFD?

(g) What do you mean by beta testing?

(h) Explain how cyclomatic complexity is computed.

(i) What do you mean by adaptive maintenance? Briefly explain with an example.

(j) What do you mean by a POFOD of 0.001?

(Sayahna Sengupta 1729158)

(b) In order to carry out effective project planning activities the following four steps should be considered and can be performed effectively within a project management such as

- Define the activities that make up the project
- Define the relationships and dependencies between activities
- Estimate the resources required
- Estimate activity durations

(c) A “person month” is the metric for expressing the effort (amount of time) principal investigators (PIs), faculty and other senior personnel devote to a specific project. The effort is based on the type of appointment of the individual with the organization; e.g., calendar year (CY), academic year (AY), and/or summer term (SM); and the organization’s definition of such. For instance, some institutions define the academic year as a 9-month appointment while others define it as a 10-month appointment.
(d) In project management, a **critical path** is the sequence of project **network activities** which add up to the longest overall duration, regardless if that longest duration has float or not. This determines the shortest time possible to complete the project.

(e) Requirements Analysis is the process of defining the expectations of the users for an application that is to be built or modified. Requirements analysis involves all the tasks that are conducted to identify the needs of different stakeholders. Therefore requirements analysis means to analyze, document, validate and manage software or system requirements. High-quality requirements are documented, actionable, measurable, testable, traceable, helps to identify business opportunities, and are defined to a facilitate system design.

(f). The concept of balancing states that all the incoming flows to a process and all the outgoing flows from a process in the parent diagram should be preserved at the next level of decomposition.

Process decomposition lets you organize your overall DFD in a series of levels so that each level provides successively more detail about a portion of the level above it.

The goal of the balancing feature is to check your system internal consistency, which is particularly useful as different levels of expertise are generally involved in a project.

(g). In software development, a beta test is the second phase of software testing in which a sampling of the intended audience tries the product out.Beta is the second letter of the Greek alphabet. Originally, the term *alpha test* meant the first phase of testing in a software development process. The first phase includes unit testing, component testing, and system testing. Beta testing can be considered "pre-release testing."

Beta testing is also sometimes referred to as user acceptance testing (UAT) or end user testing. In this phase of software development, applications are subjected to real world testing by the intended audience for the software. The experiences of the early users are forwarded back to the developers who make final changes before releasing the software commercially.

(h).Cyclomatic complexity is computed using the control flow graph of the program: the nodes of the graph correspond to indivisible groups of commands of a program, and a directed edge connects two nodes if the second command might be executed immediately after the first command.

(i). Adaptive maintenance is the implementation of changes in a part of the system, which has been affected by a change that occurred in some other part of the system. Modification of a software product performed after delivery to keep a software product

usable in a changed or changing environment. Adaptive maintenance consists of adapting software to changes in the environment such as the hardware or the operating system. The term environment in this context refers to the conditions and the influences which act (from outside) on the system. For example, business rules, work patterns, and government policies have a significant impact on the software system.

(j). A **POFOD of 0.001** means that 1 out of a thousand service requests may result in failure. ... This metric is sometimes called the failure intensity. MTTF **Mean** time to failure The average time between observed system failures. An MTTF of 500 means that 1 failure **can** be expected every 500 time units.

2. (a) How do cohesion and coupling affects software design? Discuss about various types of cohesion and coupling. [4

(b) Explain the working of RAD model. What are its applicability? [4

(Swadesh Nayak 1729085)

Ans: a) Cohesion is the measure of degree to which elelment of the modules are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Where as coupling is the measure of degree to which different module are interdepent. A good software does have less coupling.

Types of Cohesion:

Functional Cohesion: Every essential element for a single computation is contained in the component.
Sequential Cohesion: An element outputs some data that becomes the input for other elemen.
Communicational Cohesion: Two elements operate on the same input data or contribute towards the same output data.
Procedural Cohesion: Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable. Ex- calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA.
Temporal Cohesion: The elements are related by their timing involved. A module connected with temporal cohesion all the tasks must be executed in the same time-span. This cohesion contains the code for initializing all the parts of the system. Lots of different activities occur, all at init time.
Logical Cohesion: The elements are logically related and not functionally.
Coincidental Cohesion: The elements are not related(unrelated). The elements have no conceptual relationship other than location in source code. It is accidental and the worst form of cohesion.

Types of Coupling:

Data Coupling: If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent to each other and communicating

through data. Module communications don't contain tramp data. Example-customer billing system.

Stamp Coupling In stamp coupling, the complete data structure is passed from one module to another module. Therefore, it involves tramp data. It may be necessary due to efficiency factors- this choice made by the insightful designer, not a lazy programmer.
Control Coupling: If the modules communicate by passing control information, then they are said to be control coupled. It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality. Example- sort function that takes comparison function as an argument.
External Coupling: In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware. Ex- protocol, external file, device format, etc.
Common Coupling: The modules have shared data such as global data structures.The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change. So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses and reduced maintainability.
Content Coupling: In a content coupling, one module can modify the data of another module or control flow is passed from one module to the other module. This is the worst form of coupling and should be avoided.

B) RAD or **R**apid **A**pplication **D**evelopment process is derived from the waterfall model.It targets at developing software in a short span of time.

Software Development Life Cycle follows:

Business Modelling

Data Modelling

Application Generation

Test and Turnover

It focuses on input-output source and destination of the information. It emphasizes on delivering

projects in small pieces; the larger projects are divided into a series of smaller projects. The main features of RAD model are that it focuses on the reuse of templates, tools, processes, and code.

Application:

- This model should be used for a system with known requirements and requiring short development time.
- It is also suitable for projects where requirements can be modularized and reusable components are also available for development.
- The model can also be used when already existing system components can be used in developing a new system with minimum changes.

3. (a) What is function point metric? What are the steps for calculating function point metric? Compute the function point value for a project with the following information domain characteristics. [4

Number of user inputs=24

Number of user outputs=16

Number of inquiries=22

Number of files=4

Number of external interfaces=2

Assume, the complexity weighting factor is average and the various complexity weighting factors are 4, 2, 0, 4, 3, 4, 5, 3, 5, 5, 4, 3, 5, 5.

(b) With a suitable example, briefly explain project estimation using COCOMO. [4

(Tanisha Bhardwaj 1729089)

Ans. A) Function points are one of the most widely used measures of software size. The basis of function points is that the "functionality " of the system that is; what the system performs, is the measure of the system size. In function points, the system functionally is calculated in terms of the number of function it implements, the number of inputs, the number of output etc. Parameter that can be obtained after requirements analysis and that are independent of the specification (and implementation) language. Steps for calculating function point metric are:-

Step-1:

F = 14 * scale

Scale varies from 0 to 5 according to character of Complexity Adjustment Factor (CAF). Below table shows scale:

a) 0 - No Influence

b) 1 - Incidental

c) 2 - Moderate

d) 3 - Average

e) 4 - Significant

f) 5 - Essential

g) *Step-2:* Calculate Complexity Adjustment Factor (CAF).

CAF = 0.65 + (0.01 * F)

h) *Step-3:* Calculate Unadjusted Function Point (UFP).

TABLE (Required)

Function Units	Low	Avg	High
EI	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10

Multiply each individual function point to corresponding values in TABLE.

i) *Step-4:* Calculate Function Point.

```
FP = UFP * CAF
```

According to question,
Given the following values, compute function point when all complexity adjustment factor (CAF) and weighting factors are average.

```
User Input = 24
User Output = 16
User Inquiries = 22
User Files = 4
External Interface = 2
```

Explanation:

- *Step-1:* As complexity adjustment factor is average (given in question), hence,
- scale = 3.

```
F = 14 * 3 = 42
```

- *Step-2:*

```
CAF = 0.65 + ( 0.01 * 42 ) = 1.07
```

- *Step-3:* As weighting factors are also average (given in question) hence we will multiply each individual function point to corresponding values in TABLE.

```
UFP = (24*4) + (16*5) + (22*4) + (4*10) + (2*7) = 318
```

- *Step-4:*

```
Function Point =318 * 1.07 = 340.26
```

B)Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e **number of Lines of Code**. It is a procedural cost estimate model for software

projects and often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time and quality. It was proposed by Barry Boehm in 1970 and is based on the study of 63 projects, which make it one of the best-documented models.

The key parameters which define the quality of any software products, which are also an outcome of the Cocomo are primarily Effort & Schedule:

a) **Effort:** Amount of labor that will be required to complete a task. It is measured in person-months units.

b) **Schedule:** Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put. It is measured in the units of time such as weeks, months.

Different models of Cocomo have been proposed to predict the cost estimation at different levels, based on the amount of accuracy and correctness required. All of these models can be applied to a variety of projects, whose characteristics determine the value of constant to be used in subsequent calculations.

COCOMO, projects are categorized into three types:

j) Organic

k) Semidetached

l) Embedded

1.Organic: A development project can be treated of the organic type, if the project deals with developing a well-understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar methods of projects. **Examples of this type of projects are simple business systems, simple inventory management systems, and data processing systems.**

2. Semidetached: A development project can be treated with semidetached type if the development consists of a mixture of experienced and inexperienced staff. Team members may have finite experience in related systems but may be unfamiliar with some aspects of the order being developed. **Example of Semidetached system includes developing a new operating system (OS), a Database Management System (DBMS), and complex inventory management system.**

3. Embedded: A development project is treated to be of an embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational method exist. **For Example:** ATM, Air Traffic control.

For three product categories, Bohem provides a different set of expression to predict effort (in a unit of person month)and development time from the size of estimation in KLOC(Kilo Line of code) efforts estimation takes into account the productivity loss due to holidays, weekly off, coffee breaks, etc.

According to Boehm, software cost estimation should be done through three stages:

- Basic Model
- Intermediate Model
- Detailed Model

1. Basic COCOMO Model: The basic COCOMO model provide an accurate size of the project parameters. The following expressions give the basic COCOMO estimation model:

$$\textbf{Effort}=a_1*(\textbf{KLOC})\ a_2\ \textbf{PM}$$
$$\textbf{Tdev}=b_1*(\textbf{efforts})b_2\ \textbf{Months}$$

Where

KLOC is the estimated size of the software product indicate in Kilo Lines of Code,

a_1,a_2,b_1,b_2 are constants for each group of software products,

Tdev is the estimated time to develop the software, expressed in months,

Effort is the total effort required to develop the software product, expressed in **person months (PMs)**.

Estimation of development effort

For the three classes of software products, the formulas for estimating the effort based on the code size are shown below:

Organic: Effort = 2.4(KLOC) 1.05 PM

Semi-detached: Effort = 3.0(KLOC) 1.12 PM

Embedded: Effort = 3.6(KLOC) 1.20 PM

Estimation of development time

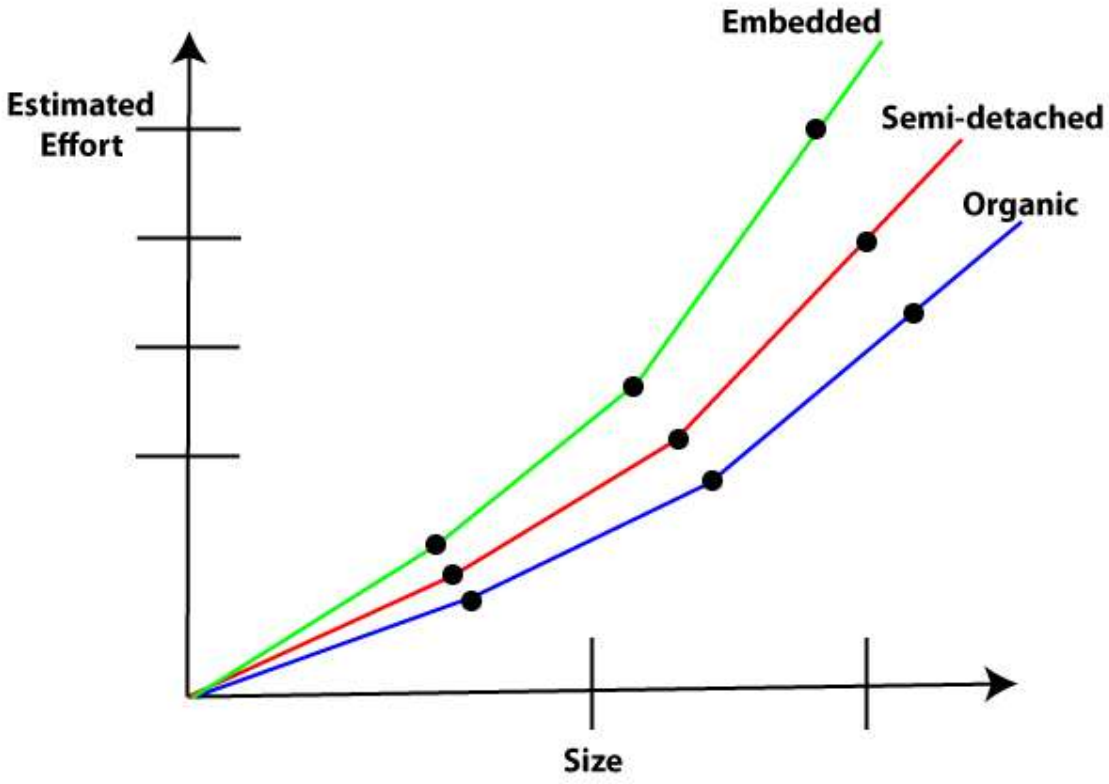
For the three classes of software products, the formulas for estimating the development time based on the effort are given below:

Organic: Tdev = 2.5(Effort) 0.38 Months

Semi-detached: Tdev = 2.5(Effort) 0.35 Months

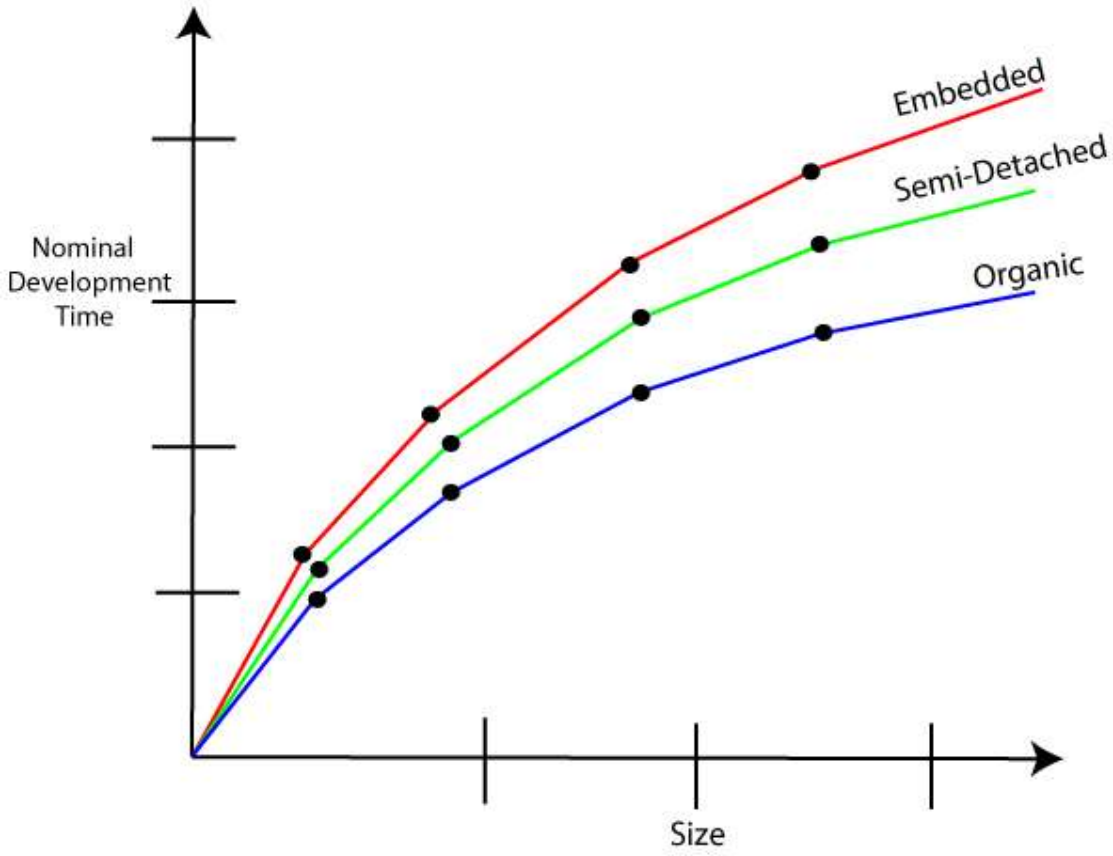
Embedded: Tdev = 2.5(Effort) 0.32 Months

Some insight into the basic COCOMO model can be obtained by plotting the estimated characteristics for different software sizes. Fig shows a plot of estimated effort versus product size. From fig, we can observe that the effort is somewhat superliner in the size of the software product. Thus, the effort required to develop a product increases very rapidly with project size.



Effort versus product size

The development time versus the product size in KLOC is plotted in fig. From fig it can be observed that the development time is a sub linear function of the size of the product, i.e. when the size of the product increases by two times, the time to develop the product does not double but rises moderately. This can be explained by the fact that for larger products, a larger number of activities which can be carried out concurrently can be identified. The parallel activities can be carried out simultaneously by the engineers. This reduces the time to complete the project. Further, from fig, it can be observed that the development time is roughly the same for all three categories of products. For example, a 60 KLOC program can be developed in approximately 18 months, regardless of whether it is of organic, semidetached, or embedded type.



Development time versus size

From the effort estimation, the project cost can be obtained by multiplying the required effort by the manpower cost per month. But, implicit in this project cost computation is the assumption that the entire project cost is incurred on account of the manpower cost alone. In addition to manpower cost, a project would incur costs due to hardware and software required for the project and the company overheads for administration, office space, etc.

It is important to note that the effort and the duration estimations obtained using the COCOMO model are called a nominal effort estimate and nominal duration estimate. The term nominal implies that if anyone tries to complete the project in a time shorter than the estimated duration, then the cost will increase drastically. But, if anyone completes the project over a longer period of time than the estimated, then there is almost no decrease in the estimated cost value.

Example1: Suppose a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three model i.e., organic, semi-detached & embedded.

Solution: The basic COCOMO equation takes the form:

$Effort=a_1*(KLOC)\ a_2\ PM$
$Tdev=b_1*(efforts)b_2\ Months$
Estimated Size of project= 400 KLOC

(i)Organic Mode

E = 2.4 * (400)1.05 = 1295.31 PM
D = 2.5 * (1295.31)0.38=38.07 PM

(ii)Semidetached Mode

E = 3.0 * (400)1.12=2462.79 PM
D = 2.5 * (2462.79)0.35=38.45 PM

(iii) Embedded Mode

E = 3.6 * (400)1.20 = 4772.81 PM
D = 2.5 * (4772.8)0.32 = 38 PM

Example2: A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the Effort, development time, average staff size, and productivity of the project.

Solution: The semidetached mode is the most appropriate mode, keeping in view the size, schedule and experience of development time.

Hence E=3.0(200)1.12=1133.12PM
D=2.5(1133.12)0.35=29.3PM

$$\text{Average Staff Size (SS)} = \frac{E}{D} \text{ Persons}$$

$$= \frac{1133.12}{29.3} = 38.67 \text{ Persons}$$

$$\text{Productivity} = \frac{KLOC}{E} = \frac{200}{1133.12} = 0.1765 \text{ KLOC/PM}$$

P = 176 LOC/PM

2. **Intermediate Model:** The basic Cocomo model considers that the effort is only a function of the number of lines of code and some constants calculated according to the various software systems. The intermediate COCOMO model recognizes these facts and refines the initial estimates obtained through the basic COCOMO model by using a set of 15 cost drivers based on various attributes of software engineering.

4. (a) Identify the functional requirements for the following software. List out the input, output and constraints for each function. [4

Supermarket Automation Software (SAS): The manager of a supermarket wants us to develop an automation software. The supermarket stocks a set of items. Customers pick up their desired items from the different counters in required quantities. The customers present these items to the sales clerk. The sales clerk passes the items over a bar code reader and an automatic weighing scale. The data regarding the item type and the quantity get registered.

- SAS should at the end of a sales transaction print the bill containing the serial number of the sales transaction, the name of the item, code number, quantity, unit price, and item price. The bill should indicate the total amount payable.
- SAS should maintain the inventory of the various items of the supermarket. The manager upon query should be able to see the inventory details. In order to support inventory management, the inventory of an item should be decreased whenever an item is sold. SAS should also support an option by which an employee can update the inventory when new supply arrives.
- SAS should support printing the sales statistics for every item the supermarket deals with for any particular day or any particular period. The sales statistics should indicate the quantity of an item sold, the price realized, and the profit.
- The manager of the supermarket should be able to change the price at which an item is sold as the prices of the different items vary on a day-to-day basis.

(b) Draw the Context diagram and Level 1 Data Flow Diagram [4

(DFD) for the above Supermarket Automation Software (SAS) given in Part (a).

(Sayahna Sengupta 1729158)

This software is designed to automatethe billing and inventory system in a supermarket. Unless otherwise stated, allrequirements specified here are high priority and committed for release □.□.

The Supermarket automation software consists of the following major functions:
1.Maintaining and updating the inventory of the various commodities of thesupermarket.
2.Creating and printing sales transaction bills.
3.Displaying and printing the sales statistics of various commodities for anyparticular product.

PRODUCT PERSPECTIVE

FIGURE 1

The supermarket automation system is a new system that replaces the current manualprocesses of billing and inventory management in a supermarket. The context diagram infigure illustrates the external entities and system interfaces for release.The system is expected to evolve over several releases.

PRODUCT FUNCTIONS

The set of functionalities that are supported by the system are documented below -

Performance Sales Transactions

Whenever any item is sold from the stock of the supermarket, this function prompts the clerk to pass the item over a bar code reader and an automatic weighing scale, the data regarding the item type and the quantity get automatically registered then. During the sales transaction, the name of the item, code number, quantity, unit price, and item price are entered into the bill. The bill indicates the total amount payable. The inventory is then suitably updated.

Read Bar code

)input: Sold items are passed over the reader.

Processing: Bar code of the item is read and the sold item is registered automatically.

Weigh

)Input: Sold items are weighed over the automatic weighing scale.

Processing: Weight of the sold item is automatically registered.

Generation of Bill

A transaction bill containing the serial number of the sales transaction, the name of the items, quantity, unit price, item price and the total amount payable after adding the taxes is printed.

Update inventory

)n order to support inventory management, this function updates the inventory whenever an item is sold. Again, when there is a new supply arrival, an employee updates the inventory level by this function.

Check inventory

The manager upon invoking this function issues query to view the inventory details.

Update prices

The manager changes the price of an item by exercising this option.

FUNCTIONAL REQUIREMENTS

A sale transaction both authorizes and settles the requested amount against the paymentmethod indicated. Through authorizing, the Transaction request confirms that thepayment method exists and that funds are available at the time of Authorization to coverthe transaction amount.

Inputs

Products')Ds from the bar code reader.
Weight reading from the automatic weighing scale.

Processing

The SAS queries the database for the product information and calculates the totalamount payable after inclusion of taxes.

A bill is created in a printable format.

Outputs

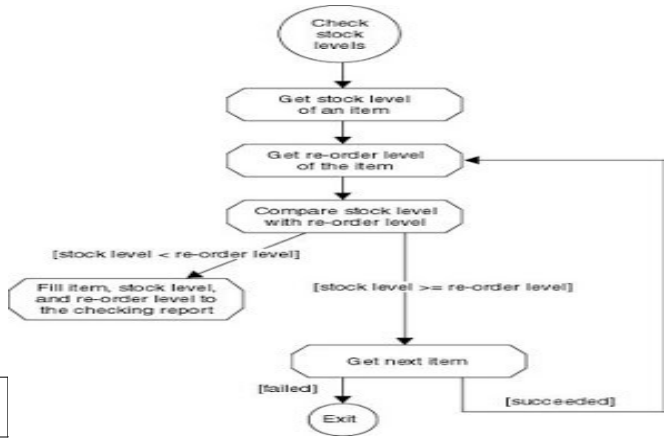
A formatted bill is printed for the customer.

Error handling

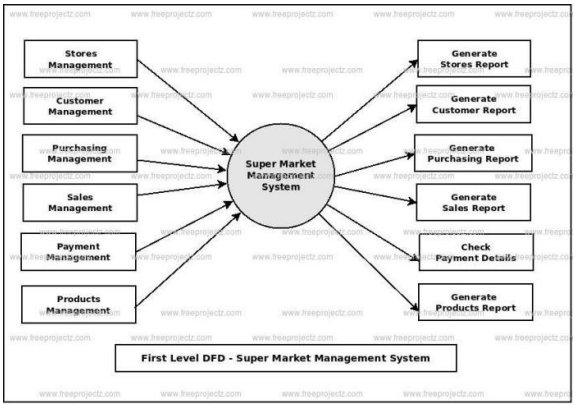
The SAS may not be able to connect to the server due to error in networkconnection, in the case of which transaction is not possible.

The manager views the sales statistics and prints them in various formats such aspie charts, bar graphs, tabular format, etc.

CONTEXT DIAGRAM:



DATA FLOW DIAGRAM:



5.

(b) What do you mean by integration testing? Briefly explain the different types of integration testing techniques along with their advantages and disadvantages. [4

(Nibedita Misra 1729141)

Integration testing is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing.

Some different types of integration testing are big-bang, mixed (sandwich), risky-hardest, top-down, and bottom-up. Other Integration Patterns[3] are: collaboration integration, backbone integration, layer integration, client-server integration, distributed services integration and high-frequency integration.

In the big-bang approach, most of the developed modules are coupled together to form a complete software system or major part of the system and then used for integration testing. This method is very effective for saving time in the integration testing process. However, if the test cases and their results are not recorded properly, the entire integration process will be more complicated and may prevent the testing team from achieving the goal of integration testing.

Bottom-up testing is an approach to integrated testing where the lowest level components are tested first, then used to facilitate the testing of higher level components. The process is repeated until the component at the top of the hierarchy is tested. All the bottom or low-level modules, procedures or functions are integrated and then tested. After the integration testing of lower level integrated modules, the next level of modules will be formed and can be used for integration testing. This approach is helpful only when all or most of the modules of the same development level are ready. This method also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage.

Top-down testing is an approach to integrated testing where the top integrated modules are tested and the branch of the module is tested step by step until the end of the related module.

Sandwich testing is an approach to combine top down testing with bottom up testing.One limitation to this sort of testing is that any conditions not stated in specified integration tests, outside of the confirmation of the execution of design items, will generally not be tested.

6. (a) Briefly explain about various types of code review techniques along with their advantages and disadvantages. [4

(Sagnik Sarbadhikari 1729151)

Code review is systematic examination (often as peer review) of computer source code. It is intended to find and fix mistakes overlooked in the initial development phase, improving both the overall quality of software and the developers' skills. Reviews are done in various forms such as pair programming,

Code reviews can often find and remove common vulnerabilities such as format string exploits, race conditions, memory leaks and buffer overflows, thereby improving software security. Online software repositories based on Subversion (with Redmine or Trac), Mercurial, Git or others allow groups of individuals to collaboratively review code. Additionally, specific tools for collaborative code review can facilitate the code review process.

Automated code reviewing software lessens the task of reviewing large chunks of code on the developer by systematically checking source code for known vulnerabilities.

Capers Jones' ongoing analysis of over 12,000 software development projects showed that the latent defect discovery rate of formal inspection is in the 60-65% range. For informal inspection, the figure is less than 50%.[*citation needed*] The latent defect discovery rate for most forms of testing is about 30%. [2]

Typical code review rates are about 150 lines of code per hour. Inspecting and reviewing more than a few hundred lines of code per hour for critical software (such as safety critical embedded software) may be too fast to find errors. [3] Industry data indicate that code review can accomplish at most an 85% defect removal rate with an average rate of about 65%. [4]

Types

Code review practices fall into three main categories: pair programming, formal code review and lightweight code review.[1]

Formal code review, such as a Fagan inspection, involves a careful and detailed process with multiple participants and multiple phases. Formal code reviews are the traditional method of review, in which software developers attend a series of meetings and review code line by line, usually using printed copies of the material. Formal inspections are extremely thorough and have been proven effective at finding defects in the code under review.

Lightweight code review typically requires less overhead than formal code inspections, though it can be equally effective when done properly.[*citation needed*] Lightweight reviews are often conducted as part of the normal development process:

- Over-the-shoulder – One developer looks over the author's shoulder as the latter walks through the code.
- Email pass-around – Source code management system emails code to reviewers automatically after checkin is made.
- Pair Programming – Two authors develop code together at the same workstation, such is common in Extreme Programming.
- Tool-assisted code review – Authors and reviewers use specialized tools designed for peer code review.

ADVANTAGES:

It is a very good way to improve your code's quality by code review, and helpful to find bugs by other software engineers, which easy to be ignored by yourself . But the disadvantage of code review is that it may takes a lot time costs and energy.

DISADVANTAGES:

1. TIME RESOIRCES

2. COMPLICATED

3. LACK OF VISIBILITY

7. (a) With suitable examples, briefly explain the various reliability metrics of software products? [4

(b) What do you mean by software re-engineering? With a suitable diagram, briefly explain the process of software reengineering along with its advantages and disadvantages. [4

(Muskan Mohanty 1729139)

Ans.

A) Reliability metrics are used to quantitatively expressed the reliability of the software product. The option of which metric is to be used depends upon the type of system to which it applies & the requirements of the application domain.

1. Mean Time to Failure (MTTF)

MTTF is described as the time interval between the two successive failures.
An **MTTF** of 200 mean that one failure can be expected each 200-time units. The time units are entirely dependent on the system & it can even be stated in the number of transactions. **MTTF** is consistent for systems with large transactions.

For example, It is suitable for computer-aided design systems where a designer will work on a design for several hours as well as for Word-processor systems.

2. Mean Time to Repair (MTTR)

Once failure occurs, some-time is required to fix the error. **MTTR** measures the average time it takes to track the errors causing the failure and to fix them.

3. Mean Time Between Failure (MTBR)

We can merge **MTTF** & **MTTR** metrics to get the MTBF metric.

MTBF = MTTF + MTTR

Thus, an **MTBF** of 300 denoted that once the failure appears, the next failure is expected to appear only after 300 hours. In this method, the time measurements are real-time & not the execution time as in **MTTF**.

4. Rate of occurrence of failure (ROCOF)

It is the number of failures appearing in a unit time interval. The number of unexpected events over a specific time of operation. **ROCOF** is the frequency of occurrence with which unexpected role is likely to appear. A **ROCOF** of 0.02 mean that two failures are likely to occur in each 100 operational time unit steps. It is also called the failure intensity metric.

5. Probability of Failure on Demand (POFOD)

POFOD is described as the probability that the system will fail when a service is requested. It is the number of system deficiency given several systems inputs.

POFOD is the possibility that the system will fail when a service request is made.

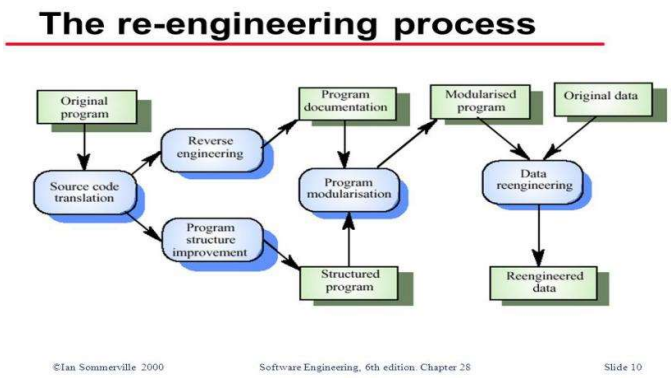
6. Availability (AVAIL)

Availability is the probability that the system is applicable for use at a given time. It takes into account the repair time & the restart time for the system. An availability of 0.995 means that in every 1000 time units, the system is feasible to be available for 995 of these. The percentage of time that a system is applicable for use, taking into account planned and unplanned downtime. If a system is down an average of four hours out of 100 hours of operation, its avail is 96%.

B) **Software Re-Engineering** is the examination and alteration of a system to reconstitute it in a new form. The principles of Re-Engineering when applied to the software development process is called software re-engineering. It affects positively at software cost, quality, service to the customer and speed of delivery. In Software Re-engineering, we are improving the software to make it more efficient and effective.

Re-Engineering cost factors:

- The quality of the software to be re-engineered.
- The tool support availability for engineering.
- Extent of the data conversion which is required.
- The availability of expert staff for Re-engineering.



Advantages and Disadvantages of Software Re-engineering :-

Advantages	Disadvantages
Produce system with a longer effective operational life.	Produce initial system that is more expensive to build and maintain.
Produces system that more closely meet user needs and requirements.	Require more extensive and accurate definitions of user needs and requirements.
Produces system with excellent documentation.	May be difficult to customize.
Produces system that needs less systems support.	Require training of maintenance staff.
Produce more flexible system.	May be difficult to use with existing system.

8. Write short notes on any TWO of the followings. [4 × 2

(a) Prototyping model
(b) Risk mitigation
(c) UML Diagrams
(d) System testing
(e) Level-3 of SEI-CMM

(Nilanjan Roy 1729143))

Ans. A) Prototyping:

Prototyping refers to an initial stage of a software release in which developmental evolution and product fixes may occur before a bigger release is initiated. These kinds of activities can also sometimes be called a beta phase or beta testing, where an initial project gets evaluated by a smaller class of users before full development.

Prototyping, as well as broad-spectrum testing and multiple software releases, is part of a more detailed process for producing sophisticated software products and services. The essential idea is that even when code features are complete on a project, the software, which is still in development, may have many bugs and user problems. For many of these to get ironed out, it helps if the software is actually in use, but developers face the issue of releasing a product that end-users can see as essentially flawed. Releasing the product to a smaller community or otherwise restricting its development in stages can be a very effective solution.

B)Risk Mitigation:

Risk mitigation can be defined as taking steps to reduce adverse effects. There are four types of risk mitigation strategies that hold unique to Business Continuity and Disaster Recovery. When mitigating risk, it's important to develop a strategy that closely relates to and matches your company's profile.

Four types of Risk Mitigation:

Risk Acceptance

Risk acceptance does not reduce any effects however it is still considered a strategy. This strategy is a common option when the cost of other risk management options such as avoidance or limitation may outweigh the cost of the risk itself. A company that doesn't want to spend a lot of money on avoiding risks that do not have a high possibility of occurring will use the risk acceptance strategy.

Risk Avoidance

Risk avoidance is the opposite of risk acceptance. It is the action that avoids any exposure to the risk whatsoever. It's important to note that risk avoidance is usually the most expensive of all risk mitigation options.

Risk Limitation

Risk limitation is the most common risk management strategy used by businesses. This strategy limits a company's exposure by taking some action. It is a strategy employing a bit of risk acceptance along with a bit of risk avoidance or an average of both. An example of risk limitation would be a company accepting that a disk drive may fail and avoiding a long period of failure by having backups.

Risk Transference

Risk transference is the involvement of handing risk off to a willing third party. For example, numerous companies outsource certain operations such as customer service, payroll services, etc. This can be beneficial for a company if a transferred risk is not a core competency of that company. It can also be used so a company can focus more on their core competencies.

2017

1. (a) What is software engineering? [2 × 10]

(b) What is software reliability?

(c) Why Spiral model is known as Meta model?

(d) What are the characteristics of a good SRS document?

(e) What do you understand by the term "Software Crisis"? List the causes of it?

(f) What is Function Point Metric? What factors we consider for this metric?

(g) Differentiate between Empirical estimation and Heuristic techniques.

(h) What is the difference between verification and validation?

(i) Differentiate between code walk-through and code inspection.

(j) What do you mean by the terms "ambiguous requirement" and "inconsistent requirement"? Support one example for each of these.

(Tuneer Bhattacharya 1729091))

a) **Software engineering** is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

b) Software Reliability is the probability of failure-free software operation for a specified period of time in a specified environment. Software Reliability is also an important factor affecting system reliability. It differs from hardware reliability in that it reflects the design perfection, rather than manufacturing perfection.

c) The Spiral model is called as a Meta Model because it subsumes all the other SDLC models. For example, a single loop spiral actually represents the Iterative Waterfall Model.The spiral model uses the approach of Prototyping Model by building a prototype at the start of each phase as a risk handling technique.

d) The characteristics of a good SRS document are:-

1.Correctness

2.Completeness

3.Consistency

4.Unambiguousness

5.Ranking for importance and stability

6.Modifiability

e) Software crisis is a term used in the early days of computing science for the difficulty of writing useful and efficient computer programs in the required time. The software crisis was due to the rapid increases in computer power and the complexity of the problems that could not be tackled.

f) Function points are one of the most widely used measures of software size. The basis of function points is that the "functionality " of the system that is; what the system performs, is the measure of the system size. In function points, the system functionally is calculated in terms of the number of function it implements, the number of inputs, the number of output etc. Parameter that can be obtained after requirements analysis and that are independent of the specification (and implementation) language.

g) A heuristic technique, often called simply a heuristic, is any approach to problem solving, learning, or discovery that employs a practical method not guaranteed to be optimal or perfect, but sufficient for the immediate goals.

Empirical methods main motivation is that it is needed from an engineering perspective to allow for informed and well-grounded decision.

h)

Verification	***Validation***
1. **Verification** is a static practice of verifying documents, design, code and program.	1. **Validation** is a dynamic mechanism of validating and testing the actual product.
2. It does not involve executing the code.	2. It always involves executing the code.

3. It is human based checking of documents and files.	*3.* It is computer based execution of program.
4. **Verification** uses methods like inspections, reviews, walkthroughs, and Desk-checking etc.	*4.* **Validation** uses methods like black box (functional) testing, gray box testing, and white box (structural) testing etc.
5. **Verification** is to check whether the software conforms to specifications.	5. **Validation** is to check whether software meets the customer expectations and requirements.
6. It can catch errors that validation cannot catch. It is low level exercise.	*6. It can catch errors that verification cannot catch. It is High Level Exercise.*
7. Target is requirements specification, application and software architecture, high level, complete design, and database design etc.	*7.* Target is actual product-a unit, a module, a bent of integrated modules, and effective final product.
8. **Verification** is done by QA team to ensure that the software is as per the specifications in the SRS document.	*8.* ***Validation*** *is carried out with the involvement of testing team.*
9. It generally comes first-done before validation.	*9.* It generally follows after **verification**.

i)

Inspection	**Walkthrough**
Formal	Informal
Initiated by the project team	Initiated by the author
Planned meeting with fixed roles assigned to all the members involved	Unplanned.
Reader reads the product	Author reads the product

code. Everyone inspects it and comes up with defects.	code and his team mate comes up with defects or suggestions
Recorder records the defects	Author makes a note of defects and suggestions offered by team mate
Moderator has a role in making sure that the discussions proceed on the productive lines	Informal, so there is no moderator

j) Ambiguous functional requirements are any requirements that:

- have any kind of ambiguity.
- have more than one type of interpretation.

Any task in requirements that can have more than one correct output that is contingent on a different understanding of the task is ambiguous.Inconsistency is an inevitable part of software development processes. Even in the most well-defined, managed and optimised development process, system requirements are often uncertain or contradictory,alternative design solutions exist, and errors in implementation arise.The requirements engineering stage of development is particularly illustrative of such inconsistencies. During requirements acquisition, customer requirements are often sketchy and uncertain. For large projects in particular, a number of “client authorities” may exist who haveconflicting, even contradictory requirements. In many instances customers may not even be certain of their own needs, and a requirements engineer’s job is partly to elicit and clarify these needs. The requirements specification produced as a result of such a specification and analysis process however is not static: it continues to evolve as new requirements are added and conflicts identified are resolved.Thus, there is a wide range of possible causes of inconsistencies in software development. Many of these are due to the heterogeneity of the products being developed (e.g., systems deploying different technologies) and the multiplicity of stakeholders and/or development
participants involved in the development process

2. (a) Suggest a suitable life cycle model for a software project where several kinds of risks are difficult to anticipate at the start of the project. Justify your answer and explain all phases of the proposed model in detail with schematic diagram. [4]

(b) What are the steps involved in project planning? Explain them briefly. [4]

(Tanya Jaiswal 1729088))

Ans. **b)** Project planning doesn't have to be difficult or cause any nervous stress since the beginning of every project is basically the same. You can follow the same set project planning steps and hone them through experience of every project you are involved with.

Breaking down the steps

A) Create and Analyze Business Case

The business case is the reason why your organization needs to carry out the project. It should outline the problem, such as a lack of repeat customers or a day longer supply line than competitors and describe how this will be solved and how much monetary benefit should accrue to the organization once the project is completed.

B) Identify and Meet Relevant Stakeholders for Approval

Identifying project stakeholders means listing anyone who will be affected by your project, so includes the public and government regulatory agencies. For the project planning phase however, it should only be necessary to meet those who will directly decide whether the project will happen or not.

C) Define Project Scope

The scope of your project is an outline of what it is and isn't setting out to achieve. It is necessary to delineate the boundaries of your project to prevent "scope creep", i.e. your resources going towards something that's not in your project's goals.

D) Set Goals and Objectives

The goals and objectives for your project will build on the initial objectives outlined in the business plan. At this step you will give finer detail to the initial broad ideas and set them in a project charter as reference points for your project as it proceeds.

It is a two-way street however, as, with growing skills shortages and the likely exodus of hundreds of thousands of EU citizens employed by UK business and public service, the country is likely to face huge labor shortages in the near future which threaten to derail its current high economic performance.

E) Determine Deliverables

Deliverables are the concrete results that your project produces. One of the most important project planning steps is to decide on what these deliverables will be and who is responsible for both producing and receiving them.

F) Create Project Schedule and Milestones

Your project schedule is a very important document that outlines when different tasks of a project are due to begin and end, along with major measurement milestones. It will be referred to when measuring project progress. It will be available to all stakeholders and should be adhered to as closely as possible.

G) Assignment of Tasks

Within your team everyone should know what their role is and who is responsible for different elements of the project. Assigning tasks clearly should remove any uncertainty about roles and responsibilities on your team.

H) Carry Out Risk Assessment

Having a functional risk management plan means performing a strong assessment at the planning stage of the project. All potential risks should be identified along with their possible effect on the project and likelihood of occurring.

3. (a) What is LOC? How does one estimates LOC of a Software product? List two short comings of LOC. [4]

(b) How CMM is different from ISO? Discuss about the KPA's present in various CMM levels. [4]

(Sagnik Misra 1729151))

Ans. A) Source lines of code (SLOC), also known as lines of code (**LOC**), is a **software** metric used to measure the size of a computer program by counting the number of lines in the text of the program's source code.

Lines of Code (LOC): As the name suggest, LOC count the total number of lines of source code in a project. The units of LOC are:

- KLOC- Thousand lines of code
- NLOC- Non comment lines of code
- KDSI- Thousands of delivered source instruction

The size is estimated by comparing it with the existing systems of same kind. The experts use it to predict the required size of various components of software and then add them to get the total size.

Advantages:

- Universally accepted and is used in many models like COCOMO.
- Estimation is closer to developer's perspective.
- Simple to use.

Disadvantages or Shortcoming:

- Different programming languages contains different number of lines.
- No proper industry standard exist for this technique.

B)ISO :-

(i) It is created for hard goods manufacturing industries.
(ii) It specifies concepts, principles and safeguards that should be in place.
(iii) Focus is customer supplier relationship, attempting to reduce customer's risk in choosing a supplier.

CMM:-

(i) Focus on the software supplier to improve its interval processes to achieve a higher quality product for the benefit of the customer.

(ii) CMM provides detailed and specific definition of what is required for given levels.
(iii) It is created for software industry.

The 5 levels of CMM are as follows:

Level-1: Initial –

- No KPA's defined.
- Processes followed are adhoc and immature and are not well defined.
- Unstable environment for software dvelopment.
- No basis for predicting product quality, time for completion, etc.

Level-2: Repeatable –

- Focuses on establishing basic project management policies.
- Experience with earlier projects is used for managing new similar natured projects.

KPA's:

- Project Planning- It includes defining resources required, goals, constraints, etc. for the project. It presents a detailed plan to be followed systematically for successful completion of a good quality software.
- Configuration Management- The focus is on maintaining the performance of the software product, including all its components, for the entire lifecycle.
- Requirements Management- It includes the management of customer reviews and feedback which result in some changes in the requirement set. It also consists of accommodation of those modified requirements.
- Subcontract Management- It focuses on the effective management of qualified software contractors i.e. it manages the parts of the software which are developed by third parties.
- Software Quality Assurance- It guarantees a good quality software product by following certain rules and quality standard guidelines while development.

Level-3: Defined –

- At this level, documentation of the standard guidelines and procedures takes place.
- It is a well defined integrated set of project specific software engineering and management processes.

KPA's:

- Peer Reviews- In this method, defects are removed by using a number of review methods like walkthroughs, inspections, buddy checks, etc.
- Intergroup Coordination- It consists of planned interactions between different development teams to ensure efficient and proper fulfilment of customer needs.
- Organization Process Definition- It's key focus is on the development and maintenance of the standard development processes.
- Organization Process Focus- It includes activities and practices that should be followed to improve the process capabilities of an organization.
- Training Programs- It focuses on the enhancement of knowledge and skills of the team members including the developers and ensuring an increase in work efficiency.

Level-4: Managed –

- At this stage, quantitative quality goals are set for the organization for software products as well as software processes.
- The measurements made help the organization to predict the product and process quality within some limits defined quantitatively.

KPA's:

- Software Quality Management- It includes the establishment of plans and strategies to develop a quantitative analysis and understanding of the product's quality.
- Quantitative Management- It focuses on controlling the project performance in a quantitative manner.

Level-5: Optimizing –

- This is the highest level of process maturity in CMM and focuses on continuous process improvement in the organization using quantitative feedback.
- Use of new tools, techniques and evaluation of software processes is done to prevent recurrence of known defects.

KPA's:

- Process Change Management- Its focus is on the continuous improvement of organization's software processes to improve productivity, quality and cycle time for the software product.
- Technology Change Management- It consists of identification and use of new technologies to improve product quality and decrease the product development time.
- Defect Prevention- It focuses on identification of causes of defects and to prevent them from recurring in future projects by improving project defined process.

4. (a) What do you mean by the term cohesion and coupling in the context of software design? Explain any two different types of cohesion and coupling with examples. [4]

(b) Explain in details the basic COCOMO model. Why COCOMO model preferred over other available models. [4]

(Swadesh Nayak 1729085))

Ans. A) Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling.

Types of Coupling:

Data Coupling: If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent to each other and communicating through data. Module communications don't contain tramp data. Example-customer billing system.

Stamp Coupling: In stamp coupling, the complete data structure is passed from one module to another module. Therefore, it involves tramp data. It may be necessary due to efficiency factors- this choice made by the insightful designer, not a lazy programmer.

Cohesion is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. A good software design will have high cohesion.

Functional Cohesion: Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation.

Sequential Cohesion: An element outputs some data that becomes the input for other element, i.e., data flow between the parts. It occurs naturally in functional programming languages.

B)Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e number of Lines of Code. It is a procedural cost estimate model for software projects and often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time and quality.

Advantage of COCOMO over other model:

1. COCOMO is factual and easy to interpret. One can clearly understand how it works.
2. Accounts for various factors that affect cost of the project.
3. Works on historical data and hence is more predictable and accurate.
4. The drivers are very helpful to understand the impact on the different factors that affect the project costs.

5. (a) What do you mean by testing? Discuss about following testing strategies for black box testing with suitable example. [4]
(i) Equivalent class partitioning (ii) Boundary value analysis

(Sayahna Sengupta 1729158)

Types of Software Testing

Introduction:-

Testing is a process of executing a program with the aim of finding error. To make our software perform well it should be error free.If testing is done successfully it will remove all the errors from the software.

Principles of Testing:-

(i) All the test should meet the customer requirements
(ii) To make our software testing should be performed by third party
(iii) Exhaustive testing is not possible.As we need the optimal amount of testing based on the risk assessment of the application.
(iv) All the test to be conducted should be planned before implementing it
(v) It follows pareto rule(80/20 rule) which states that 80% of errors comes from 20% of program components.

Boundary Value Analysis & Equivalence Partitioning with Examples

Practically, due to time and budget considerations, it is not possible to perform exhausting testing for each set of test data, especially when there is a large pool of input combinations.

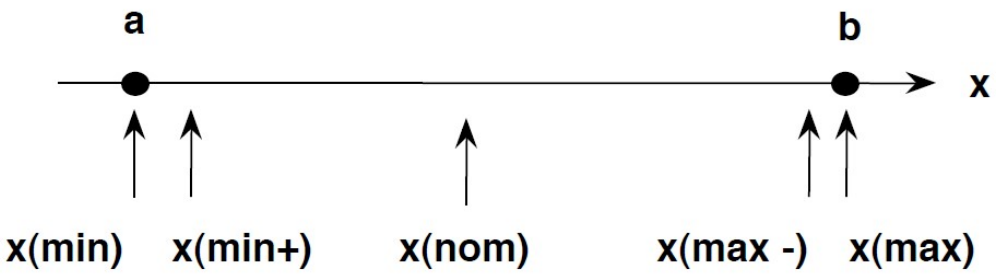
- We need an easy way or special techniques that can select test cases intelligently from the pool of test-case, such that all test scenarios are covered.
- We use two techniques - **Equivalence Partitioning & Boundary Value Analysis testing techniques** to achieve this.

What is Boundary Testing?

Boundary testing is the process of testing between extreme ends or boundaries between partitions of the input values.

- So these extreme ends like Start- End, Lower- Upper, Maximum-Minimum, Just Inside-Just Outside values are called boundary values and the testing is called "boundary testing".
- The basic idea in boundary value testing is to select input variable values at their:

1. Minimum
2. Just above the minimum
3. A nominal value
4. Just below the maximum
5. Maximum



- In Boundary Testing, Equivalence Class Partitioning plays a good role
- Boundary Testing comes after the Equivalence Class Partitioning.

What is Equivalent Class Partitioning?

Equivalent Class Partitioning is a black box technique (code is not visible to tester) which can be applied to all levels of testing like unit, integration, system, etc. In this technique, you divide the set of test condition into a partition that can be considered the same.

- It divides the input data of software into different equivalence data classes.
- You can apply this technique, where there is a range in the input field.

Example 1: Equivalence and Boundary Value

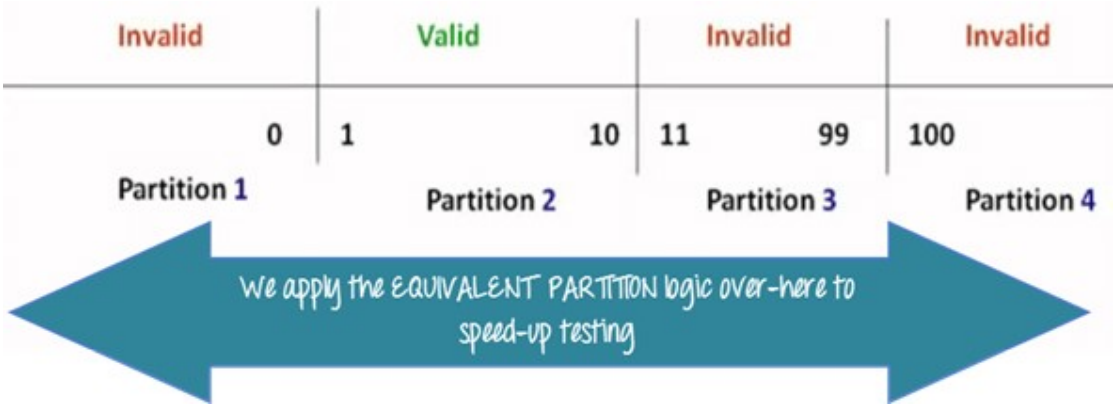
- Let's consider the behavior of Order Pizza Text Box Below
- Pizza values 1 to 10 is considered valid. A success message is shown.
- While value 11 to 99 are considered invalid for order and an error message will appear, **"Only 10 Pizza can be ordered"**

Order Pizza:

Here is the test condition

1. Any Number greater than 10 entered in the Order Pizza field(let say 11) is considered invalid.
2. Any Number less than 1 that is 0 or below, then it is considered invalid.
3. Numbers 1 to 10 are considered valid
4. Any 3 Digit Number say -100 is invalid.

We cannot test all the possible values because if done, the number of test cases will be more than 100. To address this problem, we use equivalence partitioning hypothesis where we divide the possible values of tickets into groups or sets as shown below where the system behavior can be considered the same.



The divided sets are called Equivalence Partitions or Equivalence Classes. Then we pick only one value from each partition for testing. The hypothesis behind this technique is **that if one condition/value in a partition passes all others will also pass**. Likewise**, if one condition in a partition fails, all other conditions in that partition will fail**.

6. Consider a student performance measurement system in which an instructor evaluates the students on the basis of mid-semester examinations, final examinations, presentation, attendance, lab examinations and student cognitive skills.

(a) Write four functional requirements along with their inputs, outputs and constraints from the above system. [4]

(b) Draw the context level diagram and Level-1 DFD. [2]

(c) Write the data dictionary in a tabular form. [1]

(d) Draw structure chart from Level-1 DFD. [1]

(Muskan Mohanty 1729139)

FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS OF STUDENT ATTENDANCE MANAGEMENT SYSTEM

Functional requirements of student attendance management system:-

The functional requirement of this is that it does what it is meant for. A functional requirement describes what a software system should do, while non-functional requirements place constraints on how the system will do so.

Functional requirements specify a function that a system or system component must be able to perform. It can be documented in various ways. The most common ones are written descriptions in documents and use cases.
Few of its functional requirements are as given below:-

User data should be fed into the system: this system is doing that properly in user entity.

Admin can add the users: Admin can verify and add the user, which is they are doing with this system.

Non-Functional requirements of student attendance management system:-

Non-functional necessities square measure the other demand than practical necessities. This square measure the necessities that specify criteria which will be wont to choose the operation of a system, instead of specific behaviors.

Non-functional necessities square measure within the style of “system shall be “, associate degree overall property of the system as a full or of a specific facet and not a particular operation. The system’s overall properties remarkably mark the distinction between whether or not the event project has succeeded or unsuccessful.

Non-functional necessities – are often divided into 2 main categories:
Execution qualities, like security and usefulness, that square measure evident at the run time.
Evolution qualities, like liabilities, maintainability, flexibility and quantifiable, that square measure embodied within the static structure of the code.

Non-functional of student attendance management system necessities place restrictions on the merchandise being developed, the event method, and specify external constraints that the merchandise should meet

Our project qualifies all the criteria of functional and not functional accordingly and the system is up to mark performance vise.Here we need to take care of few more things before heading towards the system.

The most important feature of application world is that application's ease of usage .application will easy to use if made while keeping in mind that user need not think twice about searching any feature.

Everything should be made distinctive by using the color combination such that everything needed most frequently highlighted with focus colors. We can use simple layouts like the card and grid layout etc.

By varying color and other UI combination, many good intuitive interfaces can be made. Which ultimately makes interface easy to use for a long time.

Unlike ancient style wherever the goal is to form the thing or application physically enticing, the goal of interface style is to form the user's interaction expertise as straightforward and intuitive as attainable – what's typically known as user-centered style.

Where smart graphic/industrial style is daring and eye catching, smart interface style is commonly delicate and invisible.

Keep things simple and consistent:-

Simple and Harmonic way making UI is very intuitive and needs to followed

Make good use of typography:-

The typography is taken care very strictly as the need of the system.

Use color and contrast properly:-

Color combo of lite and dark is a good way emphasize and done well in this system

Consider feedback messages:-

The feedback form is a very good way taking feedback of forms and improving the system.

Simplified forms:-

The form is made simple to fill with the clean user interface.

ATTENDANCE MANAGEMENT SYSTEM PROJECT

CONCLUSION OF STUDENT ATTENDANCE MANAGEMENT SYSTEM

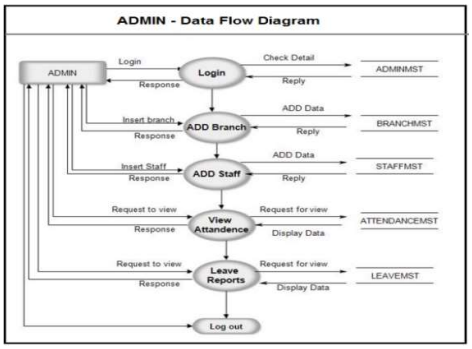
Finally in student attendance management system, the outcome of all the hard work done for attendance management system is here. It is a software which helps the user to work with the attendance, fees update, course update and messages etc.

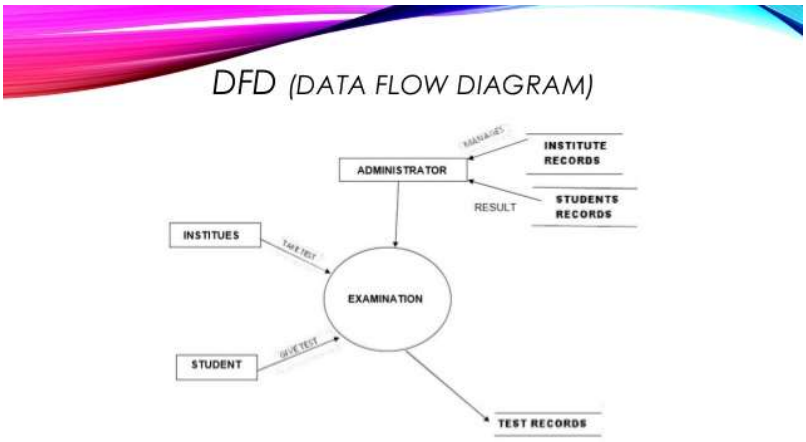
This software reduces the amount of manual data entry and gives greater efficiency. The User Interface of it is very friendly and can be easily used by anyone. It also decreases the amount of time taken to write details and other modules.

All the details about students, teachers and their other tasks can only be seen by the verified users. This Attendance Management System is a solution to all the problems related to the attendance, message, fee status, courses taken by the teachers and the students etc.

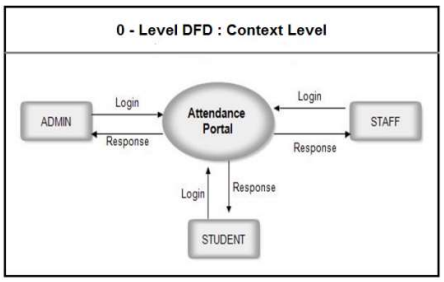
At the end, we can say that this software is performing all the tasks accurately and is doing the work for which it is made and this system can be implemented in N number of colleges and schools.

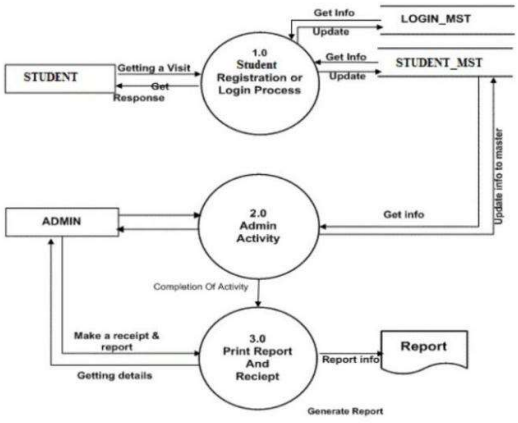
DATA FLOW DIAGRAM:





CONTEXT DIAGRAM:





Level 1

8. Write short notes on *any TWO* of the following: [4 × 2]

(a) RAD model

(b) Software Configuration Management

(c) Activity networks

(d) UML diagram

(Tanisha Bharadwaj 1729088)

(i)**RAD model**:-

The Rapid Application Development Model was first proposed by IBM in 1980's. The critical feature of this model is the use of powerful development tools and techniques.

A software project can be implemented using this model if the project can be broken down into small modules wherein each module can be assigned independently to separate teams. These modules can finally be combined to form the final product.

Advantages –

- Use of reusable components helps to reduce the cycle time of the project.
- Feedback from the customer is available at initial stages.
- Reduced costs as fewer developers are required.
- Use of powerful development tools results in better quality products in comparatively shorter time spans.
- The progress and development of the project can be measured through the various stages.
- It is easier to accommodate changing requirements due to the short iteration time spans.

Disadvantages –

- The use of powerful and efficient tools requires highly skilled professionals.
- The absence of reusable components can lead to failure of the project.
- The team leader must work closely with the developers and customers to close the project in time.
- The systems which cannot be modularized suitably cannot use this model.
- Customer involvement is required throughout the life cycle.
- It is not meant for small scale projects as for such cases, the cost of using automated tools and techniques may exceed the entire budget of the project.

(iii)**Activity Networks**:-

An Activity Network Diagram (AND) is also called an Arrow Diagram (because the pictorial display has arrows in it) or a PERT (Program Evaluation Review Technique) Diagram, and it is used for identifying time sequences of events which are pivotal to objectives. In Critical Path Analysis this helps the teams to comprehend specific event sequences driving time requirements for objective achievement. Activity Network Diagrams are also very useful when a project has multiple activities which need simultaneous management.

Activity Network Diagrams started out as an engineering and construction project management tool. Critical Path Analysis draws on this methodology to identify and standardize medical management activities.

An Activity Network Diagram helps to find out the most efficient sequence of events needed to complete any project. It enables you to create a realistic project schedule by graphically showing

(i)The total amount of time needed to complete the project

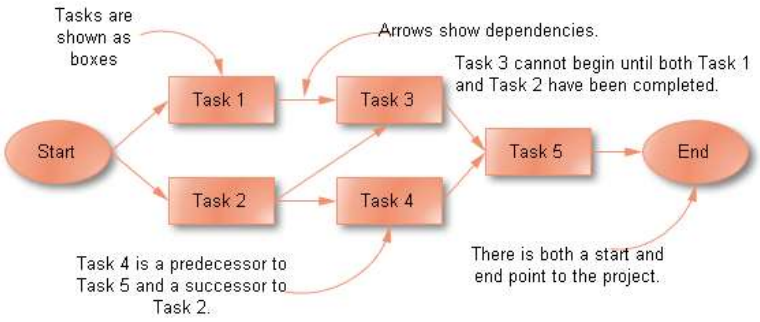
(ii)The sequence in which tasks must be carried out

(iii)Which tasks can be carried out at the same time

(iv)Which are the critical tasks that you need to keep an eye on.

Example of Activity Network Diagram

A project is composed of a set of actions or tasks which usually have some kind of interdependency. For example, before an axle can be turned, it must first be designed, the metal must be purchased, etc. This type of complex system is much easier to understand through the use of diagrams than through textual description, as actual interconnections between tasks can be shown. You can draw the activity network diagram easily with Edraw software.



The Activity Network diagram displays interdependencies between tasks through the use of boxes and arrows. Arrows pointing into a task box come from its predecessor tasks, which must be completed before the task can start. Arrows pointing out of a task box go to its successor tasks, which cannot start until at least this task is complete.

2018

1. (a) Explain if the following statement true or false -There are well defined steps through which a problem is solved in exploratory style. [2×10]

(b) Without developing SRS document, an organization might face severe problems. (True / False)Justify.

(c) What are the non-functional requirements which need to be captured well advance in a software project?

(d) Explain the role of abstraction in software design.

(e) List the advantages ad limitations of prototype life cycle model.

(f) "Statement coverage based testing is stronger testing strategy than branch coverage based testing." (True/ False) Justify your answer.

(g) How debugging is different from testing? List the types of debugging.

(h) What do you understand by adaptive maintenance? Explain with example.

(i) Differentiate between function oriented design and object oriented design.

(j) What is the difference between code walkthrough and code inspection?

(Sagnik Sarbadhikari 1729151))

(b). True, Software requirements specification(SRS) is important for developers because it minimizes the amount of time and effort developers have to expend to achieve desired software goals. It thus reduces development cost. This also benefits the client company because the lesser the development cost, the lesser the developers will charge from the client. And, if composed properly, an SRS ensures that there is less possibility of future redesigns as there is less chance of mistake on the part of developers as they have a clear idea on the functionalities and externalities of the software. It also helps clear any communication problems between the client and the developer. Furthermore, an SRS serves to form a foundation of mutual agreement between the client and the developer (supplier). It also serves as the document to verify the testing processes.

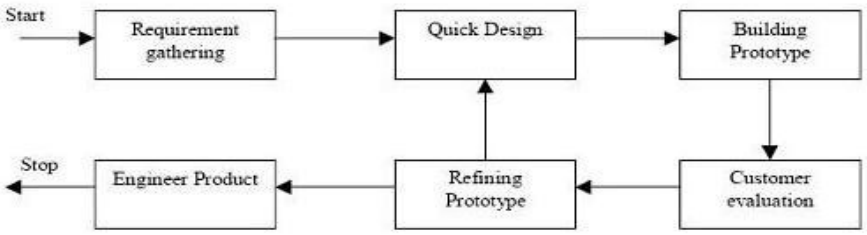
(c). Nonfunctional Requirements (NFRs) define system attributes such as security, reliability, performance, maintainability, scalability, and usability. They serve as constraints or restrictions on the design of the system across the different backlogs.

Also known as system qualities, nonfunctional requirements are just as critical as functional Epics, Capabilities, Features, and Stories. They ensure the usability and effectiveness of the entire system. Failing to meet any one of them can result in systems that fail to satisfy internal business, user, or market needs,

or that do not fulfill mandatory requirements imposed by regulatory or standards agencies.

(d). Abstraction is a technique for arranging complexity of computer systems. It works by establishing a level of simplicity on which a person interacts with the system, suppressing the more complex details below the current level. The programmer works with an idealized interface (usually well defined) and can add additional levels of functionality that would otherwise be too complex to handle. For an example, a programmer writing code that involves numerical operations may not be interested in the way numbers are represented in the underlying hardware (e.g. whether they're 16 bit or 32 bit integers), and where those details have been suppressed it can be said that they were abstracted away, leaving simply numbers with which the programmer can work.

(e). Diagram of Prototype model:



Prototyping Model

Advantages of Prototype model:

- Users are actively involved in the development
- Since in this methodology a working model of the system is provided, the users get a better understanding of the system being developed.
- Errors can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily
- Confusing or difficult functions can be identified
  Requirements validation, Quick implementation of, incomplete, but functional, application.

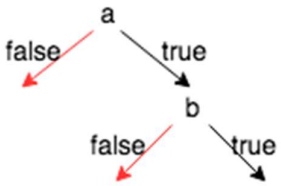
Disadvantages of Prototype model:

- Leads to implementing and then repairing way of building systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- Incomplete application may cause application not to be used as the full system was designed
  Incomplete or inadequate problem analysis.

(f). False, If the tests have **complete branch coverage then we can say it also has complete statement coverage**, but not the vice versa.
100% branch coverage => 100% statement coverage
100% statement coverage does not imply 100% branch coverage
the reason is in branch coverage apart from executing all the statements, we should also verify if the tests executes all branches, which can be interpreted as covering all edges in the control flow branch

```
if(a){
    if(b){
       bool statement1 = true;
    }
}
```

a = true, b = true will give 100% statement coverage, but not branch coverage



In the branch coverage we need to **cover all the edges**, which we missed in the statement coverage shown as **red lines** in the above image

(g). When the testing of any software application gets completed successfully, debugging activity starts. During testing errors come to the light through efficiently designed test cases. Thereafter the process by which the detected errors are practically removed is called debugging. After testing, we immediately start our hunt for the errors i.e., to identify the interface or the module responsible for causing it. Then that section of the code is to be studied to determine the cause of the problem. This process is called debugging.

Hence, we can say that debugging is the name of an activity wherein we firstly locate the errors & correct them thereafter. Debugging is totally different from testing. However debugging is an activity, which follows testing. When we execute a test case, the debugging activity gets started.

(h).Adaptive maintenance is the implementation of changes in a part of the system, which has been affected by a change that occurred in some other part of the system. Modification of a software product performed after delivery to keep a software product usable in a changed or changing environment.

(i). Both programming concepts have a goal of wanting to create easily understandable programs that are free of bugs and can be developed fast. Both concepts have different methods for storing the data and how to manipulate the data. In object oriented programming, you store the data in attributes of objects and have functions that work for that object and do the manipulation. In functional programming, we view everything as data transformation. Data is not stored in objects, it is

transformed by creating new versions of that data and manipulating it using one of the many functions.

(j).

Inspection	Walkthrough
Formal	Informal
Initiated by the project team	Initiated by the author
Planned meeting with fixed roles assigned to all the members involved	Unplanned.
Reader reads the product code. Everyone inspects it and comes up with defects.	Author reads the product code and his team mate comes up with defects or suggestions
Recorder records the defects	Author makes a note of defects and suggestions offered by team mate
Moderator has a role in making sure that the discussions proceed on the productive lines	Informal, so there is no moderator

2. (a) Suppose you are the project manager of a software project requiring the following activities: [4]

Activity No.	Activity Name	Duration (weeks)	Immediate Predecessor
1	Obtain requirement	3	-
2	Analyze operation	3	-
3	Define subsystem	1	1
4	Develop database	3	1
5	Make decision analysis	2	2
6	Identify Constraints	1	5
7	Build Module 1	7	3, 4, 6
8	Build Module 2	11	3, 4, 6
9	Build Module 3	17	3, 4, 6
10	Write report	9	6
11	Integration and Test	7	7, 8, 9
12	Implementation	1	10, 11

(a) Draw the Activity Network representation of the project.

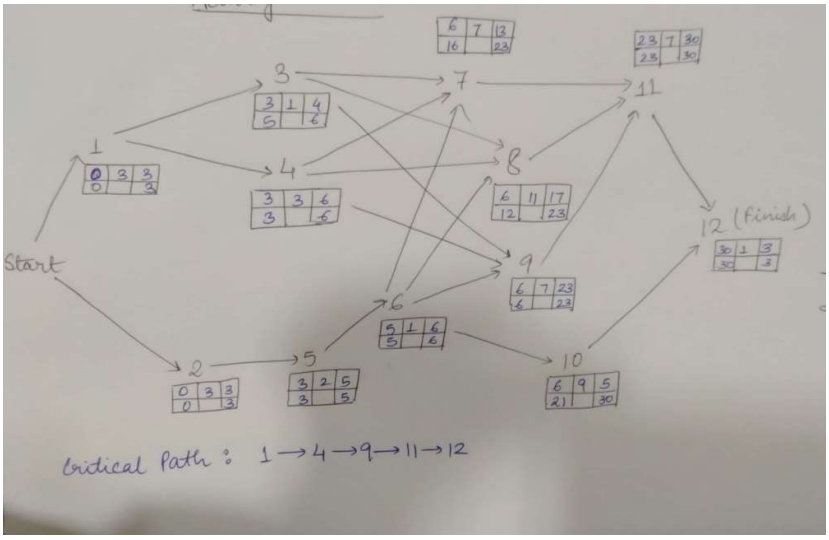
(b) Identify the critical path.

(c) Determine Earliest Start, Earliest Finish, and Latest Start, Latest Finish for every task.

(b) (i) What do you understand by software configuration management? What is the need for it? [4]

(ii) Explain how Mixed control team is evolved from Chief-programmer team and Democratic team.

(Sagnik Misra 1729152))



(B)

- **Software Configuration Management** is defined as a process to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle. It is abbreviated as the SCM process in software engineering. The primary goal is to increase productivity with minimal mistakes.

- **Need:**

  - There are multiple people working on software which is continually updating
  - It may be a case where multiple version, branches, authors are involved in a software project, and the team is geographically distributed and works concurrently
  - Changes in user requirement, policy, budget, schedule need to be accommodated.
  - Software should able to run on various machines and Operating Systems
  - Helps to develop coordination among stakeholders
  - SCM process is also beneficial to control the costs involved in making changes to a system

Team structure addresses the issue of organization of the individual project teams. There are some possible ways in which the individual project teams can be organized. There are mainly three formal team structures: chief programmer, democratic, and the mixed team organizations.

Chief Programmer Team

- In this team organization, a senior engineer provides the technical leadership and is designated as the chief programmer. The chief programmer partitions the task into small activities and assigns them to the team members.
- He also verifies and integrates the products developed by different team members.The chief programmer provides an authority, and this structure is arguably more efficient than the democratic team for well-understood problems.
- However, the chief programmer team leads to lower team morale, since team-members work under the constant supervision of the chief programmer. This also inhibits their original thinking.
- The chief programmer team is subject to single point failure since too much responsibility and authority is assigned to the chief programmer.

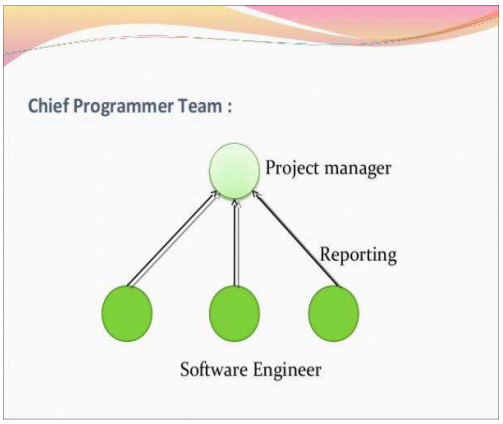
- The chief programmer team is probably the most efficient way of completing simple and small projects since the chief programmer can work out a satisfactory design and ask the programmers to code different modules of his design solution.
- For example, suppose an organization has successfully completed many simple MIS projects. Then, for a similar MIS project, chief programmer team structure can be adopted. The chief programmer team structure works well when the task is within the intellectual grasp of a single individual. However, even for simple and well-understood problems, an organization must be selective in adopting the chief programmer structure. The chief programmer team structure should not be used unless the importance of early project completion outweighs other factors such as team morale, personal developments, life-cycle cost etc

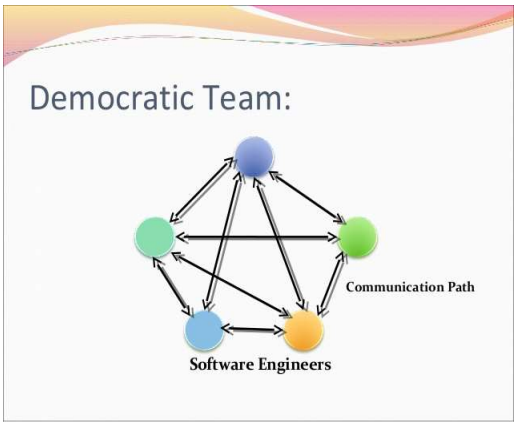
Democratic Team :

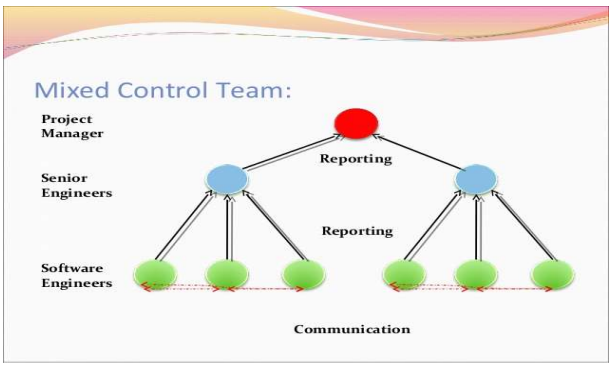
- The democratic team structure, as the name implies, does not enforce any formal team hierarchy. Typically, a manager provides the administrative leadership. At different times, different members of the group provide technical leadership.
- The democratic organization leads to higher morale and job satisfaction. Consequently, it suffers from less man-power turnover.
- Also, democratic team structure is appropriate for less understood problems, since a group of engineers can invent better solutions than a single individual as in a chief programmer team.
- A democratic team structure is suitable for projects requiring less than five or six engineers and for research-oriented projects. For large sized projects, a pure democratic organization tends to become chaotic. The democratic team organization encourages egoless programming as programmers can share and review one another's work.

The mixed team organization:

- As the name implies,it draws upon the ideas from both the democratic organization and the chief-programmer organization.
- This team organization incorporates both hierarchical reporting and democratic set up.
- The mixed control team organization is suitable for large team sizes.
- The democratic arrangement at the senior engineers level is used to decompose the problem into small parts. Each democratic setup at the programmer level attempts solution to a single part.
- Thus, this team organization is eminently suited to handle large and complex programs. This team structure is extremely popular and is being used in many software development companies.







3. (a) What do you understand by CMM? Discuss the various Key Process Areas present in various CMM levels. How CMM is different from ISO? [4]

(b) What are the shortcomings of LOC in software size estimation? In what way Function point metric overcomes them? Explain the factors we consider for function point metric. [4]

(Tuneer Bhattcharya 1729091)

- CMM(Capability Maturity Model) is not a software process model. It is a framework which is used to analyse the approach and techniques followed by any organization to develop a software product.It also provides guidelines to further enhance the maturity of those software products.It is based on profound feedback and development practices adopted by the most successful organizations worldwide.
- Each of these KPA's defines the basic requirements that should be met by a software process in order to satisfy the KPA and achieve that level of maturity.
- 5 levels of CMM:

Level-1: Initial –

No KPA's defined.Processes followed are adhoc and immature and are not well defined.Unstable environment for software dvelopment.No basis for predicting product quality, time for completion, etc.

Level-2: Repeatable –

Focuses on establishing basic project management policies.Experience with earlier projects is used for managing new similar natured projects.

- **Project Planning**- It includes defining resources required, goals, constraints, etc. for the project. It presents a detailed plan to be followed systematically for successful completion of a good quality software.
- **Configuration Management-** The focus is on maintaining the performance of the software product, including all its components, for the entire lifecycle.
- **Requirements Management-** It includes the management of customer reviews and feedback which result in some changes in the requirement set. It also consists of accommodation of those modified requirements.
- **Subcontract Management**- It focuses on the effective management of qualified software contractors i.e. it manages the parts of the software which are developed by third parties.
    - **Software Quality Assurance-** It guarantees a good quality software product by following certain rules and quality standard guidelines while development.

Level-3: Defined –

At this level, documentation of the standard guidelines and procedures takes place. It is a well defined integrated set of project specific software engineering and management processes.

- **Peer Reviews-** In this method, defects are removed by using a number of review methods like walkthroughs, inspections, buddy checks, etc.
- **Intergroup Coordination-** It consists of planned interactions between different development teams to ensure efficient and proper fulfilment of customer needs.
- **Organization Process Definition-** It's key focus is on the development and maintenance of the standard development processes.
- **Organization Process Focus-** It includes activities and practices that should be followed to improve the process capabilities of an organization.
- **Training Programs-** It focuses on the enhancement of knowledge and skills of the team members including the developers and ensuring an increase in work efficiency.

Level-4: Managed –

At this stage, quantitative quality goals are set for the organization for software products as well as software processes.The measurements made help the organization to predict the product and process quality within some limits defined quantitatively.

- **Software Quality Management-** It includes the establishment of plans and strategies to develop a quantitative analysis and understanding of the product's quality.
- **Quantitative Management-** It focuses on controlling the project performance in a quantitative manner.

Level-5: Optimizing –

This is the highest level of process maturity in CMM and focuses on continuous process improvement in the organization using quantitative feedback.Use of new tools, techniques and evaluation of software processes is done to prevent recurrence of known defects.

- **Process Change Management-** Its focus is on the continuous improvement of organization's software processes to improve productivity, quality and cycle time for the software product.
- **Technology Change Management-** It consists of identification and use of new technologies to improve product quality and decrease the product development time.
- **Defect Prevention-** It focuses on identification of causes of defects and to prevent them from recurring in future projects by improving project defined process.

- ISO is a set of international standarads on quality management and quality assurance developed to help companies effectively document the quality system elements needed to an efficient quality system where as Capability Maturity Model (CMM) specifies an increasing series of levels of a software development organization.
- ISO's focus is customer supplier relationship, attempting to reduce customer's risk in choosing a supplier where as CMM focus on the software supplier to improve its interval processes to achieve a higher quality product for the benefit of the customer.
- ISO is basically created for hard goods manufacturing industries but CMM is created for software industries.

(b) Short comings of LOC:

- It is defined on code. For example it cannot measure the size of specification.
- It characterise only one specific view of size, namely length, it takes no account of functionality or complexity

- Bad software design may cause excessive line of code
- It is language dependent
- Users cannot easily understand it.

Function point Metric overcome it by:

a. It can be applied early in the software development life cycle.
b. It is independent of the programming language, technology, techniques.
c. It provides a reliable relationship to effort.
d. Creation of more function points can define productivity goal as opposed to LOC.
e. Productivity of projects written in different languages can be measured.
f. They can be counted early and often.
g. They can be used for GUI systems.
h. It considers environmental factors.

Factors we consider for function point metric:

EI − The number of external inputs. These are elementary processes in which derived data passes across the boundary from outside to inside. In an example library database system, enter an existing patron's library card number.

EO − The number of external output. These are elementary processes in which derived data passes across the boundary from inside to outside. In an example library database system, display a list of books checked out to a patron.

EQ − The number of external queries. These are elementary processes with both input and output components that result in data retrieval from one or more internal logical files and external interface files. In an example library database system, determine what books are currently checked out to a patron.

ILF − The number of internal log files. These are user identifiable groups of logically related data that resides entirely within the applications boundary that are maintained through external inputs. In an example library database system, the file of books in the library.

ELF − The number of external log files. These are user identifiable groups of logically related data that are used for reference purposes only, and which reside entirely outside the system. In an example library database system, the file that contains transactions in the library's billing system.

4. (a) What do you understand by the term cohesion and coupling in the context of software design? [4]

   Explain various types of cohesion and coupling with suitable examples. Explain the desired degree of cohesion and coupling for a good software design.

   (b) i) Perform Structured Analysis and Structured Design by using the operations of a simple lemonade stand. The list of activities are as given below: [4]

   Customer Order, Serve Product, Collect Payment,

   Produce Product, Store Product,

   Order Raw Materials, Pay for Raw Materials,

   Pay for Labour

   ii) Briefly discuss the significance of a data dictionary in structured analysis.

(Nilanjan Roy 1729143)

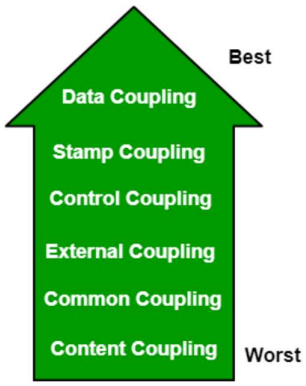
a)Modularization: Modularization is the process of dividing a software system into multiple independent modules where each module works independently. There are many advantages of Modularization in software engineering. Some of these are given below:

Easy to understand the system.

System maintenance is easy.

A module can be used many times as their requirements. No need to write it again and again.

Coupling: Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling.



Types of Coupling:

Data Coupling: If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent to each other and communicating through data. Module communications don't contain tramp data. Example-customer billing system.

Stamp Coupling In stamp coupling, the complete data structure is passed from one module to another module. Therefore, it involves tramp data. It may be necessary due to efficiency factors- this choice made by the insightful designer, not a lazy programmer.

Control Coupling: If the modules communicate by passing control information, then they are said to be control coupled. It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality. Example- sort function that takes comparison function as an argument.

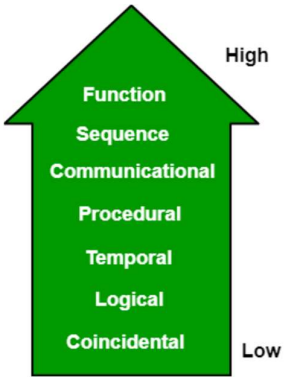
External Coupling: In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware. Ex- protocol, external file, device format, etc.

Common Coupling: The modules have shared data such as global data structures.The changes in global data mean tracing back to all modules which

access that data to evaluate the effect of the change. So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses and reduced maintainability.

Content Coupling: In a content coupling, one module can modify the data of another module or control flow is passed from one module to the other module. This is the worst form of coupling and should be avoided.

Cohesion: Cohesion is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. A good software design will have high cohesion.



Types of Cohesion:

Functional Cohesion: Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation.

Sequential Cohesion: An element outputs some data that becomes the input for other element, i.e., data flow between the parts. It occurs naturally in functional programming languages.

Communicational Cohesion: Two elements operate on the same input data or contribute towards the same output data. Example- update record int the database and send it to the printer.

Procedural Cohesion: Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable. Ex- calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA.

Temporal Cohesion: The elements are related by their timing involved. A module connected with temporal cohesion all the tasks must be executed in the same time-span. This cohesion contains the code for initializing all the parts of the system. Lots of different activities occur, all at init time.

Logical Cohesion: The elements are logically related and not functionally. Ex- A component reads inputs from tape, disk, and network. All the code for these functions is in the same component. Operations are related, but the functions are significantly different.

Coincidental Cohesion: The elements are not related(unrelated). The elements have no conceptual relationship other than location in source code. It is accidental and the worst form of cohesion. Ex- print next line and reverse the characters of a string in a single component.

B)A data dictionary is a structured repository of data elements in the system. It stores the descriptions of all DFD data elements that is, details and definitions of data flows, data stores, data stored in data stores, and the processes.

A data dictionary improves the communication between the analyst and the user. It plays an important role in building a database. Most DBMSs have a data dictionary as a standard feature

5. (a) What are the differences between Agile and Water fall model? Explain the scrum methodology in details. [4]

(b) Define a test case. Discuss about the following techniques to perform black box testing with suitable example. [4]

I. Equivalence Class Partitioning

II. Boundary value Analysis

(Nibedita Misra 1729141)

Ans.

A) Waterfall methodology is a software development methodology that is based on sequential-linear approach of software development. It reinforces the notion of "define before design" and "design before code".
Whereas agile is based on incremental-iterative approach where requirements are expected to change frequently.

	Waterfall	**Agile**
1.	Waterfall methodology is sequential and linear.	Agile methodology is incremental and iterative.
2.	Requirements have to be freezed at the beginning of SDLC.	Requirements are expected to change and changes are incorporated at any point.
3.	Working model of software is delivered at the later phases of SDLC.	Working model is delivered during initial phases and successive iteration of the model are delivered to the client for feedback.

4.	It is difficult to scale-up projects based on waterfall methodology.	Scaling up of products is easy because of the iterative approach.
5.	Customers or end user doesn't have a say after the requirements are freezed during the initial phases. They only get to know the product once it is build completely.	Frequent customer interaction and feedbacks are involved in agile methodology.
6.	Waterfall requires formalized documentations.	In agile documentation is often neglected and a working prototype serves as basis for customer's evaluation and feedback.
7.	Testing is performed once the software is build.	Continuous testing is performed during each iteration.

Scrum is an agile project management methodology or framework used primarily for software development projects with the goal of delivering new software capability every 2-4 weeks.

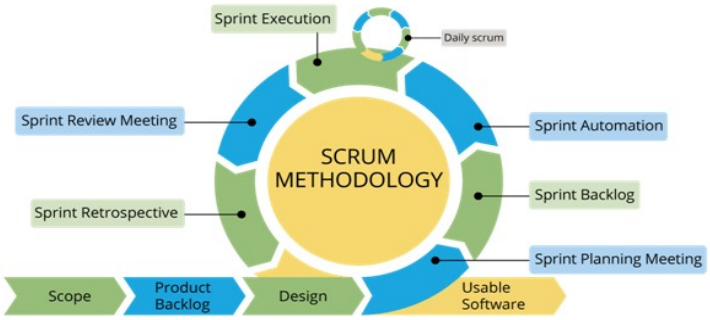
Scrum is widely used by software development teams. In fact it's the most popular agile methodology. According to the 12th annual State of Agile report, 70% of software teams use Scrum or a Scrum hybrid. However, Scrum has spread to other business functions including IT and marketing where there are projects that must move forward in the presence of complexity and ambiguity. Leadership teams are also basing their agile management practices on Scrum, often combining it with lean and Kanban practices.

Scrum is a sub-group of agile:

Agile is a set of values and principles that describe a group's day-to-day interactions and activities. Agile itself is not prescriptive or specific.The Scrum methodology follows the values and principles of agile, but includes further definitions and specifications, especially regarding certain software development practices.

Although developed for agile software development, agile Scrum became the preferred framework for agile project management in general and is sometimes simply referred to as Scrum project management or Scrum development.

Scrum addresses complexity in work by making information transparent, so that people can inspect and adapt based on current conditions, rather than predicted conditions. This allows teams to address the common pitfalls of a waterfall development process: chaos resulting from constantly changing requirements; underestimation of time, resources and cost; compromises on software quality; and inaccurate progress reporting. Transparency of common terms and standards is required in Scrum development to ensure that what is being delivered is what was expected.



B) A Test case is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly.The process of developing test cases can also help find problems in the requirements or design of an application.

Equivalence class partitioning (ECP) is a Software testing technique that divides the input data of a software unit into partitions of equivalent data from which test cases can be derived. In principle, test cases are designed to cover each partition at least once. This technique tries to define test cases that uncover classes of errors, thereby reducing the total number of test cases that must be developed. An advantage of this approach is reduction in the time required for testing software due to lesser number of test cases.

1) Equivalence partitioning is typically applied to the inputs of a tested component, but may be applied to the outputs in rare cases. The equivalence partitions are usually derived from the requirements specification for input attributes that influence the processing of the test object.

The fundamental concept of ECP comes from equivalence class which in turn comes from equivalence relation.This partitioning is called equivalence class partitioning of test input. If there are N equivalent classes, only N vectors are sufficient to fully cover the system.

2) Boundary value analysis is a software testing technique in which tests are designed to include representatives of boundary values in a range. The idea comes from the boundary. Given that we have a set of test vectors to test the system, a topology can be defined on that set. Those inputs which belong to the same equivalence class as defined by the equivalence partitioning theory would constitute the basis. Given that the basis sets are neighbors, there would exist a boundary between them. The test vectors on either side of the boundary are called boundary values. In practice this would require that the test vectors can be ordered, and that the individual parameters follows some kind of order.

6. (a) How Unified Modeling Language is helpful in object-oriented design? Draw a UML Sequence diagram and UML Activity diagram for "*Borrowing a Book from a library*" use case. [4]

(b) What is Cyclomatic Complexity?

(Sayahna Sengupta 1729158)

UML, short for Unified Modeling Language, is a standardized modeling language consisting of an integrated set of diagrams, developed to help system and software developers for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing object oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software. In this article, we will give you detailed ideas about what is UML, the history of UML and a description of each UML diagram type, along with UML examples.

In 1994, Jim Rumbaugh, the creator of OMT, stunned the software world when he left General Electric and joined Grady Booch at Rational Corp. The aim of the partnership was to merge their ideas into a single, unified method (the working title for the method was indeed the "Unified Method").

By 1995, the creator of OOSE, Ivar Jacobson, had also joined Rational, and his ideas (particularly the concept of "Use Cases") were fed into the new Unified Method - now called the Unified Modelling Language1. The team of Rumbaugh, Booch and Jacobson are affectionately known as the "Three Amigos"

UML has also been influenced by other object-oriented notations:

- Mellor and Shlaer [1998]
- Coad and Yourdon [1995]
- Wirfs-Brock [1990]
- Martin and Odell [1992]

UML also includes new concepts that were not present in other major methods at the time, such as extension mechanisms and a constraint language.

Why UML

As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software and to improve quality and reduce cost and time-to-market. These techniques include component technology, visual programming, patterns and frameworks. Businesses also seek techniques to manage the complexity of systems as they increase in scope and scale. In particular,

they recognize the need to solve recurring architectural problems, such as physical distribution, concurrency, replication, security, load balancing and fault tolerance. Additionally, the development for the World Wide Web, while making some things simpler, has exacerbated these architectural problems. The Unified Modeling Language (UML) was designed to respond to these needs. The primary goals in the design of the UML summarize by Page-Jones in Fundamental Object-Oriented Design in UML as follows:

1. Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
2. Provide extensibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development processes.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of the OO tools market.
6. Support higher-level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

UML - An Overview

Before we begin to look at the theory of the UML, we are going to take a very brief run through some of the major concepts of the UML.

The first thing to notice about the UML is that there are a lot of different diagrams (models) to get used to. The reason for this is that it is possible to look at a system from many different viewpoints. A software development will have many stakeholders playing a part.

For Example:

- Analysts
- Designers
- Coders
- Testers
- QA
- The Customer
- Technical Authors

All of these people are interested in different aspects of the system, and each of them require a different level of detail. For example, a coder needs to understand the design of the system and be able to convert the design to a low level code. By contrast, a technical writer is interested in the behavior of the system as a whole, and needs to understand how the product functions. The UML attempts to provide a language so expressive that all stakeholders can benefit from at least one UML diagram.

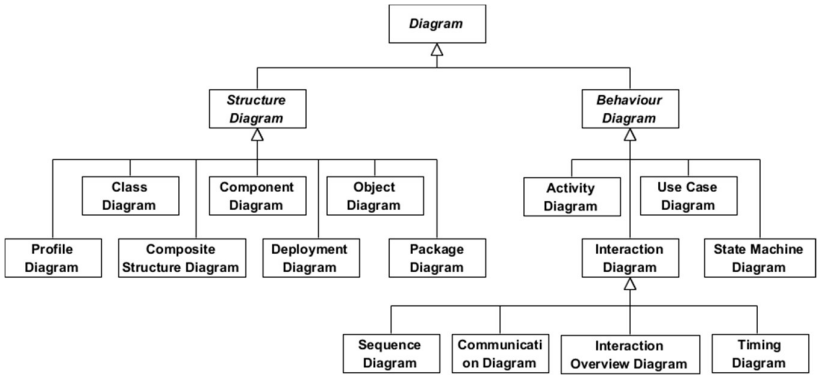
Here's a quick look at each one of these 13 diagrams in as shown in the UML 2 Diagram Structure below:

Structure diagrams show the static structure of the system and its parts on different abstraction and implementation levels and how they are related to each other. The elements in a structure diagram represent the meaningful concepts of a system, and may include abstract, real world and implementation concepts, there are seven types of structure diagram as follows:

- Class Diagram
- Component Diagram
- Deployment Diagram
- Object Diagram
- Package Diagram
- Composite Structure Diagram
- Profile Diagram

Behavior diagrams show the **dynamic behavior** of the objects in a system, which can be described as a series of changes to the system over **time**, there are seven types of behavior diagrams as follows:

- Use Case Diagram
- Activity Diagram
- State Machine Diagram
- Sequence Diagram
- Communication Diagram
- Interaction Overview Diagram
- Timing Diagram



What is a Class Diagram?

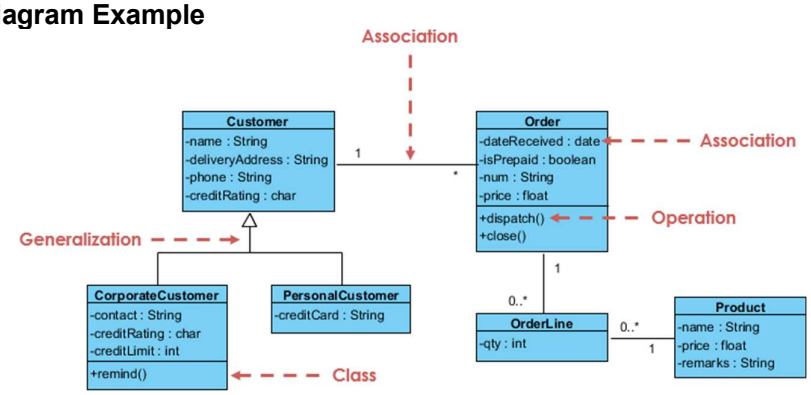
The class diagram is a central modeling technique that runs through nearly all object-oriented methods. This diagram describes the types of objects in the system and various kinds of static relationships which exist between them.

Relationships

There are three principal kinds of relationships which are important:

1. **Association** - represent relationships between instances of types (a person works for a company, a company has a number of offices.
2. **Inheritance** - the most obvious addition to ER diagrams for use in OO. It has an immediate correspondence to inheritance in OO design.
3. **Aggregation** - Aggregation, a form of object composition in object-oriented design.

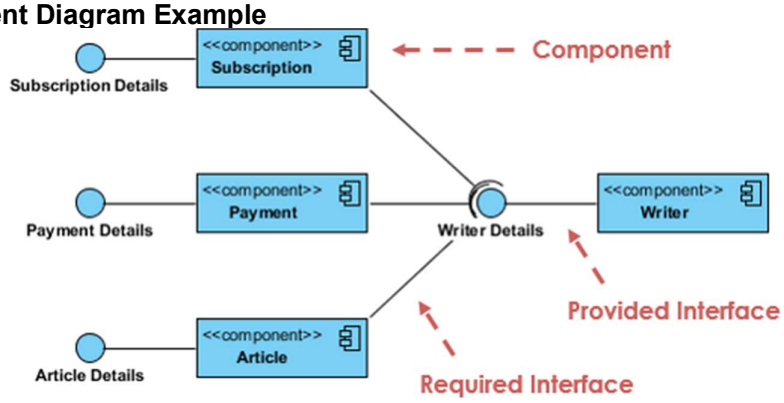
Class Diagram Example



What is Component Diagram?

In the Unified Modeling Language, a component diagram depicts how components are wired together to form larger components or software systems. It illustrates the architectures of the software components and the dependencies between them. Those software components including run-time components, executable components also the source code components.

Component Diagram Example



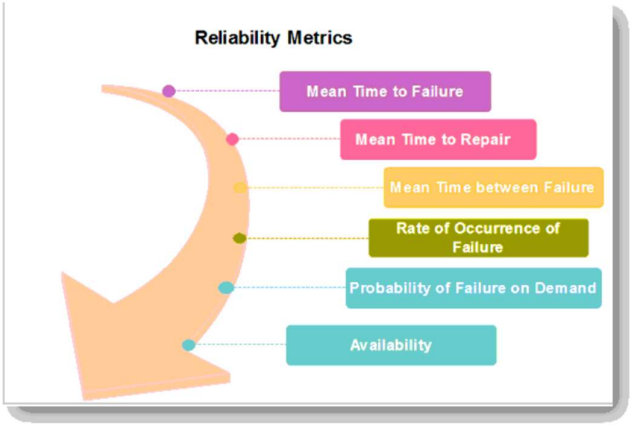
7. (a) Briefly explain the various reliability metrics of software products. [4]

   (b) How software reverse engineering is different from software re-engineering? Discuss about the different cosmetic changes made during the software reverse engineering. [4]

(Swadesh Nayak 1729085)

a)Reliability metrics are used to quantitatively expressed the reliability of the software product. The option of which metric is to be used depends upon the type of system to which it applies & the requirements of the application domain.

Some reliability metrics which can be used to quantify the reliability of the software product are as follows:



1. Mean Time to Failure (MTTF)

MTTF is described as the time interval between the two successive failures. An MTTF of 200 mean that one failure can be expected each 200-time units. The time units are entirely dependent on the system & it can even be stated in the number of transactions. MTTF is consistent for systems with large transactions.

2. Mean Time to Repair (MTTR)

Once failure occurs, some-time is required to fix the error. MTTR measures the average time it takes to track the errors causing the failure and to fix them.

3. Mean Time Between Failure (MTBR)

We can merge MTTF & MTTR metrics to get the MTBF metric.

$$MTBF = MTTF + MTTR$$

Thus, an MTBF of 300 denoted that once the failure appears, the next failure is expected to appear only after 300 hours. In this method, the time measurements are real-time & not the execution time as in MTTF.

4. Rate of occurrence of failure (ROCOF)

It is the number of failures appearing in a unit time interval. The number of unexpected events over a specific time of operation. ROCOF is the frequency of occurrence with which unexpected role is likely to appear. A ROCOF of 0.02 mean that two failures are likely to occur in each 100 operational time unit steps. It is also called the failure intensity metric.

5. Probability of Failure on Demand (POFOD)

POFOD is described as the probability that the system will fail when a service is requested. It is the number of system deficiency given several systems inputs.

POFOD is the possibility that the system will fail when a service request is made.

6. Availability (AVAIL)

Availability is the probability that the system is applicable for use at a given time. It takes into account the repair time & the restart time for the system. An availability of 0.995 means that in every 1000 time units, the system is feasible to be available for 995 of these. The percentage of time that a system is applicable for use, taking into account planned and unplanned downtime. If a system is down an average of four hours out of 100 hours of operation, its AVAIL is 96%.

B)Re-Engineering

Re-engineering is the investigation and redesign of individual components. It may also describe the entire overhaul of a device by taking the current design and improving certain aspects of it. The aims of re-engineering may be to improve a particular area of performance or functionality, reduce operational costs or add new elements to a current design. The methods used depend on the device but typically involve engineering drawings of the amendments followed by extensive testing of prototypes before production. The rights to re-engineer a product belong solely to the original owner of the design or relevant patent.

Reverse Engineering

Unlike re-engineering, reverse engineering takes a finished product with the aim of discovering how it works by testing it. Typically this is done by companies that seek to infiltrate a competitor's market or understand its new product. In doing so they can produce new products while allowing the original creator to pay all the development costs and take all the risks involved with creating a new product. Analysis of a product in this way is done without technical drawings or prior knowledge of how the device works, and the basic method used in reverse engineering begins by identifying the system's components, followed by an investigation into the relationship among these components.

8. Write short notes on any TWO of the following: [4×2]

(a) Verification and Validation.

(b) Pert Chart and Gantt Chart

(c) UML Diagrams

(d) Top down and Bottom Up Integration Testing

(Nibedita Misra 1729141)

(B)Gantt chart and Pert chart

SNO	Gantt Chart	PERT Chart
01.	Represented with Bar Chart	Represented with Flowchart (or Network Diagram)
02.	Used for Small Projects	Used For Large and Complex Projects
03.	Provide Accurate Time Duration and Percent Complete	Need to Predict the Time
04.	Cannot Display Interconnecting Tasks That Depends on Each Other	Has Numerous Interconnecting Networks of Independent Tasks
05.	The Gantt chart was first developed and introduced by Charles Gantt in 1917.	Program Evaluation and Review Technique charts were developed and introduced in 1950 by the U.S. Navy.
06.	Gantt charts are straightforward and are not made for projects which need changes	PERT charts are complex and are made for small portions of the project.
07.	**Example :** Gantt charts can be created using specific project management software such as Smart sheet.	**Example :** PERT chart can make with Smart Draw
08.	Gantt charts focus on the time required to complete a task	whereas a PERT chart focuses on intertask relationships.

(d)TOP DOWN AND BOTTOM UP INTEGRATION TESTING

- The **top-down integration testing** is an incremental technique of building a program structure. It incorporates the modules while moving downward, beginning with the main control in the hierarchy. Sub-modules are then integrated to the main module using either a depth-first or breadth-first method. The main purpose of top-down integration is to verify the significant control and decision points earlier in the test process.Integration process involves the following steps in the top-down approach:

- Starting with the major control module, stubs are then replaced for the components residing below the main modules.
- The replacement strategy of the subordinate stub relies on the type of integration approach followed (i.e., depth and breadth first), but only one stub is allowed to be replaced with actual components at a time.
- After the integration of the components, the tests are carried out.
- As a set of test is accomplished, the remaining stub is replaced with the actual component.
- In the end, the regression test is conducted to assure the absence of the new errors.

Bottom-up Integration Testing

The **bottom-up integration testing** starts with the construction of the fundamental modules (i.e., lowest level program elements). It integrates the components residing at the lowest level (i.e., lowest level) by providing a process and eliminates the need of the stubs. As the integration goes towards the upper direction, the requirement of the separate test drivers decreases. Hence, the amount of overhead is also reduced as compared to Top-bottom integration testing approach.

Bottom-up integration includes the following steps:

- It merges the low-level elements also known as builds into clusters which execute a certain software subfunction.
- The driver (Control program) is used in the bottom-up integration to arrange test case input and output.
- Then the cluster is tested.
- Clusters are incorporated while going upwardly in the program structure and drivers are eliminated.