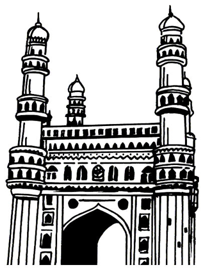
Rahul's ✔
Topper's Voice



M.C.A.

II Year III Sem

(Osmania University)

Latest 2024 Edition

SOFTWARE ENGINEERING

- **Study Manual**
- **Important Questions**
- **Solved Model Papers**
- **Solved Previous Question Papers**

Price
` . 199-00

- by -

WELL EXPERIENCED LECTURER

Rahul Publications™

Hyderabad. Cell : 9391018098, 9505799122

All disputes are subjects to Hyderabad Jurisdiction only

M.C.A.

II Year III Sem

(Osmania University)

SOFTWARE ENGINEERING

Price `. 199 -00

Sole Distributors : **Cell : 9391018098, 9505799122**

VASU BOOK CENTRE

Shop No. 2, Beside Gokul Chat, Koti, Hyderabad.

Maternity Hospital Opp. Lane, Narayan Naik Complex, Koti, Hyderabad.
Near Andhra Bank, Subway, Sultan Bazar, Koti, Hyderabad -195.

C O N T E N T S

SOFTWARE ENGINEERING

Contents

Important Questions

UNIT - I

1. What are the Challenges faced by Software Engineering?

Ans :

Refer Unit-I, Q.No. 2

2. Explain Software Engineering Process.

Ans :

Refer Unit-I, Q.No. 6

3. Explain Prototype Model and Iterative Development Model.

Ans :

Refer Unit-I, Q.No. 10

4. Explain project management process.

Ans :

Refer Unit-I, Q.No. 14

UNIT - II

1. What are Software Requirements Specification and Describe various characteristics of software require-ments specification?

Ans :

Refer Unit-II, Q.No. 1

2. Explain about Requirement Process.

Ans :

Refer Unit-II, Q.No. 4

3. What is Software Requirements Specification? Explain Structure and Characteristics of SRS.

Ans :

Refer Unit-II, Q.No. 6

4. Discuss briefly the validation of SRS.

Ans :

Refer Unit-II, Q.No. 9

5. Discuss about functional specifications using use cases.

Ans :

Refer Unit-II, Q.No. 11

6. Explain different approaches of Analysis in Software Engineering.

Ans :

Refer Unit-II, Q.No. 13

7. Explain about software architecture state its characteristics.

Ans :

Refer Unit-II, Q.No. 14

8. What are the different views of Software Architecture?

Ans :

Refer Unit-II, Q.No. 16

9. How would you build software archi-tecture document?

Ans :

Refer Unit-II, Q.No. 18

UNIT - III

1. Explain about Effort and Schedule estimates of Software Projects.

Ans :

Refer Unit-III, Q.No. 2

2. What is Staffing? Explain Project Schedule.

Ans :

Refer Unit-III, Q.No. 3

3. What is Risk Management in Software Projects? Give brief ideas of Risk Assessment and Control.

Ans :

Refer Unit-III, Q.No. 5

4. Explain the concept of detailed schedu- ling.

Ans :

Refer Unit-III, Q.No. 7

5. What are the strategies followed in designing software?

Ans :

Refer Unit-III, Q.No. 10

6. What are metrics, Measurements & Models of Project Management & Software Management?

Ans :

Refer Unit-III, Q.No. 12

UNIT - IV

1. Explain various Programming Practices used in Coding. What is meant by Information Hiding?

Ans :

Refer Unit-IV, Q.No. 1

2. State the Advantages of Coding Guidelines.

Ans :

Refer Unit-IV, Q.No. 5

3. What are the metrics of Coding?

Ans :

Refer Unit-IV, Q.No. 7

4. Explain in detail the Code verification Techniques in Software Engineering.

Ans :

Refer Unit-IV, Q.No. 8

5. Describe white box testing in software engineering.

Ans :

Refer Unit-IV, Q.No. 10

6. Explain Black Box Testing in detail.

Ans :

Refer Unit-IV, Q.No. 11

7. What are the differences between White Box and Black Box Testing?

Ans :

Refer Unit-IV, Q.No. 12

8. What are Software Testing Metric?

Ans :

Refer Unit-IV, Q.No. 13

UNIT - V

1. What is Software Maintenance? Describe the types of Software Mainta-nance.

Ans :

Refer Unit-V, Q.No. 1

2. What is Re-engineering? Describe steps followed by Reengineering.

Ans :

Refer Unit-V, Q.No. 2

3. Explain the concept of Software Re-engineering.

Ans :

Refer Unit-V, Q.No. 6

4. What is forward engineering in software?

Ans :

Refer Unit-V, Q.No. 9

5. Explain the concept of Economics of Reengineering.

Ans :

Refer Unit-V, Q.No. 10

6. Discuss in detail about Process and Product Quality In Software Engineering.

Ans :

Refer Unit-V, Q.No. 11

7. Describe in detail the frame work of PCCM.

Ans :

Refer Unit-V, Q.No. 16

8. Explain in detail about Ideal Model and Spice in SPI.

Ans :

Refer Unit-V, Q.No. 17

UNIT I

The software Problem: Cost, Schedule and Quality, Scale and change, **Software Processes:** Process and project, Component Software Processes, Software Development Process Models, Project management Process.

1.1 The Software Problem

Q1. Define Software.

Ans :

- The problem domain for software engineering is industrial-strength software. This software is meant to solve some problem of some set of users, and is expected to be of high quality.
- In this problem domain, cost, schedule, and quality are basic driving forces. Hence, methods and tools that will be used for solving problems in this domain must ensure high productivity and high quality.
- Productivity is measured as amount of output per unit of input resource. In software, output can be measured in terms of lines of code delivered, and as human time is the main resource, input can be measured as person-months. Productivity can therefore be measured as lines of code delivered per person-month.
- Software quality has many attributes which include functionality, reliability, usability, efficiency, maintainability, and portability. Reliability is often considered as the main quality attribute, and as unreliability in software is due to defects in the software, quality can be characterized by number of defects per thousand lines of code.
- The problems in this domain often tend to be very large and where the needs of the customers change fast. Hence the techniques used for developing industrial-strength software should be such that they are capable of building large software systems, and have the capability to handle changes.

Q2. What are the Challenges faced by Software Engineering?

Ans : **(Imp.)**

Software engineering employs a well-defined and systematic approach to develop software. This approach is considered to be the most effective way of producing high-quality software. However, despite this systematic approach in software development, there are still some serious challenges faced by software engineering.

Some of these challenges are listed below.

1. The methods used to develop small or medium-scale projects are not suitable when it comes to the development of large-scale or complex systems.
2. Changes in software development are unavoidable. In today's world, changes occur rapidly and accommodating these changes to develop complete software is one of the major challenges faced by the software engineers.

3. The advancement in computer and software technology has necessitated for the changes in nature of software systems. The software systems that cannot accommodate changes are not of much use. Thus, one of the challenges of software engineering is to produce high quality software adapting to the changing needs within acceptable schedules. To meet this challenge, the object oriented approach is preferred, but accommodating changes to software and its maintenance within acceptable cost is still a challenge.

4. Informal communications take up a considerable portion of the time spent on software projects. Such wastage of time delays the completion of projects in the specified time.

5. The user generally has only a vague idea about the scope and requirements of the software system. This usually results in the development of software, which does not meet the user's requirements.

6. Changes are usually incorporated in documents without following any standard procedure. Thus, verification of all such changes often becomes difficult.

7. The development ofl1igh-quality and reliable software requires the software to be thoroughly tested. Though thorough testing of software consumes the majority of resources, underestimating it because of any reasons deteriorates the software quality.

8. In addition to the above mentioned key challenges, the responsibilities of the system analyst, designers, and programmers are usually not well defined. Also, if the user requirements are not precisely defined, software developers can misinterpret the meaning. All these challenges need to be addressed in order to ensure that the software is developed within the specified time and estimated costs and also meets the requirements specified by the user.

1.2 Cost, Schedule, and Quality, Scale and Change

Q3. What are various Software Engineering Problems? Explain.

(OR)

Discuss about

(i) Cost (ii) Schedule

(iii) Qualify (iv) Scale and Change

Ans :

Though the need for high quality distinguishes industrial strength software from others, cost and schedule are other major driving forces for such software. In the industrial-strength software domain, there are three basic forces at play cost, schedule, and quality. The software should be produced at reasonable cost, in a reasonable time, and should be of good quality. These three parameters often drive and define a software project.

(i) Cost

Industrial-strength software is very expensive primarily due to the fact that software development is extremely labor-intensive. To get an idea of the costs involved, let us consider the current state of practice in the industry. Lines of code (LOC) or thousands of lines of code (KLOC) delivered is by far the most commonly used measure of software size in the industry. As the main cost of producing software is the manpower employed, the cost of developing software is generally measured in terms of person-months of effort spent in development. And productivity is frequently measured in the industry in terms of LOC (or KLOC) per person-month.

The productivity in the software industry for writing fresh code generally ranges from few hundred to about 1000+ LOC per person-month. This productivity is over the entire development cycle, not just the coding task. Software companies often charge the client for whom they are developing the software between \$3000 - \$15,000 per person-month. With a productivity of 1000 LOC per person-month, it means that each line of delivered code costs between $3 and $15! And even small projects can easily end up with software of 50,000 LOC. With this productivity, such a software project will cost between $150,000 and $750,000!

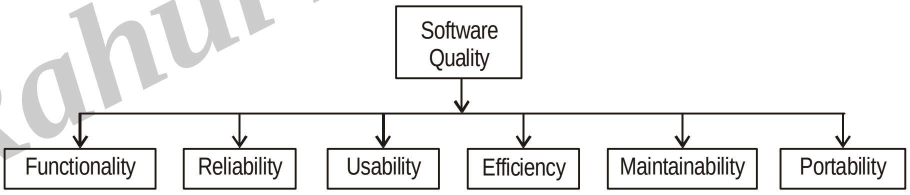
(ii) Schedule

Schedule is another important factor in many projects. Business trends are dictating that the time to market of a product should be reduced; that is, the cycle time from concept to delivery should be small. For software this means that it needs to be developed faster, and within the specified time. Unfortunately, the history of software is full of cases where projects have been substantially late.

Clearly, therefore, reducing the cost and the cycle time for software development are central goals of software engineering. Productivity in terms of output (KLOC) per person-month can adequately capture both cost and schedule concerns. If productivity is higher, it should be clear that the cost in terms of person-months will be lower (the same work can now be done with fewer person-months). Similarly, if productivity is higher, the potential of developing the software in less time improves—a team of higher productivity will finish a job in less time than a same-size team with lower productivity. (The actual time the project will take, of course, depends also on the number of people allocated to the project.) Hence, pursuit of higher productivity is a basic driving force behind software engineering and a major reason for using the different tools and techniques.

(iii) Qualify

Besides cost and schedule, the other major factor driving software engineering is quality. Today, quality is one of the main mantras, and business strategies are designed around it. Unfortunately, a large number of instances have occurred regarding the unreliability of software the software often does not do what it is supposed to do or does something it is not supposed to do. Clearly, developing high-quality software is another fundamental goal of software engineering. However, while cost is generally well understood, the concept of quality in the context of software needs further elaboration. The international standard on software product quality suggests that software quality comprises six main attributes.



These attributes can be defined as follows :

1. **Functionality:** The capability to provide functions which meet stated and implied needs when the software is used.
2. **Reliability:** The capability to provide failure-free service.
3. **Usability:** The capability to be understood, learned, and used.
4. **Efficiency:** The capability to provide appropriate performance relative to the amount of resources used.
5. **Maintainability:** The capability to be modified for purposes of making corrections, improvements, or adaptation.
6. **Portability:** The capability to be adapted for different specified environments without applying actions or means other than those provided for this purpose in the product.

With multiple dimensions to quality, different projects may emphasize different attributes, and a global single number for quality is not possible. However, despite the fact that there are many quality attributes, reliability is generally accepted to be the main quality criterion. As unreliability of software is due to the presence of defects in the software, one measure of quality is the number of defects in the delivered software per unit size (generally taken to be thousands of lines of code, or KLOC). With this as the major quality criterion, the quality objective is to reduce the number of defects per KLOC as much as possible. Current best practices in software engineering have been able to reduce the defect density to less than 1 defect per KLOC.

To determine the quality of a software product, we need to determine the number of defects in the software that was delivered. This number is clearly not known at delivery time and may never be known. One approach to measure quality is to log the defects found in 6 months (or 1 year) after delivery and define quality with respect to these defects. This means that quality of delivered software can only be determined 6 months after its delivery. The defect density can, however, also be estimated from past data of similar projects—if similar approaches are being used, then it is expected that the current project will have similar defect density as the past projects. It should be pointed out that to use this definition of quality, what a defect is must be clearly defined. A defect could be some problem in the software that causes the software to crash or a problem that causes an output to be not properly aligned or one that misspells some word, etc. The exact definition of what is considered a defect will clearly depend on the project or the standards the organization developing the project uses (typically it is the latter).

Besides reliability, another quality attribute of great interest is maintainability. Once the software is delivered and deployed, it enters the maintenance phase. Why is maintenance needed for software, when software has no physical components that can degrade with age? Software needs to be maintained because of the residual defects remaining in the system. It is commonly believed that the state of the art today is limited and developing software with zero defect density is not possible. These defects, once discovered, need to be removed, leading to what is called corrective maintenance. Maintenance is also needed to change the delivered software to satisfy the enhanced needs of the users and the environment, leading to adaptive maintenance. Over the life of a software system, maintenance cost can far exceed the cost of original development. The maintenance-to-development-cost ratio has been variously suggested as 80:20, 70:30, or 60:40. Due to this high cost, maintainability attribute of delivered software is of high interest—it is clearly desirable to have software systems that are easier to maintain.

(iv) Scale and Change

Though cost, schedule, and quality are the main driving forces for a project in our problem domain (of industry strength software), there are some other characteristics of the problem domain that also influence the solution approaches employed. We focus on two such characteristics—scale and change.

Most industrial-strength software systems tend to be large and complex, requiring tens of thousands of lines of code. Sizes of some of the well-known software products are given in Table. As can be expected, development of a large system requires a different set of methods compared to developing a small system, as the methods that are used for developing small systems often do not scale up to large systems.

An example will illustrate this point. Consider the problem of counting people in a room versus taking a census of a country. Both are essentially counting problems. But the methods used for counting people in a room will just not work when taking a census. A different set of methods will have to be used for conducting a census, and the census problem will require considerably more management, organization, and validation, in addition to counting.

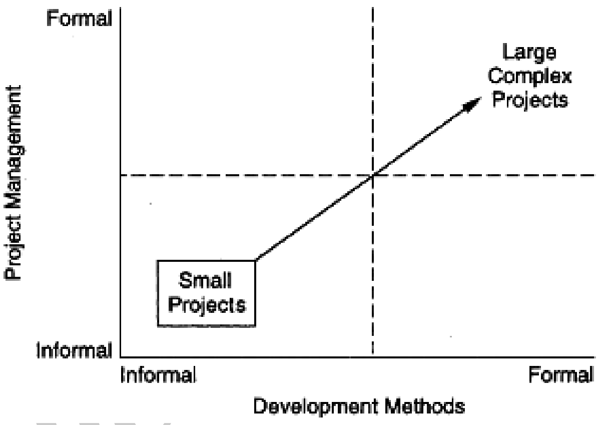
Size (KLOC)	Software	Languages
980	gcc	ansic, cpp, yacc
320	perl	perl, ansic, sh
200	openssl	ansic, cpp, perl
100	apache	ansic, sh
65	sendmail	ansic
30,000	Red Hat Limax	ansic, cpp
40,000	Windows XP	ansic, cpp

Similarly, methods that one can use to develop programs of a few hundred lines cannot be expected to work when software of a few hundred thousand lines needs to be developed. A different set of methods must be used for developing large software.

Any software project involves the use of engineering and project management. In small projects, informal methods for development and management can be used. However, for large projects, both have to be much more rigorous. In other words, to successfully execute a project, a proper method for engineering the system has to be employed and the project has to be tightly managed to make sure that cost, schedule, and quality are under control.

Large scale is a key characteristic of the problem domain and the solution approaches should employ tools and techniques that have the ability to build large software systems. Change is another characteristic of the problem domain which the approaches for development must handle. As the complete set of requirements for the system is generally not known (often cannot be known at the start of the project) or stated, as development proceeds and time passes, additional requirements are identified, which need to be incorporated in the software being developed. This need for changes requires that methods for development embrace change and accommodate it efficiently. Change requests can be quite disruptive to a project, and if not handled properly, can consume up to 30 to 40% of the development cost.

Software has to be changed even after it has been deployed. Though traditionally changes in software during maintenance have been distinguished from changes that occur while the development is taking place, these lines are blurring, as fundamentally the changes in both of these scenarios are similar-existing source code needs to be changed due to some changes in the requirements or due to some defects that need to be removed.



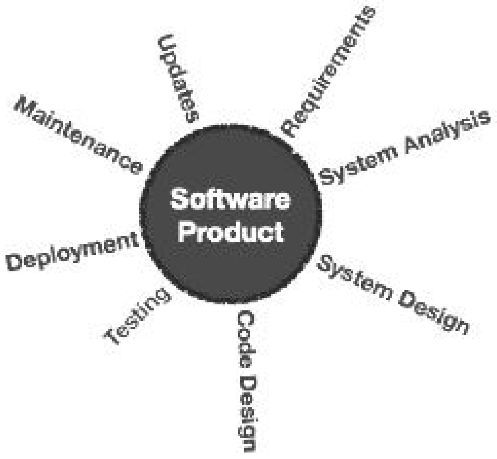
Overall, as the world changes faster, software has to change faster, even while under development. Changes in requirements are therefore a characteristic of the problem domain. In today's world, approaches that cannot accept and accommodate change are of little use—they can solve only those few problems that are change resistant.

Q4. What is Software Engineering and Explain Evolution of Software Enginee-ring?

Ans :

(i) Software: Is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called software product.

(ii) Engineering: On the other hand, is all about developing products, using well-defined, scientific principles and methods.



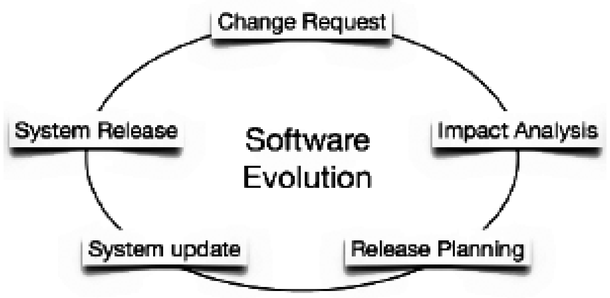
Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

Definitions

1. The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software.
2. The study of approaches as in the above statement.
3. Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and work efficiently on real machines.

Evolution

The process of developing a software product using software engineering principles and methods is referred to as software evolution. This includes the initial development of software and its maintenance and updates, till desired software product is developed, which satisfies the expected requirements.



Evolution starts from the requirement gathering process. After which developers create a prototype of the intended software and show it to the users to get their feedback at the early stage of software product development. The users suggest changes, on which several consecutive updates and maintenance keep on changing too. This process changes to the original software, till the desired software is accomplished.

Even after the user has desired software in hand, the advancing technology and the changing requirements force the software product to change accordingly. Re-creating software from scratch and to go one-on-one with requirement is not feasible. The only feasible and economical solution is to update the existing software so that it matches the latest requirements.

Software Evolution Laws

Lehman has given laws for software evolution. He divided the software into three different categories:

➢ **S-type (static-type):** This is a software, which works strictly according to defined specifications and solutions. The solution and the method to achieve it, both are immediately understood before coding. The s-type software is least subjected to changes hence this is the simplest of all. For example, calculator program for mathematical computation.

➢ **P-type (practical-type):** This is a software with a collection of procedures. This is defined by exactly what procedures can do. In this software, the specifications can be described but the solution is not obvious instantly. For example, gaming software.

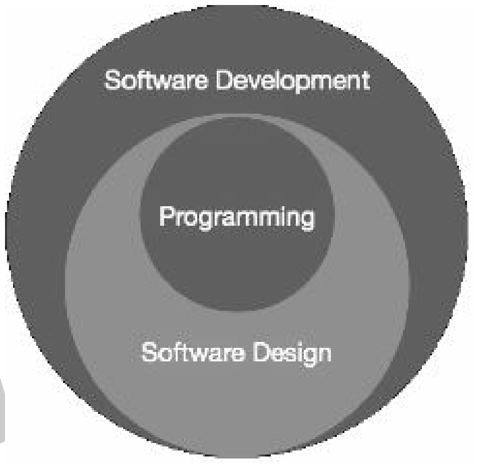
➢ **E-type (embedded-type):** This software works closely as the requirement of real-world environment. This software has a high degree of evolution as there are various changes in laws, taxes etc. in the real world situations. For example, Online trading software.

- E-Type software evolution
- Lehman has given eight laws for E-Type software evolution -
- **Continuing change:** An E-type software system must continue to adapt to the real world changes; else it becomes progressively less useful.
- **Increasing complexity -** As an E-type software system evolves, its complexity tends to increase unless work is done to maintain or reduce it.
- **Conservation of familiarity:** The familiarity with the software or the knowledge about how it was developed, why was it developed in that particular manner etc. must be retained at any cost, to implement the changes in the system.
- **Continuing growth:** In order for an E-type system intended to resolve some business problem, its size of implementing the changes grows according to the lifestyle changes of the business.
- **Reducing quality:** An E-type software system declines in quality unless rigorously maintained and adapted to a changing operational environment.
- **Feedback systems:** The E-type software systems constitute multi-loop, multi-level feedback systems and must be treated as such to be successfully modified or improved.
- **Self-regulation:** E-type system evolution processes are self-regulating with the distribution of product and process measures close to normal.
- **Organizational stability:** The average effective global activity rate in an evolving E-type system is invariant over the lifetime of the product.

Q5. Explain the Paradigm of Software Engineering and needs.

Ans :

Software paradigms refer to the methods and steps, which are taken while designing the software. There are many methods proposed and are in work today, but we need to see where in the software engineering these paradigms stand. These can be combined into various categories, though each of them is contained in one another :



Programming paradigm is a subset of Software design paradigm which is further a subset of Software development paradigm.

Software Development Paradigm

This Paradigm is known as software engineering paradigms where all the engineering concepts pertaining to the development of software are applied. It includes various researches and requirement gathering which helps the software product to build. It consists of

- Requirement gathering
- Software design
- Programming

Software Design Paradigm

This paradigm is a part of Software Development and includes:

- Design
- Maintenance
- Programming

Programming Paradigm

This paradigm is related closely to programming aspect of software development. This includes:

- Coding
- Testing
- Integration

Need

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

- **Large software:** It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.
- **Scalability:** If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.
- **Cost:** As hardware industry has shown its skills and huge manufacturing has lower down the price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.
- **Dynamic Nature:** The always growing and adapting nature of software hugely depends upon the environment in which user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.
- **Quality Management:** Better process of software development provides better and quality software product.

Characteristics

A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

- Operational
- Transitional
- Maintenance

Well-engineered and crafted software is expected to have the following characteristics:

Operational

This tells us how well software works in operations. It can be measured on:

- Budget
- Usability
- Efficiency
- Correctness
- Functionality
- Dependability
- Security
- Safety

Transitional

This aspect is important when the software is moved from one platform to another :

- Portability
- Interoperability
- Reusability
- Adaptability

Maintenance

This aspect briefs about how well a software has the capabilities to maintain itself in the ever-changing environment :

- Modularity
- Maintainability
- Flexibility
- Scalability

In short, Software engineering is a branch of computer science, which uses well-defined engineering concepts required to produce efficient, durable, scalable, in-budget and on-time software products.

1.3 Software Processes

Q6. Explain Software Engineering Process.

Ans : **(Imp.)**

Software engineering is defined as the systematic approach to the development, operation, maintenance, and retirement of software.

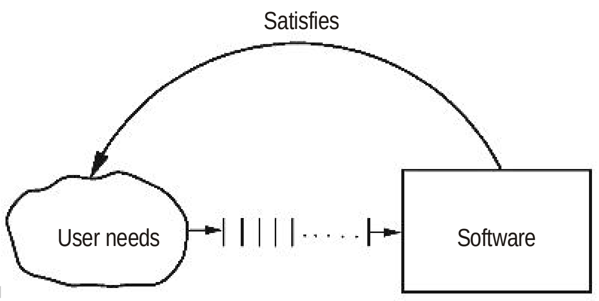
In other words, the systematic approach must help achieve a high quality and productivity (Q&P). In software, the three main factors that influence Q&P are people, processes, and technology. That is, the final quality delivered and productivity achieved depends on the skills of the people involved in the software project, the processes people use to perform the different tasks in the project, and the tools they use.

As it is people who ultimately develop and deliver (and productivity is measured with respect to people's effort as the basic input), the main job of processes is to help people achieve higher Q&P by specifying what tasks to do and how to do them. Tools are aids that help people perform some of the tasks more efficiently and with fewer errors. It should therefore be clear that to satisfy the objective of delivering software with high Q&P, processes form the core. Consequently, in software engineering, the focus is primarily on processes, which are referred to as the systematic approach in the definition given above. It is this focus on process that distinguishes software engineering from most other computing disciplines. Many other computing disciplines focus on some type of product operating systems, databases, etc., while software engineering focuses on the process for producing the products.

As processes form the heart of software engineering, with tools and technology providing support to efficiently execute the processes, this book focuses primarily on processes.

- Role of a process and a process model in a project.
- Various component processes in the software process and the key role of the development process and the project management process.
- Various models for the development process—waterfall, prototyping, iterative, RUP, time boxing, and XP.
- The overall structure of the project management process and its key phases.
- Process and Project

A process is a sequence of steps performed for a given purpose. As mentioned earlier, while developing (industrial strength) software, the purpose is to develop software to satisfy the needs of some users or clients. A software project is one instance of this problem, and the development process is what is used to achieve this purpose.



So, for a project its development process plays a key role—it is by following the process the desired end goal of delivering the software is achieved. However, as discussed earlier, it is not sufficient to just reach the final goal of having the desired software, but we want that the project be done at low cost and in low cycle time, and deliver high-quality software. The role of process increases due to these additional goals, and though many processes can achieve the basic goal of developing software, to achieve high Q&P we need some "optimum" process. It is this goal that makes designing a process a challenge.

We must distinguish process specification or description from the process itself. A process is a dynamic entity which captures the actions performed. Process specification, on the other hand, is a description of process which presumably can be followed in some project to achieve the goal for which the process is designed.

In a project, a process specification may be used as the process the project plans to follow. The actual process is what is actually done in the project. Note that the actual process can be different from the planned process, and ensuring that the specified process is being followed is a nontrivial problem. However, in this book, we will assume that the planned and actual processes are the same and will not distinguish between the two and will use the term process to refer to both.

A process model specifies a general process, which is "optimum" for a class of projects. That is, in the situations for which the model is applicable, using the process model as the project's process will lead to the goal of developing software with high Q&P. A process model is essentially a compilation of best practices into a "recipe" for success in the project. In other words, a process is a means to reach the goals of high quality, low cost, and low cycle time, and a process model provides a process structure that is well suited for a class of projects.

A process is often specified at a high level as a sequence of stages. The sequence of steps for a stage is the process for that stage, and is often referred to as a sub process of the process.

1.3.1 Component Software Processes

Q7. What are the Components of Software Process?

Ans :

A process is the sequence of steps executed to achieve a goal. Since many different goals may have to be satisfied while developing software, multiple processes are needed. Many of these do not concern software engineering, though they do impact software development. These could be considered non-software process. Business processes, social processes, and training processes are all examples of processes that come under this. These processes also affect the software development activity but are beyond the purview of software engineering.

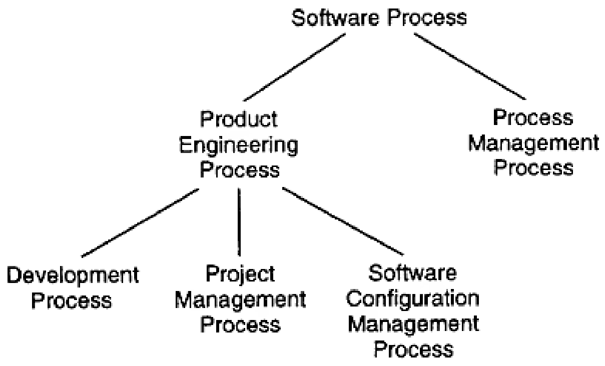
The processes that deal with the technical and management issues of software development are collectively called the software process. As a software project will have to engineer a solution and properly manage the project, there are clearly two major components in a software process a development process and a project management process. The development process specifies all the engineering activities that need to be performed, whereas the management process specifies how to plan and control these activities so that cost, schedule, quality, and other objectives are met. Effective development and project management processes are the key to achieving the objectives of delivering the desired software satisfying the user needs, while ensuring high productivity and quality.

During the project many products are produced which are typically composed of many items (for example, the final source code may be composed of many source files). These items keep evolving as the project proceeds, creating many versions on the way. As development processes generally do not focus on evolution and changes, to handle them another process called software configuration control process is often used. The objective of this component process is to primarily deal with managing change, so that the integrity of the products is not violated despite changes.

These three constituent processes focus on the projects and the products and can be considered as comprising the product engineering processes, as their main objective is to produce the desired product. If the software process can be viewed as a static entity, then these three component processes will suffice. However, a software process itself is a dynamic entity, as it must change to adapt to our increased understanding about software development and availability of newer technologies and tools. Due to this, a process to manage the software process is needed.

The basic objective of the process management process is to improve the software process. By improvement, we mean that the capability of the process to produce quality goods at low cost is improved. For this, the current software process is studied, frequently by studying the projects that have been done using the process.

The whole process of understanding the current process, analyzing its properties, determining how to improve, and then affecting the improvement is dealt with by the process management process. The relationship between these major component processes. These component processes are distinct not only in the type of activities performed in them, but typically also in the people who perform the activities specified by the process. In a typical project, development activities are performed by programmers, designers, testers, etc.; the project management process activities are performed by the project management; configuration control process activities are performed by a group generally called the configuration controller; and the process management process activities are performed by the software engineering process group (SEPG).


Software Process
Product Engineering Process
Process Management Process
Development Process
Project Management Process
Software Configuration Management Process

We will focus primarily on processes relating to product engineering, particularly the development and project management processes. Much of the book discusses the different phases of a development process and the sub-processes or methodologies used for executing these phases. For the rest of the book, we will use the term software process to mean product engineering processes, unless specified otherwise.

Software Development Process Models

For the software development process, the goal is to produce a high-quality software product. It therefore focuses on activities directly related to production of the software, for example, design, coding, and testing. As the development process specifies the major development and quality control activities that need to be performed in the project, it forms the core of the software process. The management process is often decided based on the development process.

A project's development process defines the tasks the project should perform, and the order in which they should be done. A process limits the degrees of freedom for a project by specifying what types of activities must be undertaken and in what order, such that the "shortest" (or the most efficient) path is obtained from the user needs to the software satisfying these needs. The process drives a project and heavily influences the outcome.

A process model specifies a general process, usually as a set of stages in which a project should be divided, the order in which the stages should be executed, and any other constraints and conditions on the execution of stages. The basic premise behind a process model is that, in the situations for which the model is applicable, using the process model as the project's process will lead to low cost, high quality, reduced cycle time, or provide other benefits. In other words, the process model provides generic guidelines for developing a suitable process for a project. Due to the importance of the development process, various models have been proposed. In this section we will discuss some of the major models.

A software process is a collection of various activities. There are five generic process framework activities :

1. **Communication:** The software development starts with the communication between customer and developer.

2. **Planning :** It consists of complete estimation, scheduling for project development and tracking.

3. **Modeling:** Modeling consists of complete requirement analysis and the design of the project like algorithm, flowchart etc.

   ➢ The algorithm is the step-by-step solution of the problem and the flow chart shows a complete flow diagram of a program.

4. **Construction:** Construction consists of code generation and the testing part.

   ➢ Coding part implements the design details using an appropriate programming language.

   ➢ Testing is to check whether the flow of coding is correct or not.

- Testing also check that the program provides desired output.

5. **Deployment:** Deployment step consists of delivering the product to the customer and take feedback from them.

   - If the customer wants some corrections or demands for the additional capabilities, then the change is required for improvement in the quality of the software.

1.4 Software Development Process Model

Q8. What is a Software Process Model?

(OR)

Describe various software.

Ans :

A Process Model describes the sequence of phases for the entire lifetime of a product. Therefore it is sometimes also called Product Life Cycle. This covers everything from the initial commercial idea until the final de-installation or disassembling of the product after its use. Usually there are three main phases :

- concept phase
- implementation phase
- maintenance phase

Each of these main phases usually has some sub-phases, like a requirements engineering phase, a design phase, a build phase and a testing phase. The sub-phases may occur in more than one main phase each of them with a specific peculiarity depending on the main phase. Besides the phases a Process Model shall also define at least:

- The activities that have to be carried out in each of the sub-phases, including the sequence in which these activities have to be carried out.
- The roles of the executors that have to carry out the activities, including a description of their responsibilities and required skills.
- The work products that have to be established or updated in each of the activities. Besides the final product there are usually several other items that have to be generated during the development of a product. These are for example requirements and design document, test specifications and test reports, etc.

Therefore, a Process Model provides a fixed framework that guides a project in :

- Development of the product
- Planning and organizing the project
- Tracking and running the project

Q9. Describe Waterfall Model.

Ans :

The simplest process model is the waterfall model, which states that the phases are organized in a linear order. The model was originally proposed by Royce, though variations of the model have evolved depending on the nature of activities and the flow of control between them. In this model, a project begins with feasibility analysis. Upon successfully demonstrating the feasibility of a project, the requirements analysis and project planning begins. The design starts after the requirements analysis is complete, and coding begins after the design is complete. Once the programming is completed, the code is integrated and testing is done. Upon successful completion of testing, the system is installed. After this, the regular operation and maintenance of the system takes place.

The basic idea behind the phases is separation of concerns each phase deals with a distinct and separate set of concerns. By doing this, the large and complex task of building the software is broken into smaller tasks (which, by themselves, are still quite complex) of specifying requirements, doing design, etc. Separating the concerns and focusing on a select few in a phase gives a better handle to the engineers and managers in dealing with the complexity of the problem.

The requirements analysis phase is mentioned as “analysis and planning.” Planning is a critical activity in software development. A good plan is based on the requirements of the system and should be done before later phases begin. However, in practice, detailed requirements are not necessary for planning. Consequently, planning usually overlaps with the requirements analysis, and a plan is ready before the later phases begin. This plan is an additional input to all the later phases.

Linear ordering of activities has some important consequences. First, to clearly identify the end of a phase and the beginning of the next, some certification mechanism has to be employed at the end of each phase. This is usually done by some verification and validation means that will ensure that the output of a phase is consistent with its input (which is the output of the previous phase), and that the output of the phase is consistent with the overall requirements of the system.

The consequence of the need for certification is that each phase must have some defined output that can be evaluated and certified. That is, when the activities of a phase are completed, there should be some product that is produced by that phase. The outputs of the earlier phases are often called work products and are usually in the form of documents like the requirements document or design document. For the coding phase, the output is the code. Though the set of documents that should be produced in a project is dependent on how the process is implemented, the following documents generally form a reasonable set that should be produced in each project:

- ➢ Requirements document
- ➢ Project plan
- ➢ Design documents (architecture, system, detailed)
- ➢ Test plan and test reports
- ➢ Final code
- ➢ Software manuals (e.g., user, installation, etc.)

One of the main advantages of the waterfall model is its simplicity. It is conceptually straight forward and divides the large task of building a software system into a series of cleanly divided phases, each phase dealing with a separate logical concern. It is also easy to administer in a contractual setup—as each phase is completed and its work product produced, some amount of money is given by the customer to the developing organization.

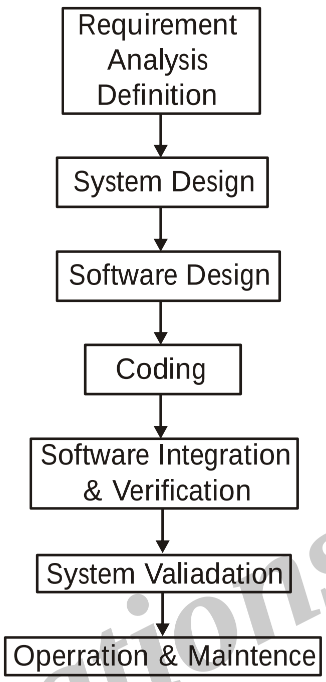


Fig.: Waterfall Model

The phases of "The Waterfall Model" are :

(i) Requirement Analysis & Definition: All requirements of the system which has to be developed are collected in this step. Like in other process models requirements are split up in functional requirements and constraints which the system has to fulfil. Requirements have to be collected by analysing the needs of the end user(s) and checking them for validity and the possibility to implement them. The aim is to generate a Requirements Specification Document which is used as an input for the next phase of the model.

(ii) System Design: The system has to be properly designed before any implementation is started. This involves an architectural design which defines and describes the main blocks and components of the system, their interfaces and interactions. By this the needed hardware is defined and the software is split up in its components. E.g. this involves the definition or selection of a computer platform, an operating system, other peripheral hardware, etc. The software components have to be defined to meet the end user requirements and to meet the need of possible scalability of the system. The aim of this phase is to generate a System Architecture Document this serves as an input for the software design phase of the development, but also as an input for hardware design or selection activities. Usually in this phase various documents are generated, one for each discipline, so that the software usually will receive a software architecture document.

(iii) **Software Design:** Based on the system architecture which defines the main software blocks the software design will break them further down into code modules. The interfaces and interactions of the modules are described, as well as their functional contents. All necessary system states like startup, shutdown, error conditions and diagnostic modes have to be considered and the activity and behaviour of the software has to be defined. The output of this phase is a Software Design Document which is the base of the following implementation work.

(iv) **Coding:** Based on the software design document the work is aiming to set up the defined modules or units and actual coding is started. The system is first developed in smaller portions called units. They are able to stand alone from an functional aspect and are integrated later on to form the complete software package.

(v) **Software Integration & Verification :** Each unit is developed independently and can be tested for its functionality. This is the so called Unit Testing. It simply verifies if the modules or units to check if they meet their specifications. This involves functional tests at the interfaces of the modules, but also more detailed tests which consider the inner structure of the software modules. During integration the units which are developed and tested for their functionalities are brought together. The modules are integrated into a complete system and tested to check if all modules cooperate as expected.

(vi) **System Validation :** After successfully integration including the related tests the complete system has to be tested against its initial requirements. This will include the original hardware and environment, whereas the previous integration and testing phase may still be performed in a different environment or on a test bench.

(vii) **Operation & Maintenance:** The system is handed over to the customer and will be used the first time by him. Naturally the customer will check if his requirements were implemented as expected but he will also validate if the correct requirements have been set up in the beginning. In case there are changes necessary it has to be fixed to make the system usable or to make it comply to the customer wishes. In most of the "Waterfall Model" descriptions this phase is extended to a never ending phase of "Operations & Maintenance". All the problems which did not arise during the previous phases will be solved in this last phase.

Weakness

The weakness of the Waterfall Model is at hand :

(i) It is very important to gather all possible requirements during the first phase of requirements collection and analysis. If not all requirements are obtained at once the subsequent phases will suffer from it. Reality is that only a part of the requirements is known at the beginning and a certain percentage will be gathered during the complete development time.

(ii) Iterations are only meant to happen within the same phase or at best from the start of the subsequent phase back to the previous phase. If the process is kept according to the school book this tends to shift the solution of problems into later phases which eventually results in a bad system design. Instead of solving the root causes the tendency is to patch problems with inadequate measures.

(iii) There may be a very big "Maintenance" phase at the end. The process only allows for a single run through the waterfall. Eventually this could be only a first sample phase which means that the further development is squeezed into the last never ending maintenance phase and virtually run without a proper process.

Q10. Explain Prototype Model and Iterative Development Model.

Ans : **(Imp.)**

I. Prototype Model

The goal of a prototyping-based development process is to counter the first limitation of the waterfall model. The basic idea here is that instead of freezing the requirements before any design or coding can proceed, a throwaway prototype is built to help understand the requirements. This prototype is developed based on the currently known requirements. Development of the prototype obviously undergoes design, coding, and testing, but each of these phases is not done very formally or thoroughly. By using this prototype, the client can get an actual feel of the system, which can enable the client to better understand the requirements of the desired system. This results in more stable requirements that change less frequently.

Prototyping is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determine the requirements. In such situations, letting the client "play" with the prototype provides invaluable and intangible inputs that help determine the requirements for the system. It is also an effective method of demonstrating the feasibility of a certain approach. This might be needed for novel systems, where it is not clear that constraints can be met or that algorithms can be developed to implement the requirements. In both situations, the risks associated with the projects are being reduced through the use of prototyping.

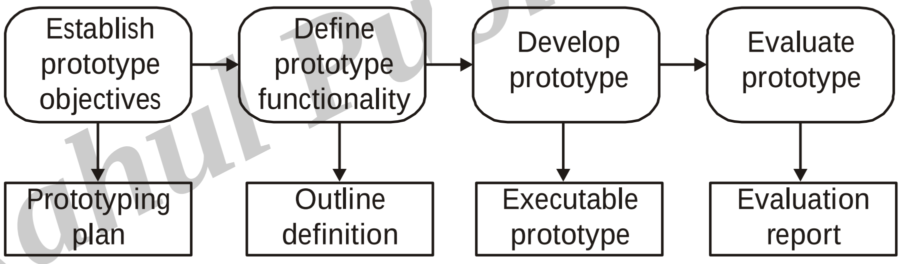


Fig.: Prototype Model

A development process using throwaway prototyping typically proceeds as follows. The development of the prototype typically starts when the preliminary version of the requirements specification document has been developed. At this stage, there is a reasonable understanding of the system and its needs and which needs are unclear or likely to change. After the prototype has been developed, the end users and clients are given an opportunity to use and explore the prototype. Based on their experience, they provide feedback to the developers regarding the prototype: what is correct, what needs to be modified, what is missing, what is not needed, etc. Based on the feedback, the prototype is modified to incorporate some of the suggested changes that can be done easily, and then the users and the clients are again allowed to use the system. This cycle repeats until, in the judgment of the prototype developers and analysts, the benefit from further changing the system and obtaining feedback is outweighed by the cost and time involved in making the changes and obtaining the feedback. Based on the feedback, the initial requirements are modified to produce the final requirements specification, which is then used to develop the production quality system.

For prototyping for the purposes of requirement analysis to be feasible, its cost must be kept low. Consequently, only those features are included in the prototype that will have a valuable return from the user experience. Exception handling, recovery, and conformance to some standards and formats are typically not included in prototypes. In prototyping, as the prototype is to be discarded, there is no point in implementing those parts of the requirements that are already well understood. Hence, the focus of the development is to include those features that are not properly understood. And the development approach is "quick and dirty" with the focus on quick development rather than quality. Because the prototype is to be thrown away, only minimal documentation needs to be produced during prototyping. For example, design documents, a test plan, and a test case specification are not needed during the development of the prototype. Another important cost-cutting measure is to reduce testing. Because testing consumes a major part of development expenditure during regular software development, this has a considerable impact in reducing costs. By using these types of cost-cutting methods, it is possible to keep the cost of the prototype to less than a few percent of the total development cost.

And the returns from this extra cost can be substantial. First, the experience of developing the prototype will reduce the cost of the actual software development. Second, as requirements will be more stable now due to the feedback from the prototype, there will be fewer changes in the requirements. Consequently the costs incurred due to changes in the requirements will be substantially reduced. Third, the quality of final software is likely to be far superior, as the experience engineers have obtained while developing the prototype will enable them to create a better design, write better code, and do better testing. And finally, developing a prototype mitigates many risks that exist in a project where requirements are not well known.

Overall, prototyping is well suited for projects where requirements are hard to determine and the confidence in the stated requirements is low. In such projects where requirements are not properly understood in the beginning, using the prototyping process model can be the most effective method for developing the software. It is also an excellent technique for reducing some types of risks associated with a project.

II. Iterative Development Model

The iterative development process model counters the third and fourth limitations of the waterfall model and tries to combine the benefits of both prototyping and the waterfall model. The basic idea is that the software should be developed in increments, each increment adding some functional capability to the system until the full system is implemented.

In the first step of this model, a simple initial implementation is done for a subset of the overall problem. This subset is one that contains some of the key aspects of the problem that are easy to understand and implement and which form a useful and usable system. A project control list is created that contains, in order, all the tasks that must be performed to obtain the final implementation. This project control list gives an idea of how far along the project is at any given step from the final system.

Each step consists of removing the next task from the list, designing the implementation for the selected task, coding and testing the implementation, performing an analysis of the partial system obtained after this step, and updating the list as a result of the analysis. These three phases are called the design phase, implementation phase, and analysis phase. The process is iterated until the project control list is empty, at which time the final implementation of the system will be available. The iterative enhancement model is shown in Figure.

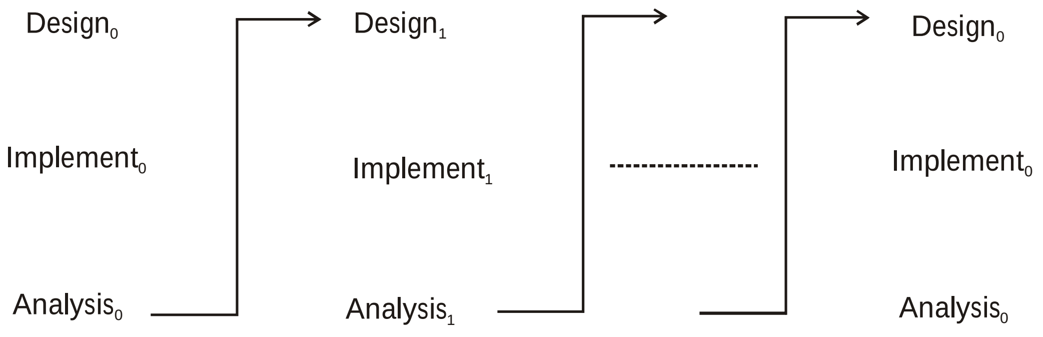
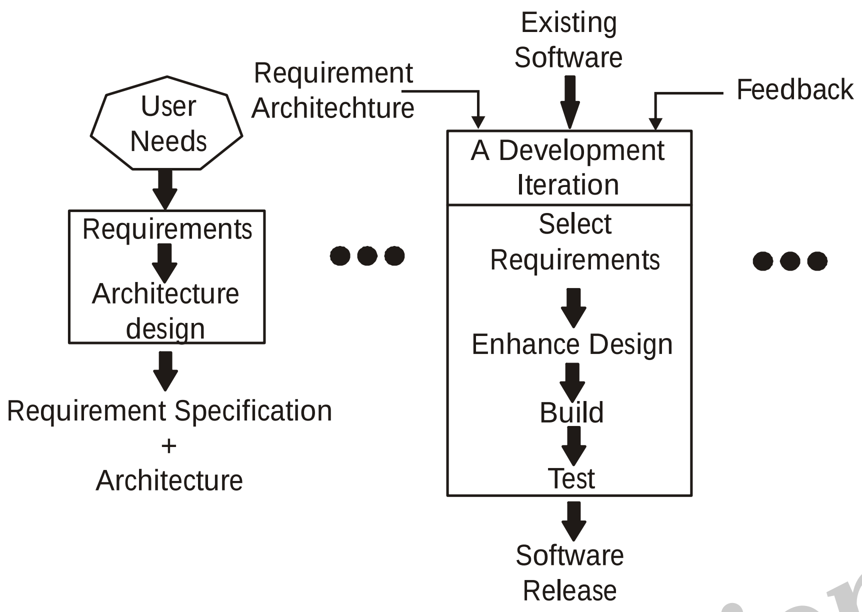


Fig.: Interative Development Model

The project control list guides the iteration steps and keeps track of all tasks that must be done. Based on the analysis, one of the tasks in the list can include redesign of defective components or redesign of the entire system. However, redesign of the system will generally occur only in the initial steps. In the later steps, the design would have stabilized and there is less chance of redesign. Each entry in the list is a task that should be performed in one step of the iterative enhancement process and should be simple enough to be completely understood. Selecting tasks in this manner will minimize the chances of error and reduce the redesign work. The design and implementation phases of each step can be performed in a top-down manner or by using some other technique.

Though there are clear benefits of iterative development, particularly in allowing changing requirements, not having the all-or-nothing risk, etc., there are some costs associated with iterative development also. For example, as the requirements for future iterations are not known, the design of a system may not be too robust. Also, changes may have to be made to the existing system to accommodate requirements of the future iterations, leading to extra rework and/or discarding of work done earlier. Overall, it may not offer the best technical solution, but the benefits may outweigh the costs in many projects.

Another common approach for iterative development is to do the requirements and the architecture design in a standard waterfall or prototyping approach, but deliver the software iteratively. That is, the building of the system, which is the most time and effort-consuming task, is done iteratively, though most of the requirements are specified upfront. We can view this approach as having one iteration delivering the requirements and the architecture plan, and then further iterations delivering the software in increments. At the start of each delivery iteration, which requirements will be implemented in this release are decided, and then the design is enhanced and code developed to implement the requirements. The iteration ends with delivery of a working software system providing some value to the end user. Selecting of requirements for an iteration is done primarily based on the value the requirement provides to the end users and how critical they are for supporting other requirements.



The advantage of this approach is that as the requirements are mostly known upfront, an overall view of the system is available and a proper architecture can be designed which can remain relatively stable. With this, hopefully rework in development iterations will diminish. At the same time, the value to the end customer is delivered iteratively so it does not have the all-or-nothing risk. Also, since the delivery is being done incrementally, and planning and execution of each iteration is done separately, feedback from an iteration can be incorporated in the next iteration. Even new requirements that may get uncovered can also be incorporated. Hence, this model of iterative development also provides some of the benefits of the model discussed above.

The iterative approach is becoming extremely popular, despite some difficulties in using it in this context. There are a few key reasons for its increasing popularity. First and foremost, in today's world clients do not want to invest too much without seeing returns. In the current business scenario, it is preferable to see returns continuously of the investment made. The iterative model permits this—after each iteration some working software is delivered, and the risk to the client is therefore limited. Second, as businesses are changing rapidly today, they never really know the "complete" requirements for the software, and there is a need to constantly add new capabilities to the software to adapt the business to changing situations. Iterative process allows this. Third, each iteration provides a working system for feedback, which helps in developing stable requirements for the next iteration. Below we will describe some other process models, all of them using some iterative approach.

Q11. Explain in detail about Rational Unified Process Model and Time Box Model.

Ans :

1. Rational Unified Proces Model

Rational Unified Process (RUP) is another iterative process model that was designed by Rational, now part of IBM. Though it is a general process model, it was designed for object-oriented development using the Unified Modeling Language (UML).

RUP proposes that development of software be divided into cycles, each cycle delivering a fully working system. Generally, each cycle is executed as a separate project whose goal is to deliver some additional capability to an existing system (built by the previous cycle). Hence, for a project, the process for a cycle forms the overall process. Each cycle itself is broken into four consecutive phases:

(i) Inception phase

(ii) Elaboration phase

(iii) Construction phase

(iv) Transition phase

(i) Inception phase

The purpose of the inception phase is to establish the goals and scope of the project, and completion of this phase is the lifecycle objectives milestone. This milestone should specify the vision and high-level capability of the eventual system, what business benefits it is expected to provide, some key illustrative use cases of the system, key risks of the project, and a basic plan of the project regarding the cost and schedule. Based on the output of this phase, a go/no-go decision may be taken. And if the project is to proceed, then this milestone represents that there is a shared vision among the stakeholders and they agree to the project, its vision, benefits, cost, usage, etc.

(ii) Elaboration phase

In the elaboration phase, the architecture of the system is designed, based on the detailed requirements analysis. The completion of this phase is the lifecycle architecture milestone. At the end of this phase, it is expected that most of the requirements have been identified and specified, and the architecture of the system has been designed (and specified) in a manner that it addresses the technical risks identified in the earlier phase.

In addition, a high-level project plan for the project has been prepared showing the remaining phases and iterations in those, and the current perception of risks. By the end of this phase, the critical engineering decisions regarding the choice of technologies, architecture, etc. have been taken, and a detailed understanding of the project exists. Outputs of this milestone allow technical evaluation of the proposed solution, as well as a better informed decision about cost-benefit analysis of the project.

(iii) Construction phase

In the construction phase, the software is built and tested. This phase results in the software product to be delivered, along with associated user and other manuals, and successfully completing this phase results in the initial operational capability milestone being achieved.

(iv) Transition phase

The purpose of the transition phase is to move the software from the development environment to the client's environment, where it is to be hosted. This is a complex task which can require additional testing, conversion of old data for this software to work, training of personnel, etc. The successful execution of this phase results in achieving the milestone product release. The different phases and milestones in RUP.

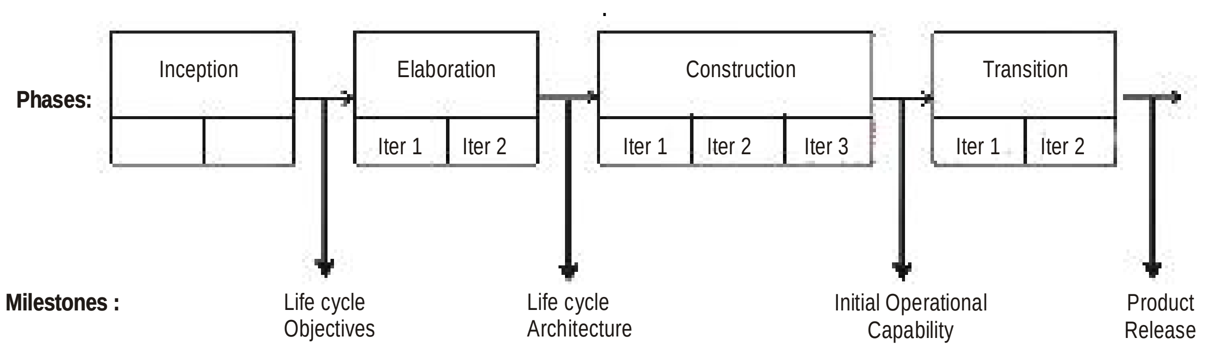


Fig.: Relational unified process model

Though these phases are consecutive, each phase itself may have multiple iterations, with each iteration delivering to an internal or external customer some well-defined output which is often a part of the final deliverable of that phase's milestone. Generally, it is expected that the construction phase will be broken into multiple iterations, each iteration producing a working system which can be used for feedback, evaluation, beta-testing, etc. Though iterations in construction are done often and it is clear what an iteration in this phase delivers, iterations may be done meaningfully in other phases as well. For example, in the elaboration phase, the first iteration may just specify the overall architecture and high-level requirements, while the second iteration may be done to thrash out the details. As another example, there may be multiple iterations to transition the developed software, with each iteration "making live" some part or some feature of the developed software.

RUP has carefully chosen the phase names so as not to confuse them with the engineering tasks that are to be done in the project, as in RUP the engineering tasks and phases are separate. Different engineering activities may be performed in a phase to achieve its milestones. RUP groups the activities into different subprocesses which it calls core process workflows. These subprocesses correspond to the tasks of performing requirements analysis, doing design, implementing the design, testing, project management, etc.

One key difference of RUP from other models is that it has separated the phases from the tasks and allows multiple of these subprocesses to function within a phase. In waterfall (or waterfall-based iterative model), a phase within a process was linked to a particular task performed by some process like requirements, design, etc. In RUP these tasks are separated from the stages, and it allows, for example, during construction, execution of the requirements process. That is, it allows some part of the requirement activity be done even in construction, something the waterfall did not allow.

So, a project, if it so wishes, may do detailed requirements only for some features during the elaboration phase, and may do detailing of other requirements while the construction is going on (maybe the first iteration of it). This not only allows a project a greater degree of flexibility in planning when the different tasks should be done, it also captures the reality of the situation it is often not possible to specify all requirements at the start and it is best to start the project with some requirements and work out the details later.

Though a subprocess may be active in many phases, as can be expected, the volume of work or the effort being spent on the subprocess will vary with phases. For example, it is expected that a lot more effort will be spent in the requirement subprocess during elaboration, and less will be spent in construction, and still less, if any, will be spent in transition. Similarly, the model has the development process active in elaboration, which allows a project to build a prototype during the elaboration phase to help its requirements activity, if needed. However, most of the implementation does happen in the construction phase. The effort spent in a subprocess in different phases will, of course, depend on the project. However, a general pattern is indicated in Table 2.1 by specifying if the level of effort for the phase is high, medium, low, etc.

Overall, RUP provides a flexible process model, which follows an iterative approach not only at a top level (through cycles), but also encourages iterative approach during each of the phases in a cycle. And in phases, it allows the different tasks to be done as per the needs of the project.

2. Time boxing Model

To speed up development, parallelism between the different iterations can be employed. That is, a new iteration commences before the system produced by the current iteration is released, and hence development of a new release happens in parallel with the development of the current release. By starting an iteration before the previous iteration has completed, it is possible to reduce the average delivery time for iterations. However, to support parallel execution, each iteration has to be structured properly and teams have to be organized suitably. The timeboxing model proposes an approach for these.

In the timeboxing model, the basic unit of development is a time box, which is of fixed duration. Since the duration is fixed, a key factor in selecting the requirements or features to be built in a time box is what can be fit into the time box. This is in contrast to regular iterative approaches where the functionality is selected and then the time to deliver is determined. Timeboxing changes the perspective of development and makes the schedule a nonnegotiable and a high-priority commitment.

Each time box is divided into a sequence of stages, like in the waterfall model. Each stage performs some clearly defined task for the iteration and produces a clearly defined output. The model also requires that the duration of each stage, that is, the time it takes to complete the task of that stage, is approximately the same. Furthermore, the model requires that there be a dedicated team for each stage. That is, the team for a stage performs only tasks of that stage—tasks for other stages are performed by their respective teams. This is quite different from other iterative models where the implicit assumption is that the same team performs all the different tasks of the project or the iteration.

Having time-boxed iterations with stages of equal duration and having dedicated teams renders itself to pipelining of different iterations. (Pipelining is a concept from hardware in which different instructions are executed in parallel, with the execution of a new instruction starting once the first stage of the previous instruction is finished.)

To illustrate the use of this model, consider a time box consisting of three stages: requirement specification, build, and deployment. The requirement stage is executed by its team of analysts and ends with a prioritized list of requirements to be built in this iteration along with a high-level design. The build team develops the code for implementing the requirements, and performs the testing. The tested code is then handed over to the deployment team, which performs predeployment tests, and then installs the system for production use. These three stages are such that they can be done in approximately equal time in an iteration.

With a time box of three stages, the project proceeds as follows. When the requirements team has finished requirements for timebox-1, the requirements are given to the build team for building the software. The requirements team then goes on and starts preparing the requirements for timebox-2. When the build for timebox-1 is completed, the code is handed over to the deployment team, and the build team moves on to build code for requirements for timebox- 2, and the requirements team moves on to doing requirements for timebox-3.

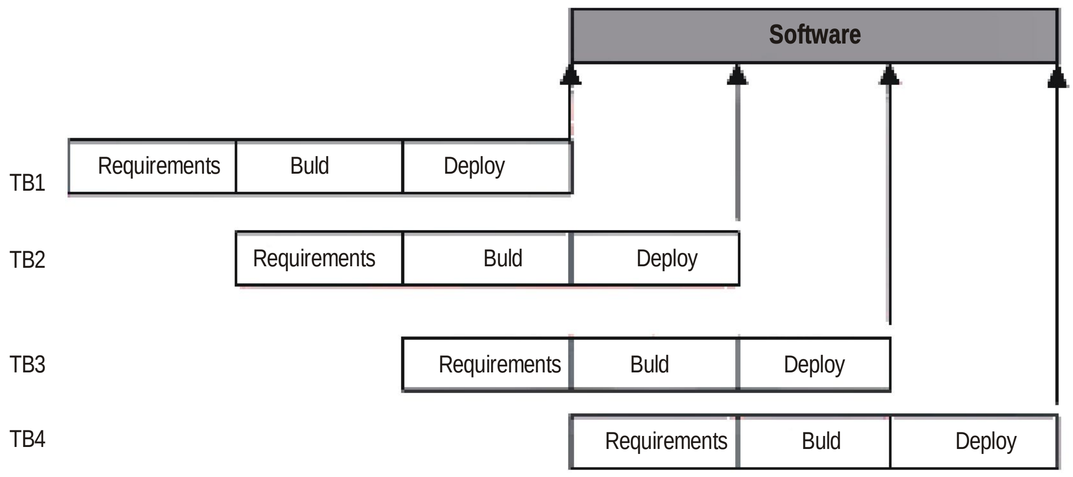


Fig.: Time boxing model

With a three-stage time box, at most three iterations can be concurrently in progress. If the time box is of size T days, then the first software delivery will occur after T days. The subsequent deliveries, however, will take place after every T/3 days. For example, if the time box duration T is 9 weeks (and each stage duration is 3 weeks), the first delivery is made 9 weeks after the start of the project. The second delivery is made after 12 weeks, the third after 15 weeks, and so on. Contrast this with a linear execution of iterations, in which the first delivery will be made after 9 weeks, the second after 18 weeks, the third after 27 weeks, and so on.

There are three teams working on the project—the requirements team, the build team, and the deployment team. It should be clear that the duration of each iteration has not been reduced.

The total work done in a time box and the effort spent in it also remains the same—the same amount of software is delivered at the end of each iteration as the time box undergoes the same stages. If the effort and time spent in each iteration also remains the same, then what is the cost of reducing the delivery time? The real cost of this reduced time is in the resources used in this model. With timeboxing, there are dedicated teams for different stages and the total team size for the project is the sum of teams of different stages. This is the main difference from the situation where there is a single team which performs all the stages and the entire team works on the same iteration.

Hence, the timeboxing provides an approach for utilizing additional manpower to reduce the delivery time. It is well known that with standard methods of executing projects, we cannot compress the cycle time of a project substantially by adding more manpower. However, through the timeboxing model, we can use more manpower in a manner such that by parallel execution of different stages we are able to deliver software quicker. In other words, it provides a way of shortening delivery times through the use of additional manpower.

Timeboxing is well suited for projects that require a large number of features to be developed in a short time around a stable architecture using stable technologies. These features should be such that there is some flexibility in grouping them for building a meaningful system in an iteration that provides value to the users. The main cost of this model is the increased complexity of project management (and managing the products being developed) as multiple developments are concurrently active. Also, the impact of unusual situations in an iteration can be quite disruptive. Further details about the model, as well as a detailed example of applying the model on a real commercial project, are given in [60, 59].

Q12. Explain the Principle of Agile Process Model.

Ans :

Agile development approaches evolved in the 1990s as a reaction to documentation and bureaucracy-based processes, particularly the waterfall approach. Agile approaches are based on some common principles, some of which are [www.extremeprogramming.org] :

- Working software is the key measure of progress in a project.
- For progress in a project, therefore, software should be developed and delivered rapidly in small increments.
- Even late changes in the requirements should be entertained (small-increment model of development helps in accommodating them).
- Face-to-face communication is preferred over documentation.
- Continuous feedback and involvement of customer is necessary for developing good-quality software.
- Simple design which evolves and improves with time is a better approach than doing an elaborate design up front for handling all possible scenarios.
- The delivery dates are decided by empowered teams of talented individuals (and are not dictated).

Agile Principles

- The highest priority of this process is to satisfy the customer.
- Acceptance of changing requirement even late in development.
- Frequently deliver a working software in small time span.
- Throughout the project business people and developers work together on daily basis.
- Projects are created around motivated people if they are given the proper environment and support.
- Face to face interaction is the most efficient method of moving information in the development team.
- Primary measure of progress is a working software.
- Agile process helps in sustainable development.
- Continuous attention to technical excellence and good design increases agility.
- From self organizing teams the best architecture, design and requirements are emerged.
- Simplicity is necessary in development.

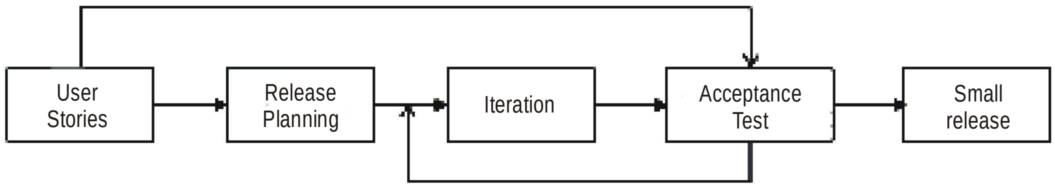
Many detailed agile methodologies have been proposed, some of which are widely used now. Extreme programming (XP) is one of the most popular and well-known approaches in the family of agile methods. Like all agile approaches, it believes that changes are inevitable and rather than treating changes as undesirable, development should embrace change. And to accommodate change, the development process has to be lightweight and quick to respond.

For this, it develops software iteratively, and avoids reliance on detailed and multiple documents which are hard to maintain. Instead it relies on face-to-face communication, simplicity, and feedback to ensure that the desired changes are quickly and correctly reflected in the programs. Here we briefly discuss the development process of XP, as a representative of an agile process.

An extreme programming project starts with user stories which are short (a few sentences) descriptions of what scenarios the customers and users would like the system to support. They are different from traditional requirements specification primarily in details—user stories do not contain detailed requirements which are to be uncovered only when the story is to be implemented, therefore allowing the details to be decided as late as possible. Each story is written on a separate card, so they can be flexibly grouped.

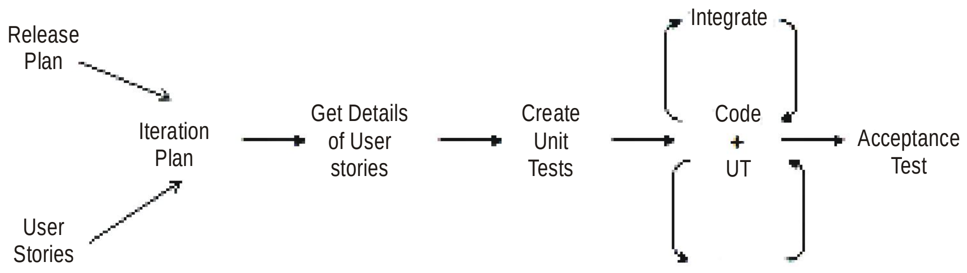
The empowered development team estimates how long it will take to implement a user story. The estimates are rough, generally stated in weeks. Using these estimates and the stories, release planning is done which defines which stories are to be built in which system release, and the dates of these releases.

Frequent and small releases are encouraged, and for a release, iterations are employed. Acceptance tests are also built from the stories, which are used to test the software before the release. Bugs found during the acceptance testing for an iteration can form work items for the next iteration.



Development is done in iterations, each iteration lasting no more than a few weeks. An iteration starts with iteration planning in which the stories to be implemented in this iteration are selected—high-value and high-risk stories are considered as higher priority and implemented in early iterations. Failed acceptance tests in previous iteration also have to be handled. Details of the stories are obtained in the iteration for doing the development.

The development approach used in an iteration has some unique practices. First, it envisages that development is done by pairs of programmers (called pair programming. Instead of individual programmers. Second, it suggests that for building a code unit, automated unit tests be written first before the actual code is written, and then the code should be written to pass the tests. This approach is referred to as test-driven development, in contrast to regular code-first development in which programmers first write code and then think of how to test it. As functionality of the unit increases, the unit tests are enhanced first, and then the code is enhanced to pass the new set of unit tests. Third, as it encourages simple solutions as well as change, it is expected that the design of the solution devised earlier may at some point become unsuitable for further development. To handle this situation, it suggests that refactoring be done to improve the design, and then use the refactored code for further development. During refactoring, no new functionality is added, only the design of the existing programs is improved. Fourth, it encourages frequent integration of different units. To avoid too many changes in the base code happening together, only one pair at a time can release their changes and integrate into the common code base.



This is a very simplified description of XP. There are many other rules in XP relating to issues like rights of programmers and customers, communication between the team members and use of metaphors, trust and visibility to all stakeholders, collective ownership of code in which any pair can change any code, team management, building quick spike solutions to resolve difficult technical and architectural issues or to explore some approach, how bugs are to be handled, how what can be done within an iteration is to be estimated from the progress made in the previous iteration, how meetings are to be conducted, how a day in the development should start, etc. The website [www.extremeprogramming.org] is a good source on these, as well as other aspects of XP.

XP, and other agile methods, are suitable for situations where the volume and pace of requirements change is high, and where requirement risks are considerable. Because of its reliance on strong communication between all the team members, it is effective when teams are collocated and of modest size, of up to about 20 members. And as it envisages strong involvement of the customer in the development, as well as in planning the delivery dates, it works well when the customer is willing to be heavily involved during the entire development, working as a team member.

Q13. Explain the process of Extreme Programming (XP).

Ans :

➢ The Extreme Programming is commonly used agile process model.

➢ It uses the concept of object-oriented programming.

➢ A developer focuses on the framework activities like planning, design, coding and testing. XP has a set of rules and practices.

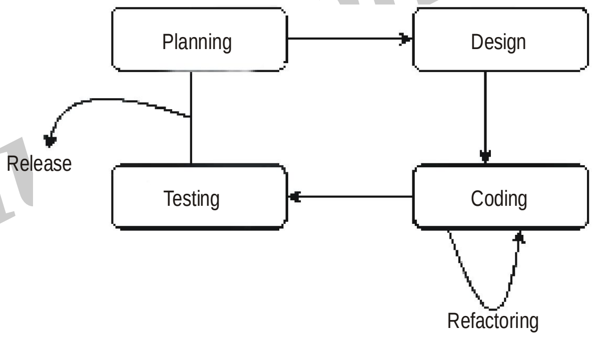


Fig. : The Extreme Programming Process

XP Values

Following are the values for extreme programming:

1. Communication

- Building software development process needs communication between the developer and the customer.
- Communication is important for requirement gathering and discussing the concept.

2. Simplicity

The simple design is easy to implement in code.

3. Feedback

Feedback guides the development process in the right direction.

4. **Courage**

In every development process there will always be a pressure situation. The courage or the discipline to deal with it surely makes the task easy.

5. **Respect**

Agile process should inculcate the habit to respect all team members, other stake holders and customer.

The XP Process

The XP process comprises four framework activities :

1. **Planning**

   - Planning starts with the requirements gathering which enables XP team to understand the rules for the software.
   - The customer and developer work together for the final requirements.

2. **Design :** The XP design follows the 'keep it simple' principle.

   - A simple design always prefers the more difficult representation.

3. **Coding**

   - The coding is started after the initial design work is over.
   - After the initial design work is done, the team creates a set of unit tests which can test each situation that should be a part of the release.
   - The developer is focused on what must be implemented to pass the test.
   - Two people are assigned to create the code. It is an important concept in coding activity.

4. **Testing**

   - Validation testing of the system occurs on a daily basis. It gives the XP team a regular indication of the progress.
   - 'XP acceptance tests' are known as the customer test.
   - Scrum
   - Scrum is an agile software development method.
   - Scrum principles are consistent with the agile platform that are used to guide development activities within a process.
   - It includes the framework activities like requirement, analysis, design, evolution and delivery.
   - Work tasks occur within a process pattern in each framework activity called as 'sprint'.
   - Scrum highlights the use of a set of software process pattern that are effective for the projects with tight timelines, changing requirements and business criticality.
   - Scrum consists of the use of a set of software process patterns.

Each process patterns defines a set of development actions which are as follows :

1. **Backlog**

   - A prioritized list of project requirements or features that provide business value for the customer.
   - Items can be added to the backlog at any time.
   - The product manager accesses the backlog and updates priorities, as required.

2. **Sprints**

   - It consists of work units that are required to achieve a requirement defined in the backlog.
   - Changes are not introduced during the sprints. It allows team members to work in short-term but in the stable environment.

3. **Scrum meeting**

   - The short meetings are held daily by the scrum team.
   - The key questions are asked and answered by all team members.

- Daily meetings guide to 'knowledge socialization' and that encourages a self-organizing team structure.

4. Demos

- Deliver the software increment to the customer. Using which the customer evaluates and demonstrates the functionalities.

1.5 Project Management Process

Q14. Explain project management process.

Ans : **(Imp.)**

We have seen many different development process models. What is the need for the different models? As mentioned earlier, while developing (industrial strength) software, the purpose is not only to develop software to satisfy the needs of some users or clients, but we want that the project be done in low cost and cycle time, and deliver high-quality software.

In addition, there could be other constraints in a project that the project may need to satisfy. Hence, given the constraints of the project, we would like to employ the process model that is likely to maximize the chances of delivering the software, and achieve the highest Q&P. Hence, selecting a suitable development process model for a project is a key decision that a project manager has to take. Let us illustrate this by a few examples.

Suppose a small team of developers has been entrusted with the task of building a small auction site for a local university. The university administration is willing to spend some time at the start to help develop the requirements, but it is expected that their availability will be limited later. The team has been given 4 months to finish the project, and an extension of the deadline seems very improbable. It also seems that the auction site will have some features that are essential, but will also have some features that are desirable but without which the system can function reasonably well.

With these constraints, it is clear that a waterfall model is not suitable for this project, as the "all or nothing" risk that it entails is unacceptable due to the inflexible deadline. The iterative enhancement model where each iteration does a complete waterfall is also not right as it requires requirements analysis for each iteration, and the users and clients are not available later. However, the iterative delivery approach in which the complete requirements are done in the first iteration but delivery is done in iterations seems well suited, with delivery being done in two (or three) iterations (as time is short). From the requirements, the project team can decide what functionality is essential to have in a working system and include it in the first iteration. The other desirable features can be planned for the second iteration. With this approach, the chances of completing the first iteration before the final deadline increase. That is, with this model, the chances of delivering a working system increase. RUP, as it allows iterations in each phase, is also a suitable model.

Consider another example where the customers are in a highly competitive environment where requirements depend on what the competition is doing, and delivering functionality regularly is highly desirable. Furthermore, to reduce cost, the customer wants to outsource as much project work as possible to another team in another country.

For this project, clearly waterfall is not suitable as requirements are not even known at the start. Iterative enhancement also may not work as it may not be able to deliver rapidly. XP will be hard to apply as it requires that the entire team, including the customer, be collocated. For this project, the time boxing model seems to fit the best. The whole project can employ three teams—one of analysts who will work with the customer to determine the requirements, one to do the development (which could be in some low-cost destination), and the third to do the deployment, which will be where the site is hosted. By suitably staffing the teams, the duration of each of the three phases—analysis and design, build, and deployment—can be made approximately equal. Then the time boxing model can be applied.

Consider another project, where a university wants to automate the registration process. It already has a database of courses and pre-requisites, and a database of student records. In this project, as the requirements are well understood (since registrations have been happening manually), the waterfall model seems to be the optimum.

Project Management Process

While the selection of the development process decides the phases and tasks to be done, it does not specify things like how long each phase should last, or how many resources should be assigned to a phase, or how a phase should be monitored. And quality and productivity in the project will also depend critically on these decisions. To meet the cost, quality, and schedule objectives, resources have to be properly allocated to each activity for the project, and progress of different activities has to be monitored and corrective actions taken when needed. All these activities are part of the project management process. Hence, a project management process is necessary to ensure that the engineering process ends up meeting the real-world objectives of cost, schedule, and quality.

The project management process specifies all activities that need to be done by the project management to ensure that cost and quality objectives are met. Its basic task is to ensure that, once a development process is chosen, it is implemented optimally. That is, the basic task is to plan the detailed implementation of the process for the particular project and then ensure that the plan is properly executed. For a large project, a proper management process is essential for success.

The activities in the management process for a project can be grouped broadly into three phases: planning, monitoring and control, and termination analysis. Project management begins with planning, which is perhaps the most critical project management activity. The goal of this phase is to develop a plan for software development following which the objectives of the project can be met successfully and efficiently. A software plan is usually produced before the development activity begins and is updated as development proceeds and data about progress of the project becomes available. During planning, the major activities are cost estimation, schedule and milestone determination, project staffing, quality control plans, and controlling and monitoring plans. Project planning is undoubtedly the single most important management activity, and it forms the basis for monitoring and control. We will devote one full chapter later in the book to project planning.

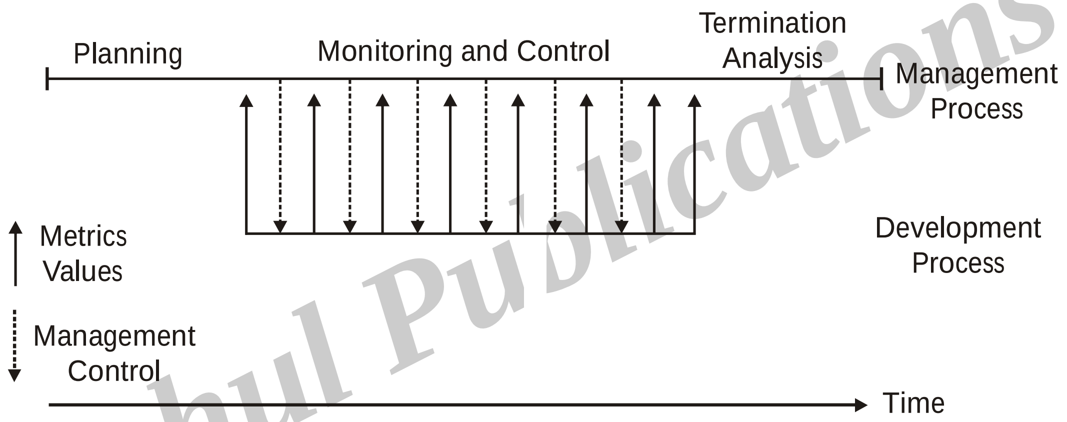
Project monitoring and control phase of the management process is the longest in terms of duration; it encompasses most of the development process. It includes all activities the project management has to perform while the development is going on to ensure that project objectives are met and the development proceeds according to the developed plan (and update the plan, if needed). As cost, schedule, and quality are the major driving forces, most of the activity of this phase revolves around monitoring factors that affect these. Monitoring potential risks for the project, which might prevent the project from meeting its objectives, is another important activity during this phase. And if the information obtained by monitoring suggests that objectives may not be met, necessary actions are taken in this phase by exerting suitable control on the development activities.

Monitoring a development process requires proper information about the project. Such information is typically obtained by the management process from the development process. Consequently, the implementation of a development process model should ensure that each step in the development process produces information that the management process needs for that step. That is, the development process provides the information the management process needs. However, interpretation of the information is part of monitoring and control.

Whereas monitoring and control last the entire duration of the project, the last phase of the management process—termination analysis—is performed when the development process is over. The basic reason for performing termination analysis is to provide information about the development process and learn from the project in order to improve the process. This phase is also often called postmortem analysis. In iterative development, this analysis can be done after each iteration to provide feedback to improve the execution of further iterations. We will not discuss it further in the book.

The temporal relationship between the management process and the development process is shown. This is an idealized relationship showing that planning is done before development begins, and termination analysis is done after development is over. As the figure shows, during the development, from the various phases of the development process, quantitative information flows to the monitoring and control phase of the management process, which uses the information to exert control on the development process.

As a plan also includes planning for monitoring, we will not discuss the monitoring separately but discuss it as part of the planning activity.



Q15. What are the differences between computer science and software engineering?

Ans : **(Imp.)**

S.No.	Computer Science	Software Engineering
1.	Computer science takes abroad approach to study of the principles and use of computers that covers both theory and applications. This filed involves the understanding and application of both abstract and concrete knowledge.	Software engineering is a filed largely concerned with the application of engineering process to the creation, maintence, and design of software for a variety of different purposes.
2.	The computer science program is generaly contained in he engineering department in four-year universities.	The software engineering program is most often housed in the engineering department in four year universities.
3.	Understanding the interaction between hardware and software will be included in curricula; however, specific training on hardware development generally will not.	In general, students in this field will not receive training on hardware development; however, they will gain knowledge on the interplay between hardware and software.
4.	Algorithmic training should be part of a computer science curriculum, pursuant to the 2013 Computer Science Curricula offered by the Joint Task Force on Computing Curricula Association for Computer Machinery (ACM), and the IEEE Computer Society.	Student will likely take class on algorithms; however, it will not be a focal point of the degree.
5.	Project management is often included in the computer science curriculum, sometimes as part of a software engineering course.	Students studying software engineering will likely take courses on project management, both in undergraduate and graduate programs.

6.	Depending on the institution, a wide array of specializations may be available, including a focus on artificial intelligence, mobile and internet computing, security, real-world computing, and theory.	While pursuing a degree in software engineering, students may have the opportunity to focus on a number of different specializations, including network-centric systems, modeling and simulation, games and entertainment systems, digital and embedded systems, and other areas.
7.	A degree in computer science may prepare students for careers as computer and information research scientists; computer network architects; computer programmers; systems analysts; information security analysts; software developers; or web developers, all of which are described in detail by the Bureau of Labor Statistics.	According to the Bureau of Labor Statistics, established positions related to software engineering may include careers in software development; computer network architecture; computer systems analysis; web development; and information research.
8.	Emerging positions in the field of computer science may include careers in cloud computing; robotics and artificial intelligence; application development; and forensic analysis. By nature, this list will continue to grow in the future.	Emerging occupations related to software engineering depend on the state of software and technology in the future. That being said, those with a degree in software engineering may find work in artificial intelligence, app development, and software development for any future technologies that arise.
9.	The following is a list of 8 schools that offer online degree programs in computer science: • Oregon State University • University of Illinois • University of Maryland University College • Florida State University • Champlain College • Penn State World Campus • Drexel University • California State University Online	The following is a list of 8 schools that offer online degree programs in software engineering: • Penn State World Campus • Brigham Young University - Idaho • Colorado Technical University • Arizona State University • Herzing University • Walden University • North Dakota State University • Carnegie Mellon University
10.	Ultimately, while it is impossible to place a single label on this field of study, largely due to the ever-changing nature of technology, those studying computer science should expect to become familiar with computers, their functions, and their general application. While some software programming may play a part in this area, computer scientists may also be tasked with developing new coding languages,or researching new technology to extend the boundaries of the industry. Overall, computer science mixes both concrete and abstract concepts as they relate to computers and technology.	Software engineering, much like other engineering disciplines, aims to utilize general applications of computers and software to create efficiencies or solve problems. As such, software engineers can expect to create and maintain existing software for a number of different purposes. While software engineers will generally not find themselves in research-based positions, they should expect to become extremely familiar with the development of software, as well as how to use the variety of tools in their arsenal to create useful finished product.

UNIT II

Software Requirements Analysis and Specification: Value of a good SRS, Requirements Process, Requirements Specification, Functional Specification with Use Cases, Other approaches for analysis.

Software Architecture: Role of Software Architecture Views, Component and connector view, Architectural styles for C & C view, Documenting Architecture Design, Evaluating Architectures.

2.1 Software Requirements Analysis and Specification

Q1. What are Software Requirements Specification and Describe various characteristics of software requirements specification?

Ans : **(Imp.)**

Meaning

The output of the requirements phase of the software development process is Software Requirements Specification (SRS) (also known as requirements document). This document lays a foundation for software engineering activities and is created when entire requirements are elicited and analyzed. SRS is a formal document, which acts as a representation of software that enables the users to review whether it (SRS) is according to their requirements. In addition, it includes user requirements for a system as well as detailed specifications of the system requirements.

IEEE defines software requirements specification as, 'a document that clearly and precisely describes each of the essential requirements (functions, performance, design constraints and quality attributes) of the software and the external interfaces. Each requirement is defined in such a way that its achievement can be objectively verified by a prescribed method, for example, inspection, demonstration, analysis or test.' Note that requirements specification can be in the form of a written document, a mathematical model, a collection of graphical models, a prototype, and so on.

Essentially, what passes from requirements analysis activity to the specification activity is the knowledge acquired about the system. The need for maintaining a requirements document is that the modeling activity essentially focuses on the problem structure and not its structural behavior. While in SRS, performance constraints, design constraints, and standard compliance recovery are clearly specified. This information helps in developing a proper design of the system.

Characteristics

1. **Feedback:** Provides a feedback, which ensures to the user that the organization (which develops the software) understands the issues or problems to be solved and the software behavior necessary to address those problems.
2. **Decompose problem into components :** Organizes the information and divides the problem into its component parts in an orderly manner.
3. **Validation:** Uses validation strategies applied to the requirements to acknowledge that requirements are stated properly.
4. **Input to design:** Contains sufficient detail in the functional system requirements to devise a design solution.
5. **Basis for agreement between the** user and **the organization :** Provides a complete description of the functions to be performed by the system. In addition, it helps the users to determine whether the specified requirements are accomplished.
6. **Reduce the development effort :** Enables developers to consider user requirements

before the designing of the system commences. As a result, 'rework' and inconsistencies in the later stages can be reduced.

7. **Estimating costs and schedules :** Determines the requirements of the system and thus enables the developer to have a 'rough' estimate of the total cost and schedule of the project.

Q2. What is requirements analysis in Software Engineering?

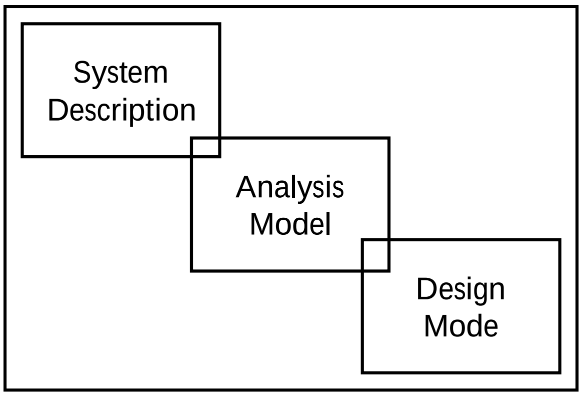
Ans :

IEEE defines requirements analysis as (1) the process of studying user needs to arrive at a definition of a system, hardware or software requirements. (2) The process of studying and refining system, hardware or software requirements.' Requirements analysis helps to understand, interpret, classify, and organize the software requirements in order to assess the feasibility, completeness, and consistency of the requirements. Various other tasks performed using requirements analysis are listed below.

1. To detect and resolve conflicts that arise due to unclear and unspecified requirements.
2. To determine operational characteristics of the software and how they interact with the environment.
3. To understand the problem for which the software is to be developed.
4. To develop an analysis model to analyze the requirements in the software.
5. Software engineers perform analysis modeling and create an analysis model to provide information of 'what' software should do instead of 'how' to fulfill the requirements in software. This model emphasizes information such as the functions that software should perform, behavior it should exhibit, and constraints that are applied on the software. This model also determines the relationship of one component with other components. The clear and complete requirements specified in the analysis model help the software development team to develop the software according to those requirements. An analysis model is created to help the development team to assess the quality of the software when it is developed. An analysis model helps to define a set of requirements that can be validated when the software is developed.

Let us consider an example of constructing a study room, where the user knows the dimensions of the room, the location of doors and windows, and the available wall space. Before constructing the study room, he provides information about flooring, wallpaper, and so on to the constructor. This information helps the constructor to analyze the requirements and prepare an analysis model that describes the requirements. This model also describes what needs to be done to accomplish those requirements. Similarly, an analysis model created for the software facilitates the software development team to understand what is required in the software and then they develop it.

In Figure the analysis model connects the system description and design model. System description provides information about the entire functionality of the system, which is achieved by implementing the software, hardware and data. In addition, the analysis model specifies the software design in the form of a design model, which provides information about the software's architecture, user interface, and component level structure.



The guidelines followed while creating an analysis model are listed below.

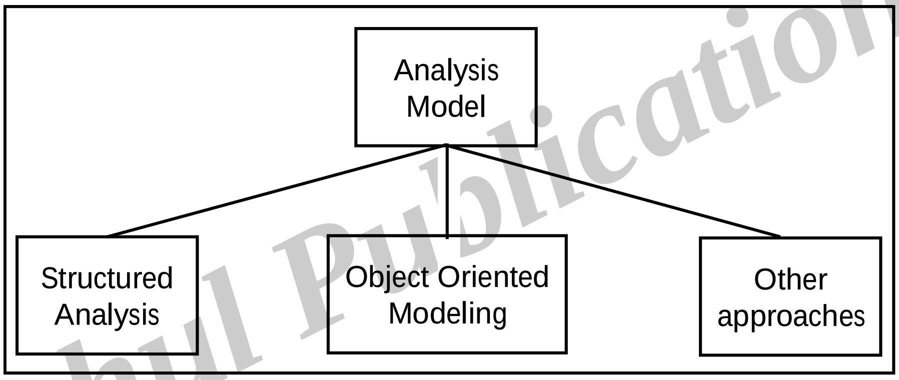
1. The model should concentrate on requirements in the problem domain that are to be accomplished. However, it should not describe the procedure to accomplish the requirements in the system.

2. Every element of the analysis model should help in understanding the software requirements. This model should also describe the information domain, function, and behavior of the system.

3. The analysis model should be useful to all stakeholders because every stakeholder uses this model in his own manner. For example, business stakeholders use this model to validate requirements whereas software designers view this model as a basis for design.

4. The analysis model should be as simple as possible. For this, additional diagrams that depict no new or unnecessary information should be avoided.

Also, abbreviations and acronyms should be used instead of complete notations. The choice of representation is made according to the requirements to avoid inconsistencies and ambiguities. Due to this, the analysis model comprises structured analysis, object-oriented modeling, and other approaches. Each of these describes a different manner to represent the functional and behavioral information. Structured analysis expresses this information through data-flow diagrams whereas object-oriented modeling specifies the functional and behavioral information using objects. Other approaches include ER modeling and several requirements specification languages and processors.



2.2 Value of a Good SRS

Q3. Explain how good is your SRS.

Ans :

The origin of most software systems is in the needs of some clients. The software system itself is created by some developers. Finally, the completed system will be used by the end users. Thus, there are three major parties interested in a new system: the client, the developer, and the users. Somehow the requirements for the system that will satisfy the needs of the clients and the concerns of the users have to be communicated to the developer. The problem is that the client usually does not understand software or the software development process, and the developer often does not understand the client's problem and application area. This causes a communication gap between the parties involved in the development project. A basic purpose of the SRS is to bridge this communication gap so they have a shared vision of the software being built. Hence, one of the main advantages of a good SRS is:

➤ **An SRS establishes the basis for agreement between the client and the supplier on what the software product will do.**

This basis for agreement is frequently formalized into a legal contract between the client (or the customer) and the developer (the supplier). So, through SRS, the client clearly describes what it expects from the supplier, and the developer clearly understands what capabilities to build in the software. A related, but important, advantage is :

➢ **An SRS provides a reference for validation of the final product.**

That is, the SRS helps the client determine if the software meets the requirements. Without a proper SRS, there is no way a client can determine if the software being delivered is what was ordered, and there is no way the developer can convince the client that all the requirements have been fulfilled. Providing the basis of agreement and validation should be strong enough reasons for both the client and the developer to do a thorough and rigorous job of requirement understanding and specification, but there are other very practical and pressing reasons for having a good SRS.

Studies have shown that many errors are made during the requirements phase. And an error in the SRS will manifest itself as an error in the final system implementing the SRS. Clearly, if we want a high-quality end product that has few errors, we must begin with a high-quality SRS. In other words, we can conclude that:

➢ **A high-quality SRS is a prerequisite to high-quality software.**

Finally, the quality of SRS has an impact on cost (and schedule) of the project. We know that errors can exist in the SRS. It is also known that the cost of fixing an error increases almost exponentially as time progresses [10, 12]. Hence, by improving the quality of requirements, we can have a huge savings in the future by having fewer expensive defect removals. In other words.

➢ **A high-quality SRS reduces the development cost**

Characteristics of a Good Software Requirements Specification

A software requirements specification should be clear, concise, consistent and unambiguous. It must correctly specify all of the software requirements, but no more. However the software requirement specification should not describe any of the design or verification aspects, except where constrained by any of the stakeholders requirements.

➢ **Complete :** For a software requirements specification to be complete, it must have the following properties: Description of all major requirements relating to functionality, performance, design constraints and external interfaces. Definition of the response of the software system to all reasonable situations. Conformity to any software standards, detailing any sections which are not appropriate Have full labelling and references of all tables and references, definitions of all terms and units of measure. Be fully defined, if there are sections in the software requirements specification still to be defined, the software requirements specification is not complete

➢ **Consistent :** A software requirement specification is consistent if none of the requirements conflict. There are a number of different types of confliction: Multiple descriptors - This is where two or more words are used to reference the same item, i.e. where the term cue and prompt are used interchangeably. Opposing physical requirements - This is where the description of real world objects clash, e.g. one requirement states that the warning indicator is orange, and another states that the indicator is red. Opposing functional requirements - This is where functional characteristics conflict, e.g. perform function X after both A and B has occurred, or perform function X after A or B has occurred.

➢ **Traceable:** A software requirement specification is traceable if both the origins and the references of the requirements are available. Traceability of the origin or a requirement can help understand who asked for the requirement and also what modifications have been made to the requirement to bring the requirement to its current state. Traceability of references are used to aid the modification of future documents by stating where a requirement has been referenced. By having foreword traceability, consistency can be more easily contained.

- **Unambiguous:** As the Oxford English dictionary states the word unambiguous means [Hawkins '88]: "not having two or more possible meanings". This means that each requirement can have one and only one interpretation. If it is unavoidable to use an ambiguous term in the requirements specification, then there should be clarification text describing the context of the term. One way of removing ambiguity is to use a formal requirements specification language. The advantage to using a formal language is the relative ease of detecting errors by using lexical syntactic analysers to detect ambiguity. The disadvantage of using a formal requirements specification language is the learning time and loss of understanding of the system by the client.

- **Verifiable:** A software requirement specification is verifiable if all of the requirements contained within the specification are verifiable. A requirement is verifiable if there exists a finite cost-effective method by which a person or machine can check that the software product meets the requirement. Non-verifiable requirements include "The system should have a good user interface" or "the software must work well under most conditions" because the performance words of good, well and most are subjective and open to interpretation. If a method cannot be devised to determine whether the software meets a requirement, then the requirement should be removed or revised.

2.3 Requirements Process

Q4. Explain about Requirement Process.

Ans : **(Imp.)**

The requirement process is the sequence of activities that need to be performed in the requirements phase and that culminate in producing a high-quality document containing the SRS. The requirements process typically consists of three basic tasks:

i) Problem or requirement analysis

ii) Requirements specification, and

iii) Requirements validation

i) Problem or requirement analysis

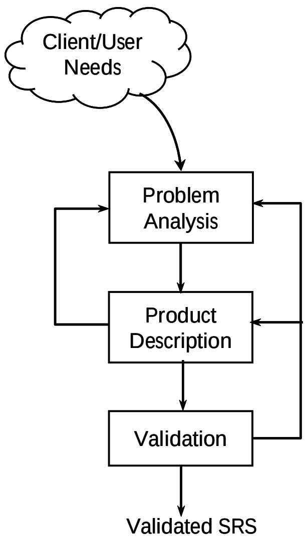
Problem analysis often starts with a high-level "problem statement." During analysis the problem domain and the environment are modeled in an effort to understand the system behavior, constraints on the system, its inputs and outputs, etc. The basic purpose of this activity is to obtain a thorough understanding of what the software needs to provide. Frequently, during analysis, the analyst will have a series of meetings with the clients and end users. In the early meetings, the clients and end users will explain to the analyst about their work, their environment, and their needs as they perceive them. Any documents describing the work or the organization may be given, along with outputs of the existing methods of performing the tasks. In these early meetings, the analyst is basically the listener, absorbing the information provided. Once the analyst understands the system to some extent, he uses the next few meetings to seek clarifications of the parts he does not understand. He may document the information or build some models, and he may do some brainstorming or thinking about what the system should do. In the final few meetings, the analyst essentially explains to the client what he understands the system should do and uses the meetings as a means of verifying if what he proposes the system should do is indeed consistent with the objectives of the clients.

ii) Requirements specification

The understanding obtained by problem analysis forms the basis of requirements specification, in which the focus is on clearly specifying the requirements in a document. Issues such as representation, specification languages, and tools are addressed during this activity. As analysis produces large amounts of information and knowledge with possible redundancies, properly organizing and describing the requirements is an important goal of this activity.

iii) Requirements validation

Requirements validation focuses on ensuring that what have been specified in the SRS are indeed all the requirements of the software and making sure that the SRS is of good quality. The requirements process terminates with the production of the validated SRS. We will discuss this more later in the chapter.



It should be pointed out that the requirements process is not a linear sequence of these three activities and there is considerable overlap and feedback between these activities. The specification activity we may go back to the analysis activity. This happens as frequently some parts of the problem are analyzed and then specified before other parts are analyzed and specified. Furthermore, the process of specification frequently shows shortcomings in the knowledge of the problem, thereby necessitating further analysis. Once the specification is done, it goes through the validation activity. This activity may reveal problems in the specification itself, which requires going back to the specification step, or may reveal shortcomings in the understanding of the problem, which requires going back to the analysis activity.

Q5. What is Software Requirement? State its guidlines.

Ans :

In the software development process, requirement phase is the first software engineering activity. This phase is a user-dominated phase and translates the ideas or views into a requirements document. Note that defining and documenting the user requirements in a concise and unambiguous manner is the first major step to achieve a high-quality product.

The requirement phase encompasses a set of tasks, which help to specify the impact of the software on the organization, customers' needs, and how users will interact with the developed software. The requirements are the basis of the system design. If requirements are not correct the end product will also contain errors. Note that requirements activity like all other software engineering activities should be adapted to the needs of the process, the project, the product and the people involved in the activity. Also, the requirements should be specified at different levels of detail. This is because requirements are meant for people such as users, business managers, system engineers, and so on. For example, business managers are interested in knowing which features can be implemented within the allocated budget whereas end-users are interested in knowing how easy it is to use the features of software.

Requirement is a condition or capability possessed by the software or system component in order to solve a real world problem. The problems can be to automate a part of a system, to correct shortcomings of an existing system, to control a device, and so on. IEEE defines requirement as

1. A condition or capability needed by a user to solve a problem or achieve an objective.
2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.
3. A documented representation of a condition or capability as in (1) or (2).'

Requirements describe how a system should act, appear or perform. For this, when users request for software, they provide an approximation of what the new system should be capable of doing. Requirements differ from one user to another and from one business process to another.

Guidelines

The purpose of the requirements document is to provide a basis for the mutual understanding between the users and the designers of the initial definition of the software development life cycle (SDLC) including the requirements, operating environment and development plan.

The requirements document should include the overview, the proposed methods and procedures, a summary of improvements, a summary of impacts, security, privacy, internal control considerations, cost considerations, and alternatives. The requirements section should state the functions required in the software in quantitative and qualitative terms and how these functions will satisfy the performance objectives.

The requirements document should also specify the performance requirements such as accuracy, validation, timing, and flexibility. Inputs, outputs, and data characteristics need to be explained. Finally, the requirements document needs to describe the operating environment and provide (or make reference to) a development plan.

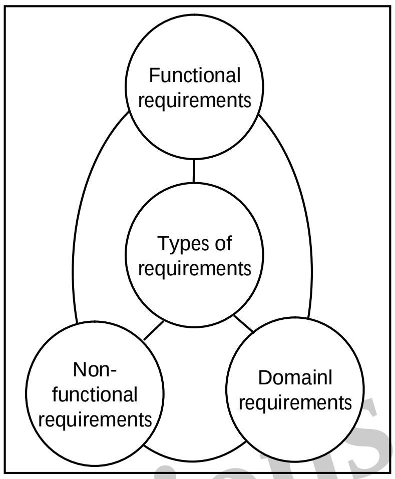
There is no standard method to express and document requirements. Requirements can be stated efficiently by the experience of knowledgeable individuals, observing past requirements, and by following guidelines. Guidelines act as an efficient method of expressing requirements, which also provide a basis for software development, system testing, and user satisfaction. The guidelines that are commonly followed to document requirements are listed below.

1. Sentences and paragraphs should be short and written in active voice. Also, proper grammar, spelling, and punctuation should be used.
2. Conjunctions such as 'and' and 'or' should be avoided as they indicate the combination of several requirements in one requirement.
3. Each requirement should be stated only once so that it does not create redundancy in the requirements specification document.

Types

Requirements help to understand the behavior of a system, which is described by various tasks of the system. For example, some of the tasks of a system are to provide a response to input values, determine the state of data objects, and so on. Note that requirements are considered prior to the development of the software. The requirements, which are commonly considered, are classified into three categories, namely.

i) Functional requirements
ii) Non-functional requirements
iii) Domain requirements.



i) Functional requirements

IEEE defines functional requirements as 'a function that a system or component must be able to perform.' These requirements describe the interaction of software with its environment and specify the inputs, outputs, external interfaces, and the functions that should be included in the software. Also, the services provided by functional requirements specify the procedure by which the software should react to particular inputs or behave in particular situations.

To understand functional requirements properly, let us consider the following example of an online banking system.

1. The user of the bank should be able to search the desired services from the available ones.
2. There should be appropriate documents' for users to read. This implies that when a user wants to open an account in the bank, the forms must be available so that the user can open an account.
3. After registration, the user should be provided with a unique acknowledgment number so that he can later be given an account number.

The above mentioned functional requirements describe the specific services provided by the online banking system. These requirements indicate user requirements and specify that functional requirements may be described at different levels of detail in an online banking system. With the help of these functional requirements, users can easily view, search and download registration forms and other information about the bank. On the other hand, if requirements are not stated properly, they are misinterpreted by software engineers and user requirements are not met.

The functional requirements should be complete and consistent. Completeness implies that all the user requirements are defined. Consistency implies that all requirements are specified clearly without any contradictory definition. Generally, it is observed that completeness and consistency cannot be achieved in large software or in a complex system due to the problems that arise while defining the functional requirements of these systems. The different needs of stakeholders also prevent the achievement of completeness and consistency. Due to these reasons, requirements may not be obvious when they are,'first specified and may further lead to inconsistencies in the requirements specification.

ii) Non-functional requirements

The non-functional requirements (also known as **quality requirements)** are related to system attributes such as reliability and response time. Non-functional requirements arise due to user requirements, budget constraints, organizational policies, and so on. These requirements are not related directly to any particular function provided by the system.

Non-functional requirements should be accomplished in software to make it perform efficiently. For example, if an Airplane is unable to fulfill reliability requirements, it is not approved for safe operation. Similarly, if a real time control system is ineffective in accomplishing non-functional requirements, the control functions cannot operate correctly.

The description of different types of non-functional requirements is listed below.

1. **Product requirements :** These requirements specify how software product performs. Product requirements comprise the following.

2. **Efficiency requirements:** Describe the extent to which the software makes optimal use of resources, the speed with which the system executes, and the memory it consumes for its operation. For example, the system should be able to operate at least three times faster than the existing system.

3. **Reliability requirements:** Describe the acceptable failure rate of the software. For example, the software should be able to operate even if a hazard occurs.

4. **Portability requirements:** Describe the ease with which the software can be transferred from one platform to another. For example, it should be easy to port the software to a different operating system without the need to redesign the entire software.

5. **Usability requirements:** Describe the ease with which users are able to operate the software. For example, the software should be able to provide access to functionality with fewer keystrokes and mouse clicks.

6. **Organizational requirements:** These requirements are derived from the policies and procedures of an organization. Organizational requirements comprise the following.

7. **Delivery requirements:** Specify when the software and its documentation are to be delivered to the user.

8. **Implementation requirements:** Describe requirements such as programming language and design method.

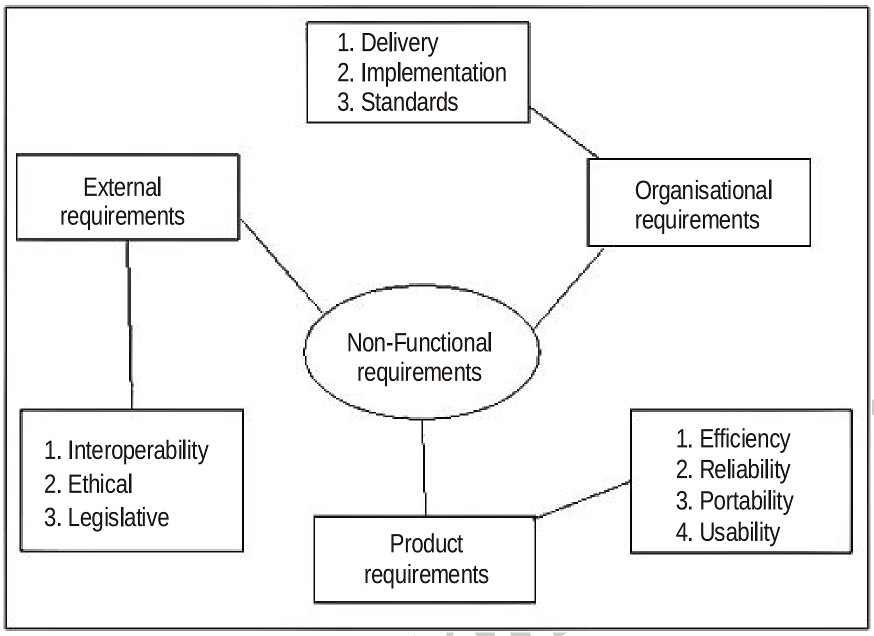
9. **Standards requirements:** Describe the process standards to be used during software development. For example, the software should be developed using standards specified by the ISO and IEEE standards.

10. **External requirements :** These requirements include all the requirements that affect the software or its development process externally. External requirements comprise the following.

11. **Interoperability requirements:** Define the way in which different computer based systems will interact with each other in one or more organizations.

12. Ethical requirements: Specify the rules and regulations of the software so that they are acceptable to users.

13. Legislative requirements: Ensure that the software operates within the legal jurisdiction. For example, pirated software should not be sold.



Non-functional requirements are difficult to verify. Hence, it is essential to write non-functional requirements quantitatively, so that they can be tested. For this, non-functional requirements metrics are used. These metrics are listed in Table.

Features	Measures
Speed	Processed transaction/ second User/event response time Screen refresh rate
Size	Amount of memory (KB) Number of RAM chips.
Ease of use	Training time Number of help windows
Reliability	Mean time to failure (MTTF) Portability of unavailability Rate of failure occurrence
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target-dependent statements Number of target systems

iii) Domain requirements

Requirements which are derived from the application domain of the system instead from the needs of the users are known as domain requirements. These requirements may be new functional requirements or specify a method to perform some particular computations. In addition, these requirements include any constraint that may be present in the existing functional requirements. As domain requirements reflect the fundamentals of the application domain, it is important to understand these requirements. Also, if these requirements are not fulfilled, it may be difficult to make .the system work as desired.

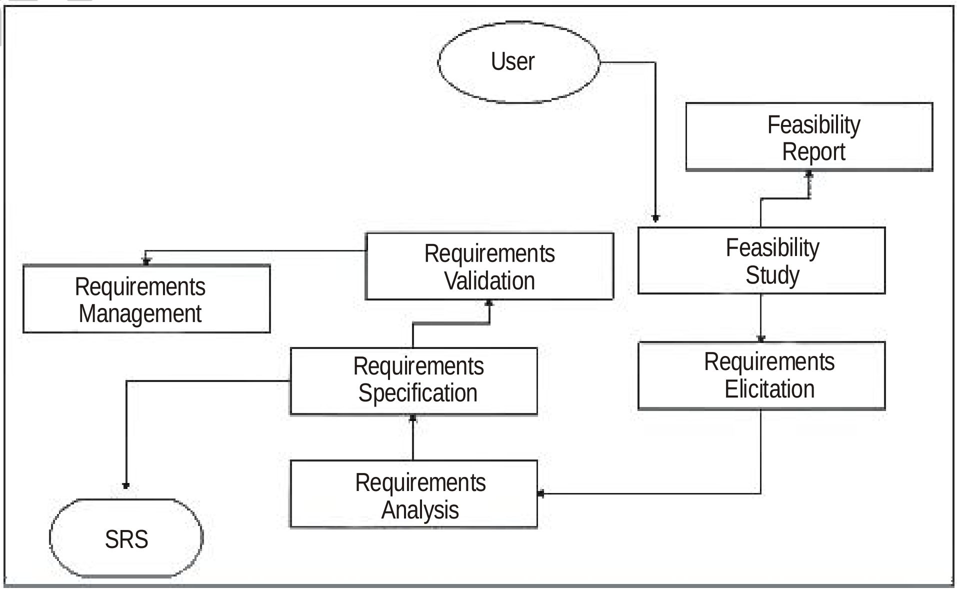
A system can include a number of domain requirements. For example, it may comprise a design constraint that describes the user interface, which is capable of accessing all the databases used in a system. It is important for a development team to create databases and interface designs as per established standards. Similarly, the requirements of the user such as copyright restrictions and security mechanism for the files and documents used in the system are also domain requirements. When domain requirements are not expressed clearly, it can result in the following difficulties.

- **Problem of understandability:** When domain requirements are specified in the language of application domain (such as mathematical expressions), it becomes difficult for software engineers to understand them.
- **Problem of implicitness:** When domain experts understand the domain requirements but do not express these requirements clearly, it may create a problem (due to incomplete information) for the development team to understand and implement the requirements in the system.

Requirements Engineering Process

This process is a series of activities that are performed in the requirements phase to express requirements in the Software Requirements Specification (SRS)document. It focuses on understanding the requirements and its type so that an appropriate technique is determined to carry out the **Requirements Engineering** (RE) **process.** The new software developed after collecting requirements either replaces the existing software or enhances its features and functionality. For example, the payment mode of the existing software can be changed from payment through hand-written cheques to electronic payment of bills.

An RE process is shown, which comprises various steps including feasibility study, requirements elicitation, requirements analysis, requirements specification, requirements validation, and requirements management.



The requirements engineering process begins with feasibility study of the requirements. Then requirements elicitation is performed, which focuses on gathering user requirements. After the requirements are gathered, an analysis is performed, which further leads to requirements specification. The output of this is stored in the form of software requirements specification document. Next, the requirements are checked for their completeness and correctness in requirements validation. Last of all, to understand and control changes to system requirements, requirements management is performed.

2.4 Requirements Specification

Q6. What is Software Requirements Specification? Explain Structure and Characteristics of SRS.

Ans : **(Imp.)**

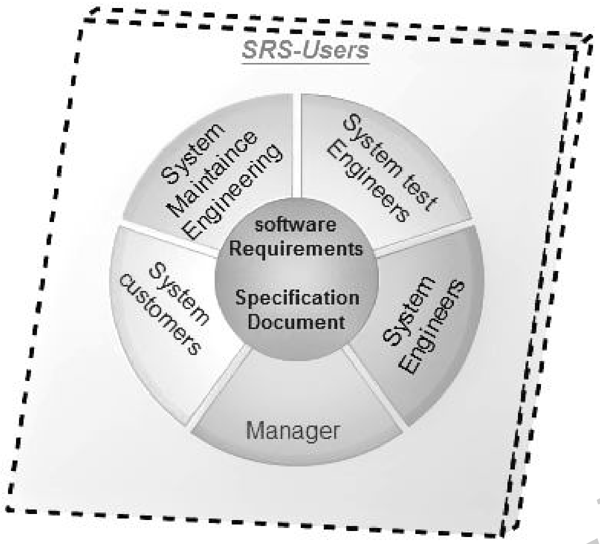
The output of the requirements phase of the software development process is Software Requirements Specification (SRS) (also known as requirements document). This document lays a foundation for software engineering activities and is created when entire requirements are elicited and analyzed. SRS is a formal document, which acts as a representation of software that enables the users to review whether it (SRS) is according to their requirements. In addition, it includes user requirements for a system as well as detailed specifications of the system requirements.

IEEE defines software requirements specification as, 'a document that clearly and precisely describes each of the essential requirements (functions, performance, design constraints and quality attributes) of the software and the external interfaces. Each requirement is defined in such a way that its achievement can be objectively verified by a prescribed method, for example, inspection, demonstration, analysis or test.' Note that requirements specification can be in the form of a written document, a mathematical model, a collection of graphical models, a prototype, and so on.

Essentially, what passes from requirements analysis activity to the specification activity is the knowledge acquired about the system. The need for maintaining a requirements document is that the modeling activity essentially focuses on the problem structure and not its structural behavior. While in SRS, performance constraints, design constraints, and standard compliance recovery are clearly specified. This information helps in developing a proper design of the system. Various other purposes served by SRS are listed below.

1. **Feedback:** Provides a feedback, which ensures to the user that the organization (which develops the software) understands the issues or problems to be solved and the software behavior necessary to address those problems.
2. **Decompose problem into components :** Organizes the information and divides the problem into its component parts in an orderly manner.
3. **Validation :** Uses validation strategies applied to the requirements to acknowledge that requirements are stated properly.
4. **Input to design:** Contains sufficient detail in the functional system requirements to devise a design solution.
5. **Basis for agreement between the user and the organization :** Provides a complete description of the functions to be performed by the system. In addition, it helps the users to determine whether the specified requirements are accomplished.
6. **Reduce the development effort:** Enables developers to consider user requirements before the designing of the system commences. As a result, 'rework' and inconsistencies in the later stages can be reduced.
7. **Estimating costs and schedules :** Determines the requirements of the system and thus enables the developer to have a 'rough' estimate of the total cost and schedule of the project.

SRS is used by various individuals in the organization. System customers need SRS to specify and verify whether requirements meet the desired needs. In addition, SRS enables the managers to plan for the system development processes. System engineers need a requirements document to understand what system is to be developed. These engineers also require this document to develop validation tests for the required system. Lastly, requirements document is needed by system maintenance engineers to use the requirement and the relationship between its parts.


SRS-Users
System Maintaince Engineering
System test Engineers
System Engineers
Manager
System customers
software Requirements Specification Document

The requirements document has diverse users. Therefore, along with communicating the requirements to the users it also has to define the requirements in precise detail for developers and testers. In addition it should also include information about possible changes in the system, which can help system designers avoid restricted decisions on design. SRS also helps maintenance engineers to adapt the system to new requirements.

Characteristics

Software requirements specification should be accurate, complete, efficient, and of high quality, so that it does not affect the entire project plan. An SRS is said to be of high quality when the developer and user easily understand the prepared document. Other characteristics of SRS are discussed below.

1. **Correct:** SRS is correct when all user requirements are stated in the requirements document. The stated requirements should be according to the desired system. This implies that each requirement is examined to ensure that it (SRS) represents user requirements. Note that there is no specified tool or procedure to assure the correctness of SRS. Correctness ensures that all specified requirements are performed correctly.

2. **Unambiguous:** SRS is unambiguous when every stated requirement has only one interpretation. This implies that each requirement is uniquely interpreted. In case there is a term used with multiple meanings, the requirements document should specify the meanings in the SRS so that it is clear and easy to understand.

3. **Complete:** SRS is complete when the requirements clearly define what the software is required to do. This includes all the requirements related to performance, design and functionality.

4. **Ranked for importance/stability:** All requirements are not equally important, hence each requirement is identified to make differences among other requirements. For this, it is essential to clearly identify each requirement. Stability implies the probability of changes in the requirement in future.

5. **Modifiable:** The requirements of the user can change, hence requirements document should be created in such a manner that those changes can be modified easily, consistently maintaining the structure and style of the SRS.

6. **Traceable:** SRS is traceable when the source of each requirement is clear and facilitates the reference of each requirement in future. For this, forward tracing and backward tracing are used. Forward tracing implies that each requirement should be traceable to design and code elements. Backward tracing implies defining each requirement explicitly referencing its source.

7. **Verifiable:** SRS is verifiable when the specified requirements can be verified with a cost-effective process to check whether the final software meets those requirements. The requirements are verified with the help of reviews. Note that unambiguity is essential for verifiability.

8. **Consistent:** SRS is consistent when the subsets of individual requirements defined do not conflict with each other. For example, there can be a case when different requirements can use different terms to refer to the same object. There can be logical or temporal conflicts between the specified requirements and some requirements whose logical or temporal characteristics are not satisfied. For instance, a requirement states that an event 'a' is to occur before another event 'b'. But then another set of requirements states (directly or indirectly by transitivity) that event 'b' should occur before event 'a'.

Structure

The requirements document is devised in a manner that is easier to write, review, and maintain. It is organized into independent sections and each section is organized into modules or units. Note that the level of detail to be included in the SRS depends on the type of the system to be developed and the process model chosen for its development. For example, if a system is to be developed by an external contractor, then critical system specifications need to be precise and detailed. Similarly, when flexibility is required in the requirements and where an in-house development takes place, requirements documents can be less detailed.

Since the requirements document serves as a foundation for subsequent software development phases, it is important to develop the document in the prescribed manner. For this, certain guidelines are followed while preparing SRS. These guidelines are listed below.

1. **Functionality :** It should be separate from implementation.
2. **Analysis model :** It should be developed according to the desired behavior of a system. This should include data and functional response of a system to various inputs given to it.
3. **Cognitive model :** It should be developed independently of design or implementation model. This model expresses a system as perceived by the users.
4. **The content and structure of the specification :** It should be flexible enough to accommodate changes.
5. **Specification :** It should be robust. That is, it should be tolerant towards incompleteness and complexity.

The information to be included in SRS depends on a number of factors, for example, the type of software being developed and the approach used in its development. If software is developed using the iterative development process, the requirements document will be less detailed as compared to that of the software developed for critical systems. This is because specifications need to be very detailed and accurate in these systems. A number of standards have been suggested to develop a requirements document. However, the most widely used standard is by IEEE, which acts as a general framework. This general framework can be customized and adapted to meet the needs of a particular organization.

Each SRS fits a certain pattern; thus, it is essential to standardize the structure of the requirements document to make it easier to understand. For this IEEE standard is used for SRS to organize requirements for different projects, which provides different ways of structuring SRS. Note that in all requirements documents, the first two sections are the same.

This document comprises the following sections.

1. **Introduction:** This provides an overview of the entire information described in SRS. This involves purpose and the scope of SRS, which states the functions to be performed by the system. In addition, it describes definitions, abbreviations, and the acronyms used. The references used in SRS provide a list of documents that is referenced in the document.
2. **Overall description:** It determines the factors which affect the requirements of the system. It provides a brief description of the requirements to be defined in the next section called 'specific requirement'. It comprises the following sub-sections.

3. **Product perspective:** It determines whether the product is an independent product or an integral part of the larger product. It determines the interface with hardware, software, system, and communication. It also defines memory constraints and operations utilized by the user.

4. **Product functions:** It provides a summary of the functions to be performed by the software. The functions are organized in a list so that they are easily understandable by the user.

5. **User characteristics :** It determines general characteristics of the users.

6. **Constraints:** It provides the general description of the constraints such as regulatory policies, audit functions, reliability requirements, and so on.

7. **Assumption and dependency:** It provides a list of assumptions and factors that affect the requirements as stated in this document.

8. **Apportioning of requirements:** It determines the requirements that can be delayed until release of future versions of the system.

9. **Specific requirements:** These determine all requirements in detail so that the designers can design the system in accordance with them. The requirements include description of every input and output of the system and functions performed in response to the input provided. It comprises the following subsections.

10. **External interface :** It determines the interface of the software with other systems, which can include interface with operating system and so on. External interface also specifies the interaction of the software with users, hardware, or other software. The characteristics of each user interface of the software product are specified in SRS. For the hardware interface, SRS specifies the logical characteristics of each interface among the software and hardware components. If the software is to be executed on the existing hardware, then characteristics such as memory restrictions are also specified.

11. **Functions :** It determines the functional capabilities of the system. For each functional requirement, the accepting and processing of inputs in order to generate outputs are specified. This includes validity checks on inputs, exact sequence of operations, relationship of inputs to output, and so on.

12. **Performance requirements :** It determines the performance constraints of the software system. Performance requirement is of two types: static requirements and dynamic requirements. Static requirements (also known as capacity requirements) do not impose constraints on the execution characteristics of the system. These include requirements like number of terminals and users to be supported. Dynamic requirements determine the constraints on the execution of the behavior of the system, which includes response time (the time between the start and ending of an operation under specified conditions) and throughput (total amount of work done in a given time).

13. **Logical database of requirements:** It determines logical requirements to be stored in the database. This includes type of information used, frequency of usage, data entities and relationships among them, and so on.

14. **Design constraint:** It determines all design constraints that are imposed by standards, hardware limitations, and so on. Standard compliance determines requirements for the system, which are in compliance with the specified standards. These standards can include accounting procedures and report format. Hardware limitations implies when the software can operate on existing hardware or some pre-determined hardware. This can impose restrictions while developing the software design. Hardware limitations include hardware configuration of the machine and operating system to be used.

15. **Software system attributes:** It provide attributes such as reliability, availability, maintainability and portability. It is essential to describe all these attributes to verify that they are achieved in the final system.

16. **Organizing Specific Requirements:** It determines the requirements so that they can be properly organized for optimal understanding. The requirements can be organized on the basis of mode of operation, user classes, objects, feature, response, and functional hierarchy.

17. **Change management process:** It determines the change management process in order to identify, evaluate, and update SRS to reflect changes in the project scope and requirements.

18. **Document approvals:** These provide information about the approvers of the SRS document with the details such as approver's name, signature, date, and so on.

19. **Supporting information:** It provides information such as table of contents, index, and so on. This is necessary especially when SRS is prepared for large and complex projects.

Q7. Describe Various advantages of an SRS. What are various types of errors that occur in SRS?

Ans :

Software requirement specification (SRS) is a document that completely describes what the proposed software should do without describing how software will do it. The basic goal of the requirement phase is to produce the SRS, Which describes the complete behavior of the proposed software. SRS is also helping the clients to understand their own needs.

Advantages

Software SRS establishes the basic for agreement between the client and the supplier on what the software product will do.

1. A SRS provides a reference for validation of the final product.
2. A high-quality SRS is a prerequisite to high-quality software.
3. A high-quality SRS reduces the development cost.

Types of Errors that Occur in SRS

- **Omission** : is a common error in requirements. In this type of error, some user requirements are simply not included in the SRS The omitted requirement may be related to the behavior of the system, its performance, constraints or any other factor.
- **Inconsistency** : It can be due to contradictions within the requirements themselves or to incompatibility of the stated requirements with the actual requirements of the client or with the environment in which the system will operate.
- **Incorrect fact** : This error occurs when some fact recorded in the SRS is not correct.
- **Ambiguity** : This error occurs when there are some requirements that have multiple meanings, that is, their interpretation is not unique.

2.4.1 Components of SRS

Q8. List & Explain various components of an SRS.

Ans :

Completeness of specifications is difficult to achieve and even more difficult to verify. Having guidelines about what different things an SRS should specify will help in completely specifying the requirements. Here we describe some of the system properties than an SRS should specify.

The basic issues an SRS must address :

I) Functional Requirements

II) Design constraints

III) External interfaces Requirements

I) Functional Requirements

1. Which outputs should be produced from the given inputs?
2. Relationship between the input and output.
3. A detailed description of all the data inputs and their source, the units of measure.
4. The range of valid inputs.

II) Design Constraints

1. Standards that must be followed.
2. Resource limits & operating environment.
3. Reliability
4. Security requirement
5. Policies that may have an impact on the design of the system.

Standards Compliance

This specifies the requirements for the standards that the system must follow.

Hardware Limitations

The software may have to operate on some existing or predetermined hardware thus imposing restrictions on the design.

- Reliability and Fault Tolerance:
- Fault tolerance requirements can place a major constraint on how the system is to be designed. Fault tolerance requirements often make the system more complex and expensive.
- **Security :** Security requirements are particularly significant in defense systems and many database systems. Security requirements place restrictions on the use of certain commands, control access to data, provide different kinds of access requirements for different people require the use of passwords and cryptography techniques and maintain a log of activities in the system.
- **External Interface Requirements:** All the possible interactions of the software with people, hardware and other software should be clearly specified. For the user interface, the characteristics of each user interface of the software product should be specified. User interface is becoming increasingly important and must be given proper attention. A preliminary user manual should be created with all use commands, screen formats and explanation of how the system will appear to the user, and feedback and error message.

Like other specifications these requirements should be precise and verifiable. So a statement likes "the system should be no longer than six characters" or command names should reflect the function they perform used. If the software is to execute on existing hardware or on predetermined hardware, all the characteristics of the hardware, including memory restrictions, should be specified. In addition, the current use and load characteristics of the hardware should be given.

Q9. Discuss briefly the validation of SRS.

Ans : **(Imp.)**

The development of software starts with the requirements document, which is also used to determine eventually whether or not the delivered software system is acceptable. It is therefore important that the requirement specification contains no error and specifies the client's requirement correctly.

Furthermore due to the nature of the requirement specification phase, there is a lot of room for misunderstanding and committing errors, and it is quite possible that the requirements specification does not accurately represents the client's needs. The basic objective of the requirement validation activity is to ensure that SRS reflects the actual requirements accurately and clearly. A related objective is to check that the SRS documents are itself of "good quality" (Some desirable quality objectives are given later).

Many different types of errors are possible , but the most common errors that occurs can be classified in four types :

i) Omission

ii) Inconsistency

iii) Incorrect fact

iv) Ambiguity

i) **Omission:** Omission is a common error in requirements. In this type of error, some user requirements is simply not included in the SRS; the omitted requirement may be related to the behavior of the system, its performance, constraints, or any other factor.

Omission directly affects the external completeness of the SRS. Another common form of error in requirement is inconsistency.

ii) **Inconsistency:** Inconsistency can be due to contradictions within the requirements themselves or due to incompatibility of the started requirements with the actual requirements of the client or with the environment in which the system will operate.

iii) **Incorrect fact:** The third common requirement error is incorrect fact. Errors of this type occur when some facts recorded in the SRS are incorrect.

iv) **Ambiguity:** The fourth common error type is ambiguity. Errors of this type occur when there are some requirements that have multiple meanings that is their interpretation is not unique.

Omission	**Incorrect Fact**	**Inconsistency**	**Ambiguity**
26%	10%	38%	26

In the errors detected in the requirement specification of the A-7 project (which deals with a real time flight control software) were reported. A total of about 80 errors were detected. Out of the which, about 23% were clerical in nature, of the remaining the distribution with error type was :

Omission	**Incorrect Fact**	**Inconsistency**	**Ambiguity**
32%	49%	13%	5%

Apart from Requirement Reviews what are the other Methods Used for the Validation of SRS

Requirement reviews remain the most commonly used and viable means for requirement validation. However, there are other approaches that may be applicable for some system or parts of system or system that have been specified formally.

Automated Cross Referencing

Automated cross-referencing uses processors to verify some properties of requirements. Any automated processing of requirements is possible if the requirements are written in a formal specification language or a language specifically designed for machine processing.

We saw example of such language earlier. These tools typically focus on checks for internal consistency and completeness, which sometimes leads to checking of external completeness. However, these tools cannot directly checks for external completeness. For this reason, requirement reviews are needed even if the requirements are specified through a tool or in a formal notation.

If the requirements are in machine process able form, they can be analyzed for internal consistency among different elements of the requirement.

Reading

The goal in reading is to have someone other than the author or the requirements read the requirement specification document to identify potential problems. Having the requirement read by another person who may have different interpretation of requirements, many of the requirements problems caused by misinterpretations or ambiguities can be identified. Furthermore, if the reader is a person who is in interested in the project (like a person from the quality assurance group that will eventually test the system) can address issues that could cause problem later.

For example, if a tester reads the requirement, it is likely that the testability of requirement will be well examined.

Constructing Scenarios

Scenarios describe different situations of how the system will work once it is operational. The most common area for constructing scenarios is that of system user interaction. Constructing scenarios is good for clarifying misunderstandings in human computer interaction area. They are of limited value for verifying the consistency and completeness of requirements;

- **Prototyping :** Though prototypes are generally built to ascertain requirements, a prototype can be built to verify requirements. Prototype can be quite useful in verifying the feasibility of some of the requirement (such as answering the question Can this be done?) A prototype that has been built during problem analysis can also aid validation. For example, if the prototype has a use interfaces and the client has approved them after use, then the user interface, as specified by the prototype, can be considered validated. No further validation need be performed for user interface.

- **Metrics :** The basic purpose of metrics at any point during a development project is to provide qualitative information of the management process so that the information can be used effectively to control the development process.

Q10. Explain the SRS with use case Modeling.

Ans :

Many software teams are discovering that mixing use-case modeling techniques for requirements expression along with traditional methods of documenting specific requirements within a “software requirements specification” (SRS) document provides an efficient means to record the complete set of detailed requirements for a system or application to be built. The Rational Unified Process provides the following definitions:

- A requirement describes a condition or capability to which a system must conform; either derived directly from user needs, or stated in a contract, standard, specification, or other formally imposed document.

- A use case describes a sequence of actions a system performs that yields an observable result of value to a particular actor.

In our experience, we have found that a fully specified set of use cases for a system often does a great job of stating many of the requirements for that system. But just as often there are also a significant number of requirements that do not fit well within this modeling technique. Especially for non-functional requirements (e.g., specifications for usability, reliability, performance, maintainability, supportability), it is usually best to use the good-ol' tried-and-true traditional method for stating requirements.

Traditionally, requirements specified in an SRS are simple declarative statements written in a text-based natural-language style (e.g., " When the user logs in, the system displays the splash screen described in Figure x."). Use cases should be viewed merely as an alternative for specifying requirements; moreover they describe complete threads, or scenarios, through the system which provide additional value in understanding the intended behavior of the system. Use-case modeling is a hallmark of the Unified Modeling Language (UML) and the Rational Unified Process (RUP) as well as being a central feature provided by visual modeling tools, such as Rational Rose. In most circumstances use cases should simply be applied as an added form of expression which increases understandability, as opposed to simply replacing the specification of requirements in the traditional fashion.

We need to be able to support the traditional requirements approach in all of our processes, specifications and tools since it may well will be needed whether or not a project decides to use the use-case methodology (e.g., in specifying requirements that do not fit easily in the use-case model, especially non-functional requirements). It does not have to be an "either-or". Neither approach is "best" in all circumstances - although it could definitely be argued as to which approach might be "better" for a particular application.

A common concern that we hear from development teams first trying to apply use-case modeling is that they do not want to maintain redundant specifications of their requirements - one expression in use-case form and another in the traditional SRS form. They like use cases for working with their customers to specify the functional requirements of a system in the customers' language, however they are often more familiar with the more traditional form of expression (and perhaps they are required by their boss, or management or an outside agency) to produce a Software Requirements Specification (SRS) that contains the "complete" set of detailed software requirements for the system. A key concept of good requirements management advocates minimizing redundancy, so this is definitely a valid concern that should be addressed.

To facilitate the combination of traditional SRS methods and use cases, we have defined a simple construct called the "SRS Package". This "package" pulls together the complete set of software requirements for a system (or portion of a system) which may be contained in a single document, multiple documents, a requirements repository (consisting of the requirements' text, attributes and traceability), use case specifications and even the graphical use case model diagrams which describes relationships amongst the use cases and actors. We've found this simple combination to be effective in most all application development projects, and it's an easy way to kickstart your teams ability to realize the additional benefits provided by the use case technique.

Purpose

The SRS provides an organized way to collect all software requirements surrounding a project (or perhaps at a subsystem or feature level) into one document. In these days of automation and visual modeling, we often find ourselves with several different tools for collecting these requirements. The complete set of requirements may indeed be found in several different artifacts accessible by multiple tools. For example, you might find it useful to collect some (or all) of the functional requirements of your system in use cases and you might find it handy to use a tool appropriate to the needs of defining the use-case model. On the other hand, you might find it appropriate to collect traditionally stated natural-language requirements (such as non-functional requirements, design constraints, etc.) with a word processing tool in "supplementary specifications". And a requirements management tool must be able to access all requirements for maintaining requirement attributes and traceability.

Arguably, some combination of the use case and traditional specification technique should be used in all projects. To facilitate this, we will collect the requirements for our SRS in a UML "package" construct that may include a single document, multiple documents, use case specifications and even the graphical use case model which describes relationships amongst the use cases.

The “SRS package” controls the evolution of the system throughout the development and release phases of the project. In the Rational Unified Process, the following workers use the SRS Package :

➢ The system analyst creates and maintains the Vision, the use-case model overview and supplementary specifications, which serve as input to the SRS and are the communication medium between the system analyst, the customer, and other developers.

➢ The use-case specifier creates and maintains the individual use cases of the use-case model and other components of the SRS package,

➢ Designers use the SRS Package as a reference when defining responsibilities, operations, and attributes on classes, and when adjusting classes to the implementation environment.

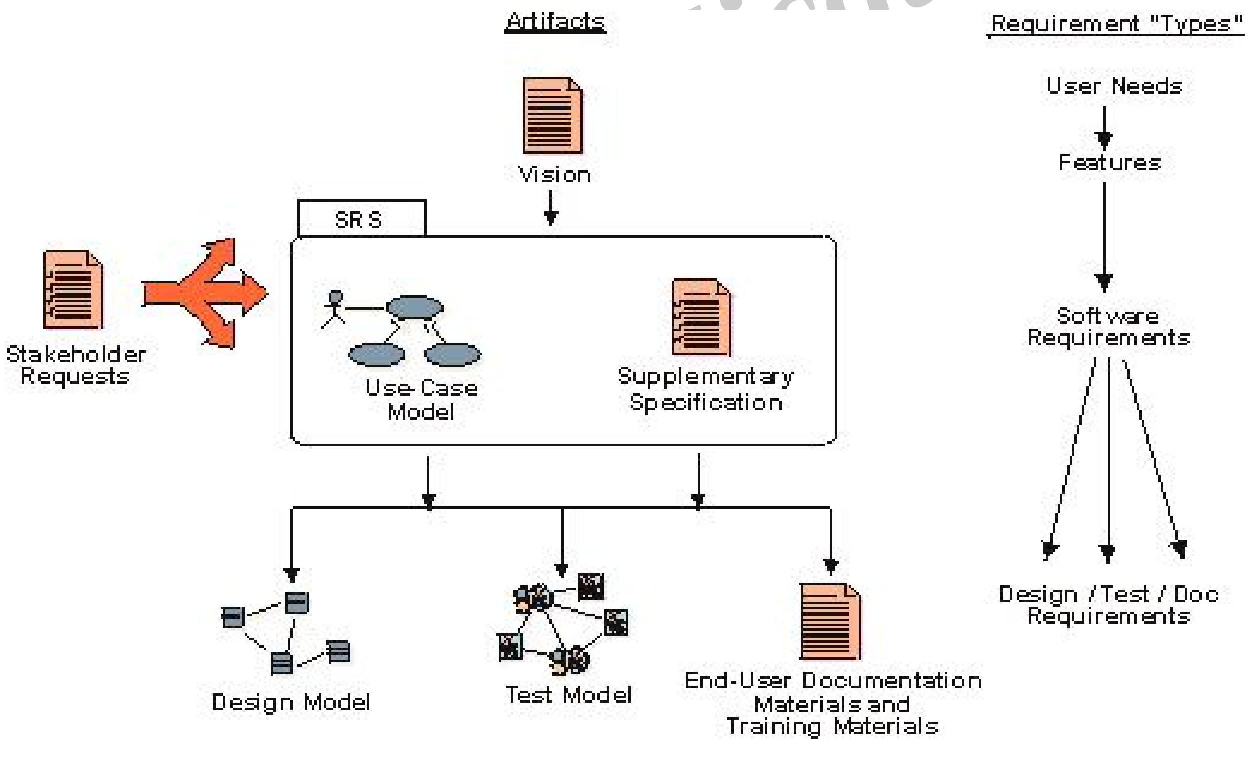
➢ Implementers refer to the SRS Package for input when implementing classes.

➢ The project manager refers to the SRS Package for input when planning iterations.

➢ The testers use the SRS Package to verify system compliance.

From Vision to SRS

In the Requirements workflow of the Rational Unified Process, our requirements artifact structure starts with an artifact called the Vision which describes the user’s or customer’s view of the product to be developed, specified at the level of key user needs and features of the system.



The SRS Package is obviously related to the Vision document. Indeed, the Vision document serves as the input to the SRS. But the two artifacts serve different needs and are typically written by different authors. At this stage in the project, the focus of the project moves from the broad statement of user needs, goals and objectives, target markets and features of the system to the details of how these features are going to be implemented in the solution.

What we need now is a collection, or package, of artifacts that describes the complete external behavior of the system - i.e., an artifact that says specifically: "Here is what the system has to do to deliver those features." That is what we refer to as the SRS Package.

A Living Artifact

The SRS Package is an active, living artifact. Indeed it plays a number of crucial roles as the developers embark upon their implementation effort: It serves as a basis of communication between all parties - i.e., between the developers themselves, and between the developers and the external groups (users and other stakeholders) with whom they must communicate. Formally or informally, it represents a contractual agreement between the various parties - if it's not in the SRS Package, the developers shouldn't be working on it. And if it is in the SRS Package, then they are accountable to deliver that functionality.

The Project Manager's Reference Standard

The SRS serves as the project manager's reference standard. The project manager is unlikely to have the time, energy, or skills to read the code being generated by the developers and compare that directly to the Vision document. He or she must use the SRS as the standard reference for discussions with the project team.

As noted earlier, it serves as input to the design and implementation groups. Depending on how the project is organized, the implementers may have been involved in the earlier problem-solving and feature-definition activities that ultimately produced the Vision document. But it's the SRS they need to focus on for deciding what their code must do. It serves as input to the software testing and QA groups. These groups should also have been involved in some of the earlier discussions, and it's obviously helpful for them to have a good understanding of the "vision" laid out in the Vision documents. But their charter is to create test cases and QA procedures to ensure that the developed system does indeed fulfill the requirements laid out in the SRS. The SRS also serves as the standard reference for their planning and testing activities.

Sample Templates for an SRS Package

In support of these concepts, two separate SRS document templates have been developed and are enclosed :

- An SRS template for use with projects using use-case modeling, and
- An SRS template for projects using only traditional requirements specification techniques.

See the Rational Unified Process (v5.5) for more information on these and other artifacts, along with best practices for developing software with UML.

2.5 Functional Specification with Use Cases

Q11. Discuss about functional specifications using use cases.

Ans : **(Imp.)**

Functional requirements often form the core of a requirements document. The traditional approach for specifying functionality is to specify each function that the system should provide. Use cases specify the functionality of a system by specifying the behavior of the system, captured as interactions of the users with the system. Use cases can be used to describe the business processes of the larger business or organization that deploys the software, or it could just describe the behavior of the software system. We will focus on describing the behavior of software systems that are to be built.

Though use cases are primarily for specifying behavior, they can also be used effectively for analysis. Later when we discuss how to develop use cases, we will discuss how they can help in eliciting requirements also.

Use cases drew attention after they were used as part of the object-oriented modeling approach proposed by Jacobson [56]. Due to this connection with an object-oriented approach, use cases are sometimes viewed as part of an object-oriented approach to software development. However, they are a general method for describing the interaction of a system (even non-IT systems).

Basics

A software system (in our case whose requirements are being uncovered) may be used by many users, or by other systems. In use case terminology, an actor is a person or a system which uses the system for achieving some goal. Note that as an actor interacts for achieving some goal, it is a logical entity that represents a group of users (people or system) who behave in a similar manner. Different actors represent groups with different goals. So, it is better to have a “receiver” and a “sender” actor rather than having a generic “user” actor for a system in which some messages are sent by users and received by some other users.

A primary actor is the main actor that initiates a use case (UC) for achieving a goal, and whose goal satisfaction is the main objective of the use case. The primary actor is a logical concept and though we assume that the primary actor executes the use case, some agent may actually execute it on behalf of the primary actor. For example, a VP may be the primary actor for get sales growth report by region use case, though it may actually be executed by an assistant. We consider the primary actor as the person who actually uses the outcome of the use case and who is the main consumer of the goal. Time-driven trigger is another example of how a use case may be executed on behalf of the primary actor (in this situation the report is generated automatically at some time).

Note, however, that although the goal of the primary actor is the driving force behind a use case, the use case must also fulfill goals that other stakeholders might have for this use case. That is, the main goal of a use case is to describe behavior of the system that results in satisfaction of the goals of all the stakeholders, although the use case may be driven by the goals of the primary actor. For example, a use case “Withdraw money from the ATM” has a customer as its primary actor and will normally describe the entire interaction of the customer with the ATM. However, the bank is also a stakeholder of the ATM system and its goals may include that all steps are logged, money is given only if there are sufficient funds in the account, and no more than some amount is given at a time, etc. Satisfaction of these goals should also be shown by the use case “Withdraw money from the ATM.”

For describing interaction, use cases use scenarios. A scenario describes a set of actions that are performed to achieve a goal under some specified conditions. The set of actions is generally specified as a sequence (as that is the most convenient way to express it in text), though in actual execution the actions specified may be executed in parallel or in some different order. Each step in a scenario is a logically complete action performed either by the actor or the system. Generally, a step is some action by the actor (e.g., enter information), some logical step that the system performs to progress toward achieving its goals (e.g., validate information, deliver information), or an internal state change by the system to satisfy some goals (e.g., log the transaction, update the record).

A use case always has a main success scenario, which describes the interaction if nothing fails and all steps in the scenario succeed. There may be many success scenarios. Though the use case aims to achieve its goals, different situations can arise while the system and the actor are interacting which may not permit the system to achieve the goal fully. For these situations, a use case has extension scenarios which describe the system behavior if some of the steps in the main scenario do not complete successfully. Sometimes they are also called exception scenarios. A use case is a collection of all the success and extension scenarios related to the goal. The terminology of use cases is summarized in Table.

Term	Definition
Actor	A person or a system which uses the system being built for achieving some goal.
Primary actor	The main actor for whom a use case is initiated and whose goal satisfaction is the main objective of the use case.
Scenario	A set of actions that are performed to achieve a goal under some specified conditions.
Main success scenario	Describe the interaction if nothing fails and all steps in the scenario succeed.
Extension scenario	Describe the system behaviour if some of the steps in the main scenario do not complete successfully

To achieve the desired goal, a system can divide it into subgoals. Some of these subgoals may be achieved by the system itself, but they may also be treated as separate use cases executed by supporting actors, which may be another system. For example, suppose for verifying a user in "Withdraw money from the ATM" an authentication service is used. The interaction with this service can be treated as a separate use case. A scenario in a use case may therefore employ another use case for performing some of the tasks. In other words, use cases permit a hierarchic organization.

It should be evident that the basic system model that use cases assume is that a system primarily responds to requests from actors who use the system. By describing the interaction between actors and the system, the system behavior can be specified, and through the behavior its functionality is specified. A key advantage of this approach is that use cases focus on external behavior, thereby cleanly avoiding doing internal design during requirements, something that is desired but not easy to do with many modeling approaches.

Use cases are naturally textual descriptions, and represent the behavioral requirements of the system. This behavior specification can capture most of the functional requirements of the system. Therefore, use cases do not form the complete SRS, but can form a part of it. The complete SRS, as we have seen, will need to capture other requirements like performance and design constraints.

Though the detailed use cases are textual, diagrams can be used to supplement the textual description. For example, the use case diagram of UML provides an overview of the use cases and actors in the system and their dependency. A UML use case diagram generally shows each use case in the system as an ellipse, shows the primary actor for the use case as a stick figure connected to the use case with a line, and shows dependency between use cases by arcs between use cases. Some other relationships between use cases can also be represented. However, as use cases are basically textual in nature, diagrams play a limited role in either developing or specifying use cases. We will not discuss use case diagrams further

2.5.1 Developing Use Cases

Q12. Explain the process of developing use Cases.

Ans :

UCs not only document requirements, as their form is like storytelling and uses text, both of which are easy and natural with different stakeholders, they also are a good medium for discussion and brainstorming. Hence, UCs can also be used for requirements elicitation and problem analysis. While developing use cases, informal or formal models may also be built, though they are not required.

UCs can be evolved in a stepwise refinement manner with each step adding more details. This approach allows UCs to be presented at different levels of abstraction. Though any numbers of levels of abstraction are possible, four natural levels emerge :

- **Actors and goals.** The actor-goal list enumerates the use cases and specifies the actors for each goal. (The name of the use case is generally the goal.) This table may be extended by giving a brief description of each of the use cases. At this level, the use cases together specify the scope of the system and give an overall view of what it does. Completeness of functionality can be assessed fairly well by reviewing these.
- **Main success scenarios.** For each of the use cases, the main success scenarios are provided at this level. With the main scenarios, the system behavior for each use case is specified. This description can be reviewed to ensure that interests of all the stakeholders are met and that the use case is delivering the desired behavior.
- **Failure conditions**. Once the success scenario is listed, all the possible failure conditions can be identified. At this level, for each step in the main success scenario, the different ways in which a step can fail form the failure conditions. Before deciding what should be done in these failure conditions (which is done at the next level), it is better to enumerate the failure conditions and review for completeness.

- **Failure handling:** This is perhaps the most tricky and difficult part of writing a use case. Often the focus is so much on the main functionality that people do not pay attention to how failures should be handled. Determining what should be the behavior under different failure conditions will often identify new business rules or new actors.

The different levels can be used for different purposes. For discussion on overall functionality or capabilities of the system, actors and goal-level description is very useful. Failure conditions, on the other hand, are very useful for understanding and extracting detailed requirements and business rules under special cases.

Steps

These four levels can also guide the analysis activity. A step-by-step approach for analysis when employing use cases is :

- **Step 1:** Identify the actors and their goals and get an agreement with the concerned stakeholders as to the goals. The actor-goal list will clearly define the scope of the system and will provide an overall view of what the system capabilities are.
- **Step 2:** Understand and specify the main success scenario for each UC, giving more details about the main functions of the system. Interaction and discussion are the primary means to uncover these scenarios though models may be built, if required. During this step, the analyst may uncover that to complete some use case some other use cases are needed, which have not been identified. In this case, the list of use cases will be expanded.
- **Step 3:** When the main success scenario for a use case is agreed upon and the main steps in its execution are specified, then the failure conditions can be examined. Enumerating failure conditions is an excellent method of uncovering special situations that can occur and which must be handled by the system.
- **Step 4:** Finally, specify what should be done for these failure conditions. As details of handling failure scenarios can require a lot of effort and discussion, it is better to first enumerate the different failure conditions and then get the details of these scenarios. Very often, when deciding the failure scenarios, many new business rules of how to deal with these scenarios are uncovered.

Though we have explained the basic steps in developing use cases, at any step an analyst may have to go back to earlier steps as during some detailed analysis new actors may emerge or new goals and new use cases may be uncovered. That is, using use cases for analysis is also an interactive task.

What should be the level of detail in a use case? There is no one answer to a question like this; the actual answer always depends on the project and the situation. So it is with use cases. Generally it is good to have sufficient details which are not overwhelming but are sufficient to build the system and meet its quality goals. For example, if there is a small collocated team building the system, it is quite likely that use cases which list the main exception conditions and give a few key steps for the scenarios will suffice. On the other hand, for a project whose development is to be subcontracted to some other organization, it is better to have more detailed use cases.

For writing use cases, general technical writing rules apply. Use simple grammar, clearly specify who is performing the step, and keep the overall scenario as simple as possible. Also, when writing steps, for simplicity, it is better to combine some steps into one logical step, if it makes sense. For example, steps "user enters his name," "user enters his SSN," and "user enters his address" can be easily combined into one step "user enters personal information."

2.6 Other Approaches for Analysis

Q13. Explain different approaches of Analysis in Software Engineering.

Ans : **(Imp.)**

The basic aim of problem analysis is to obtain a clear understanding of the needs of the clients and the users, what exactly is desired from the software, and what the constraints on the solution are. Frequently the client and the users do not understand or know all their needs, because the potential of the new system is often not fully appreciated. The analysts have to ensure that the real needs of the clients and the users are uncovered, even if they don't know them clearly. That is, the analysts are not just collecting and organizing information about the client's organization and its processes, but they also act as consultants who play an active role of helping the clients and users identify their needs.

The basic principle used in analysis is the same as in any complex task: divide and conquer. That is, partition the problem into subproblems and then try to understand each subproblem and its relationship to other subproblems in an effort to understand the total problem. The concepts of state and projection can sometimes also be used effectively in the partitioning process. A state of a system represents some conditions about the system. Frequently, when using state, a system is first viewed as operating in one of the several possible states, and then a detailed analysis is performed for each state. This approach is sometimes used in real-time software or process-control software.

In projection, a system is defined from multiple points of view [86]. While using projection, different viewpoints of the system are defined and the system is then analyzed from these different perspectives. The different "projections" obtained are combined to form the analysis for the complete system. Analyzing the system from the different perspectives is often easier, as it limits and focuses the scope of the study.

In the remainder of this section we will discuss two other methods for problem analysis. As the goal of analysis is to understand the problem domain, an analyst must be familiar with different methods of analysis and pick the approach that he feels is best suited to the problem at hand.

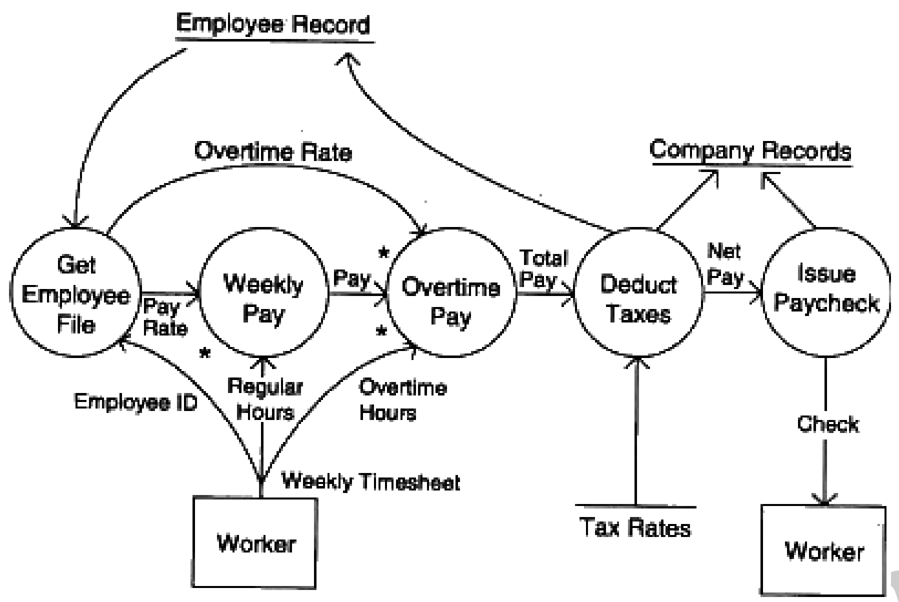
i) Data Flow Diagrams

Data flow diagrams (also called data flow graphs) are commonly used during problem analysis. Data flow diagrams (DFDs) are quite general and are not limited to problem analysis for software requirements specification. They were in use long before the software engineering discipline began. DFDs are very useful in understanding a system and can be effectively used during analysis.

A DFD shows the flow of data through a system. It views a system as a function that transforms the inputs into desired outputs. Any complex system will not perform this transformation in a "single step," and data will typically undergo a series of transformations before it becomes the output. The DFD aims to capture the transformations that take place within a system to the input data so that eventually the output data is produced. The agent that performs the transformation of data from one state to another is called a process (or a bubble). Thus, a DFD shows the movement of data through the different transformations or processes in the system. The processes are shown by named circles and data flows are represented by named arrows entering or leaving the bubbles. A rectangle represents a source or sink and is a net originator or consumer of data. A source or a sink is typically outside the main system of study.

In this DFD there is one basic input data flow, the weekly timesheet, which originates from the source worker. The basic output is the paycheck, the sink for which is also the worker. In this system, first the employee's record is retrieved, using the employee ID, which is contained in the timesheet. From the employee record, the rate of payment and overtime are obtained. These rates and the regular and overtime hours (from the timesheet) are used to compute the pay. After the total pay is determined, taxes are deducted. To compute the tax deduction, information from the tax-rate file is used. The amount of tax deducted is recorded in the employee and company records. Finally, the paycheck is issued for the net pay. The amount paid is also recorded in company records.

Some conventions used in drawing this DFD should be explained. All external files such as employee record, company record, and tax rates are shown as a labeled straight line. The need for multiple data flows by a process is represented by a "*" between the data flows. This symbol represents the AND relationship. For example, if there is a "*" between the two input data flows A and B for a process, it means that A AND B are needed for the process. In the DFD, for the process "weekly pay" the data flow "hours" and "pay rate" both are needed, as shown in the DFD. Similarly, the OR relationship is represented by a "+" between the data flows.



This DFD is an abstract description of the system for handling payment. It does not matter if the system is automated or manual. This diagram could very well be for a manual system where the computations are all done with calculators, and the records are physical folders and ledgers. The details and minor data paths are not represented in this DFD. For example, what happens if there are errors in the weekly timesheet is not shown in this DFD. This is done to avoid getting bogged down with details while constructing a DFD for the overall system. If more details are desired, the DFD can be further refined. It should be pointed out that a DFD is not a flowchart. A DFD represents the flow of data, while a flowchart shows the flow of control. A DFD does not represent procedural information. So, while drawing a DFD, one must not get involved in procedural details, and procedural thinking must be consciously avoided. For example, considerations of loops and decisions must be ignored. In drawing the DFD, the designer has to specify the major transforms in the path of the data flowing from the input to output. How those transforms are performed is not an issue while drawing the data flow graph.

Many systems are too large for a single DFD to describe the data processing clearly. It is necessary that some decomposition and abstraction mechanism be used for such systems. DFDs can be hierarchically organized, which helps in progressively partitioning and analyzing large systems. Such DFDs together are called a leveled DFD set.

A leveled DFD set has a starting DFD, which is a very abstract representation of the system, identifying the major inputs and outputs and the major processes in the system. Often, before the initial DFD, a context diagram may be drawn in which the entire system is shown as a single process with all its inputs, outputs, sinks, and sources. Then each process is refined and a DFD is drawn for the process. In other words, a bubble in a DFD is expanded into a DFD during refinement. For the hierarchy to be consistent, it is important.

That the net inputs and outputs of a DFD for a process are the same as the inputs and outputs of the process in the higher-level DFD. This refinement stops if each bubble is considered to be “atomic,” in that each bubble can be easily specified or understood. It should be pointed out that during refinement, though the net input and output are preserved, a refinement of the data might also occur. That is, a unit of data may be broken into its components for processing when the detailed DFD for a process is being drawn. So, as the processes are decomposed, data decomposition also occurs.

In a DFD, data flows are identified by unique names. These names are chosen so that they convey some meaning about what the data is. However, for specifying the precise structure of data flows, a data dictionary is often used. The associated data dictionary states precisely the structure of each data flow in the DFD. To define the data structure, a regular expression type notation is used. While specifying the structure of a data item, sequence or composition is represented by "+", selection by vertical bar "|" (means one OR the other), and repetition by "*".

ii) ER Diagrams

Entity relationship diagrams (ERDs) have been used for years for modeling the data aspects of a system. An ERD can be used to model the data in the system and how the data items relate to each other, but does not cover how the data is to be processed or how the data is actually manipulated and changed in the system. It is used often by database designers to represent the structure of the database and is a useful tool for analyzing software systems which employ databases. ER models form the logical database design and can be easily converted into initial table structure for a relational database.

ER diagrams have two main concepts and notations to representing them. These are entities and relationships. Entities are the main information holders or concepts in a system. Entities can be viewed as types which describe all elements of some type which have common properties. Entities are represented as boxes in an ER diagram, with a box representing all instances of the concept or type the entity is representing. An entity is essentially equivalent to a table in a database or a sheet in a spreadsheet, with each row representing an instance of this entity. Entities may have attributes, which are properties of the concept being represented. Attributes can be viewed as the columns of the database table and are represented as ellipses attached to the entity. To avoid cluttering, attributes are sometimes not shown in an ER diagram.

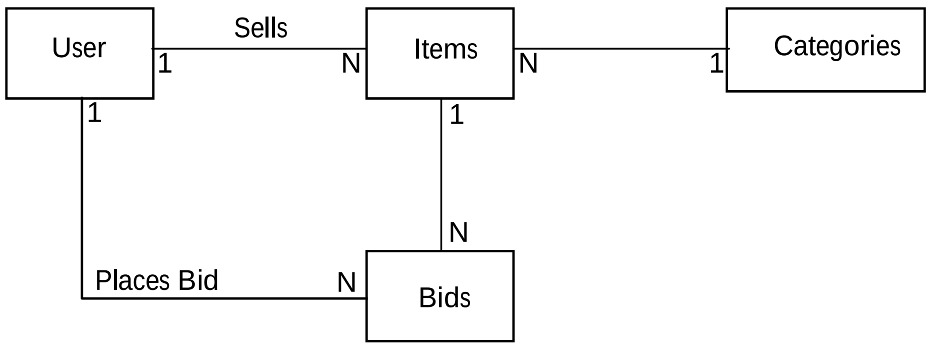
If all identities are identified and represented, we will have a set of labeled boxes in the diagram, each box representing some entity. Entities, of course, do not exist in isolation. They have relationships between them and that is the reason why they exist together in the same system.

Relationships between two entities are represented by a line connecting the boxes representing the entities. Having a line between two boxes means that each element of one entity is related to elements of the other entity, and vice versa. This relationship can also be given a name by labeling the line. In some notations, the name of the relationship is mentioned inside a diamond. Some examples of relationships are: studies-in (between students and college), worksfor (between employees and companies), and owner (between people and cars). Note that the relationships need not be between two distinct entities. There can be a relationship between elements of the same entity. For example, for an entity type Employee, there can be a relationship Supervisor, which is between elements of the entity Employee.

An ER diagram specifies some properties of the relationship also. In particular, it can specify if the relationship is optional (or necessary), and with how many elements an element of an entity is related to. This leads to many forms of relationships. The common ones are one-to-one (that one element of an entity is related to exactly one element of the other entity), one-to-many or many-to-one (that one element is related to many elements of the other entity), and many-to-many (that one element of entity A is related to many elements of entity B and one element of entity B is related to many elements of entity A). There are various notations to express the nature of relationships; a common one is to put "0", "1", or "M" on the two sides of the relationship line to represent the cardinality of the relationship. Thus, for a one-to-many relationship, a "1" will be put on one end and "N" will be put on the other end of the line.

Relationships reflect some properties of the problem domain. For example, a course has many students and a student is taking many courses, leading to many-to-many relationship between courses and students. But a student studies in exactly one college though the college has many students, leading to manyto- one relationship between students and college. A department in a college has exactly one head, and one person can be Head of only one department, leading to one-to-one relationship.

Let us draw the ER diagram for the university auction system, some use cases of which were discussed earlier. From the use cases described, we can easily identify some entities—users, categories, items, and bids. The relationships between them are also clear. A user can sell many items, but each item has only one seller, so there is a one-to-many relationship "Sell" between the user and items. Similarly, there is a one-to-many relationship between items and bids, between users and bids, and between categories and items.



From the ER diagram, it is easy to determine the initial logical structure of the tables. Each entity represents a table, and relationships determine what fields a table must have to support the relationship, in addition to having fields for each of the attributes. For example, from the ER diagram for the auction system, we can say that there will be four tables for users, categories, items, and bids. As user is related to item by one-to-many, the item table should have a user-ID field to uniquely identify the user who is selling the item. Similarly, the bid table must have a user-ID to identify the user who placed the bid, and an item-ID to identify the item for which the bid has been made.

As we can see, an ER diagram is complementary to methods like use cases. Whereas use cases focus on the nature of interaction and functionality, ER diagrams focus on the structure of the entities being used in the use cases. Due to their complementary nature, both use cases and ER diagrams can be used while analyzing the requirements of a system and both may be contained in an SRS.

2.7 Software Architecture

Q14. Explain about software architecture state its characteristics.

Ans : **(Imp.)**

Any complex system is composed of subsystems that interact under the control of system design such that the system provides the expected behavior. When designing such a system, therefore, the logical approach is to identify the subsystems that should compose the system, the interfaces of these subsystems, and the rules for interaction between the subsystems. This is what software architecture aims to do.

Software architecture is a relatively recent area. As the software systems increasingly become distributed and more complex, architecture becomes an important step in building the system. Due to a wide range of options now available for how a system may be configured and connected, carefully designing the architecture becomes very important. It is during the architecture design where choices like using some type of middleware, or some type of backend database, or some type of server, or some type of security component are made. Architecture is also the earliest place when properties like reliability and performance can be evaluated for the system, a capability that is increasingly becoming important.

Software Architecture also called High Level Software Design is the first design step after analyzing all requirements for software. The goal is to define a software structure which is able to fulfill the requirements. Also the non-functional requirements, such as scalability, portability and maintainability have to be considered in this step.

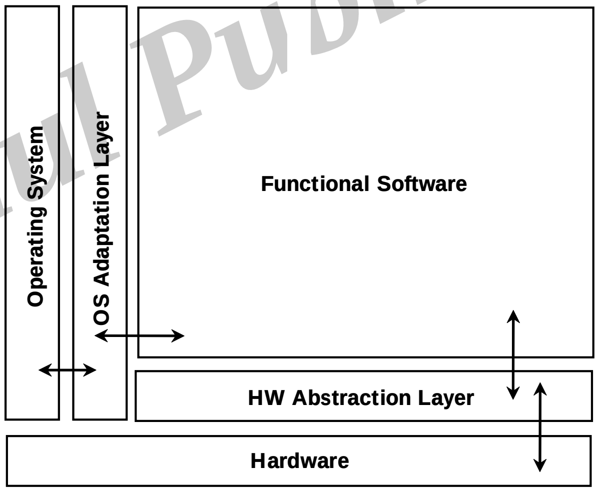
This first design step could be more or less independent of a programming language. However, the programming language has to be defined prior to defining the interfaces.

I) The Static Architecture

The first step in designing software is to define the static architecture. Simply speaking this is a very high level outline of the components and layers of software. Even if there are no requirements which explicitly ask for some of the below listed features, it is good design style to adhere to the following principles :

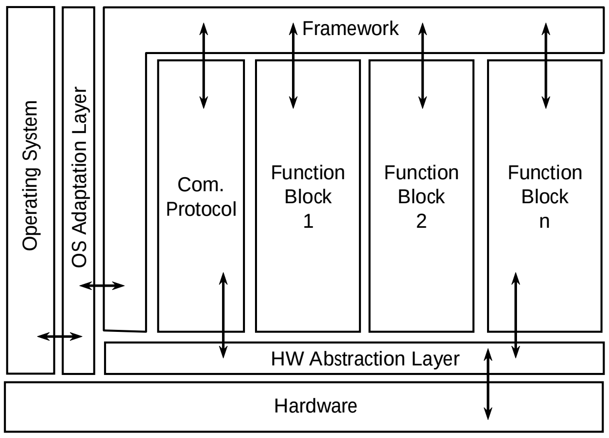
1. Define layers which make the functional part of the software independent of a hardware platform. There should be a hardware abstraction layer which encapsulates microcontroller specific code and features within the layer. All other layers have to be free of controller specific code. Another layer called "Physical Layer" should adapt the functional software to the specific signals. This is a data processing layer which filters signals and prepares them to be presented in physical units and defined resolutions at its interface.

2. If necessary, design a layer to adapt to a special operating system. Operating systems may offer services and semaphores, but never use them directly in your functional software. Define your own services and semaphores and go through a special layer to adapt them to the operating system services.



3. Design any additional layers inside your functional software as appropriate.

4. Design components inside your functional software. Depending on your requirements and future strategies it may be wise to e.g. design communication protocol components in a way that they can be easily removed and replaced by another protocol, to adapt to different platforms and systems. E.g. in the automotive industry the CAN bus is widely used for communication between the various electronic systems in a vehicle. However some customers require different communication systems as for example Flex Ray or any proprietary system. Your software can be designed in a way to modify the communication systems and protocols easily. Almost as easy as "plug and play", if the design is done properly.



5. Design an own framework which controls the calls and interactions of your software.

6. Consider organizational aspects in your architecture. Parts of the software may be developed by different departments or even outsourced to external companies. Your static architecture has to reflect this so that complete components or even layers can be assigned to a vendor. This has to be also reflected in the interface definition.

The Interfaces

The design of your interfaces is another element which adds to the stability, portability and maintainability of your software. The following things have to be observed :

1. Only use function calls as interfaces and refrain from using memory pools or any other globally shared elements as interface.

2. Make your interfaces as narrow as possible. Therefore use simple basic data types rather than complicated proprietary structures at the interfaces. It is sometimes amazing how simple interfaces can be if the functionality is distributed in a sensible way in appropriate components.

3. Preferably make your interfaces uni-directional. This means that input components provide interfaces used by the processing components and layers. Avoid bidirectional interaction between the same components.

4. Describe your interfaces clearly. Already in the architecture the kind of information, the data width, resolution and sign has to be defined. This is especially important if components are developed by different vendors.

II) The Dynamic Architecture

i) Operating Systems and Timing

Basically there are two categories of microcontroller systems. The first one is EVENT driven, as e.g. cell phones and other modern communication equipment.

The other kind of application is TIME driven. These microcontroller systems usually have the task to measure and evaluate signals and react on this information accordingly. This measuring activity means that signal sampling has to be performed. Additionally there may be activities like feedback current controls which have to be performed. Both activities imply by the underlying theories that sample rates have to be as exact as possible.

Both categories of systems are called REALTIME SYSTEMS, but they are like two different worlds!

The EVENT driven systems are comparatively simple. The are usually in an idle state until one of the defined events triggers a task or process, which is executed sequentially until it is finished and the system returns to the idle state. During the execution of such a task these systems usually do not react on other events. This "first comes first serves" principle can be seen in a cell phone, where incoming calls are ignored after you started to dial an outgoing call.

TIME driven system is much more complicated. Usually all possible inputs to the system have to be sampled and all outputs have to be served virtually simultaneously. This means that time slices have to be granted to the various activities and their duration has to be defined and limited to ensure the overall function of the system.

It would be too much to go into more details here. However there are some general rules which should be considered :

1. The CPU selection should be made according to the application. There are some CPUs which support time driven application in an optimized way. E.g. it is recommendable to have sufficient interrupt levels, CAPCOM units and I/O ports which can be accessed without delays for time driven applications.

   In recent years some well-known microcontrollers which originate in the event driven world were pushed into the time driven market. The ease of development and stability of the systems suffer from this. Although the attempt was made by the CPU manufacturer to cover for that by designing suitable peripheral units, the whole situation still looks like an odd patchwork rather than a sound design.

2. Operating systems can be event driven and non-preemptive for EVENT driven applications.
3. Operating systems should be time driven and preemptive for TIME driven systems.
4. Standard operating systems may fail you for high speed tasks, such as a 250 micro second cyclic tasks for feedback current controls. In this case you have to do this by timer interrupts outside of the operating system. Therefore have a close look at the OS from the shelf before you base your system on it.

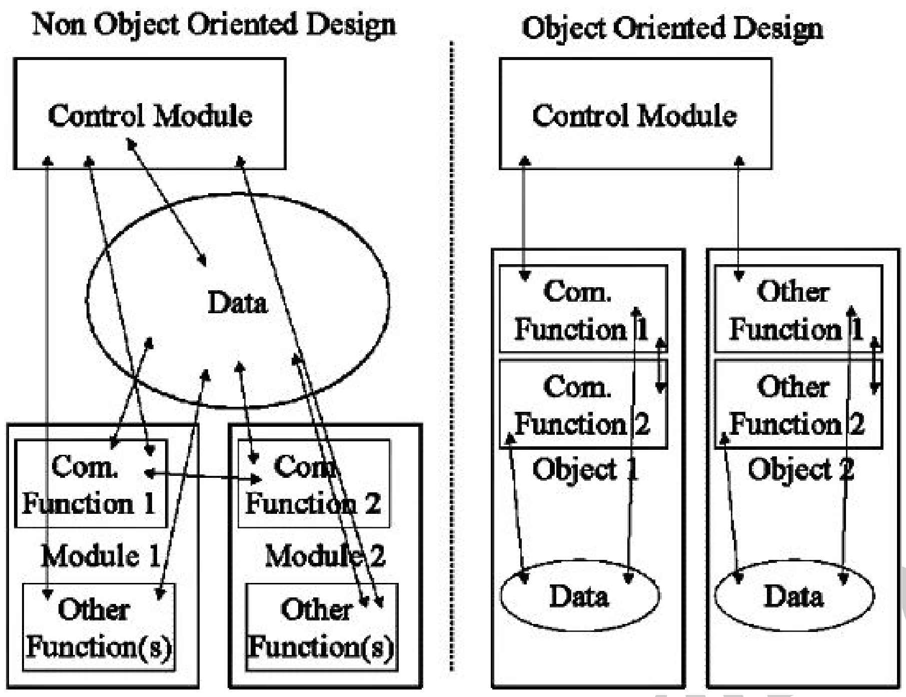
ii) Module Design

Module design which is also called "low level design" has to consider the programming language which shall be used for implementation. This will determine the kind of interfaces you can use and a number of other subjects. On these pages I want focus on module design for the C programming language and show some crucial principles for a successful design, which are the following:

iii) Object Orientation

Object orientation is nowadays usually associated with certain design methods like UML and programming languages like C++ or Java. However, the principles of object orientation were developed long before these methods and programming languages were invented. The first steps of object oriented design were done by using C. And indeed object orientation is a principle of design rather than a tool or method based feature. Some of these principles are:

1. Orientation of program components at the real physical world. This means that the division of a software package is done according to the real outside world of a system and according to main internal tasks of such a system.
2. Combining of all elements of a software (i.e. data, definitions and procedures) into an object. This means that everything that is needed to handle an element of the system is grouped and contained in one object. This is called encapsulation and will be further elaborated.
3. Access to the object's data and functions via a clearly defined narrow interface and hiding of elements which are not required for the outside world. Example:



iv) Encapsulation and Information Hiding

The principle of encapsulation goes hand in hand with "information hiding" and is part of the idea of object orientation. The principle is that only the data which are part of the interface of an object are visible to the outside world. Preferably these data are only available via function calls, rather than being presented as global variables. An encapsulated module design (related to C-programs) can be achieved by :

1. The use of local variables inside the functions as far as possible. I.e. avoid variables with validity across functions or even modules.
2. The use of C-function interfaces i.e. pass parameters and return parameters for data exchange, rather than global or static variables.
3. If values have to have a lifetime bejond one execution loop, use static variables rather than global variables.
4. Design your software with a synchronized control and data flow as outlined below.

The advantages of encapsulation are at hand :

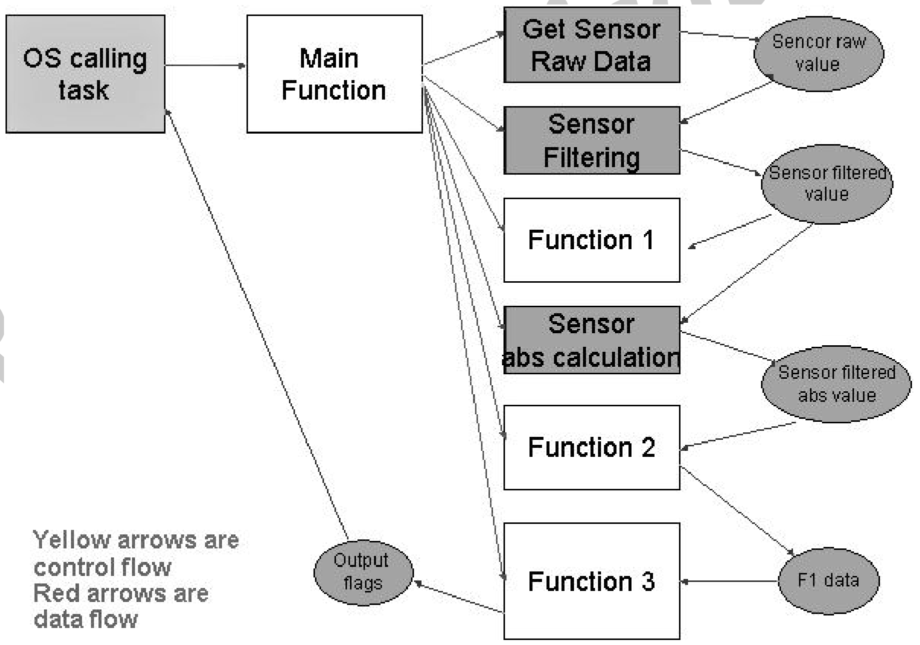
- No interference from other software parts. I.e. global variables are not available to the outside world and thus no other software portion can access them to modify them.
- No unexpected results for users of an object. Accessing global variables inside another object for reading may give you unexpected behaviour of these values. Unless you fully understand and consider the interiors of another object you never can be sure if it is behaving as expected. Clearly

defined interfaces can be tested and documented regarding their behavior. Some global value from in between a module may give you surprises.

- Good testability of the individual components.
- Good maintainability because of clearly defined behavior and interfaces.
- Reduced resource consumption! In experienced programmers don't believe this, but a couple of design improvements we did in the past clearly confirmed it again and again. Runtime is optimized in many cases by up to 40%. If the outlined design principles are followed consequently values are kept in CPU registers rather than in memory. The access to registers is usually much faster than a memory access. RAM is thus optimized by 30-40% as well, since variables are in registers or on the dynamic stack rather than on fixed locations in the memory. These advantages are much bigger than the tradeoff i.e. the execution time and stack consumption for calling a couple of additional functions.

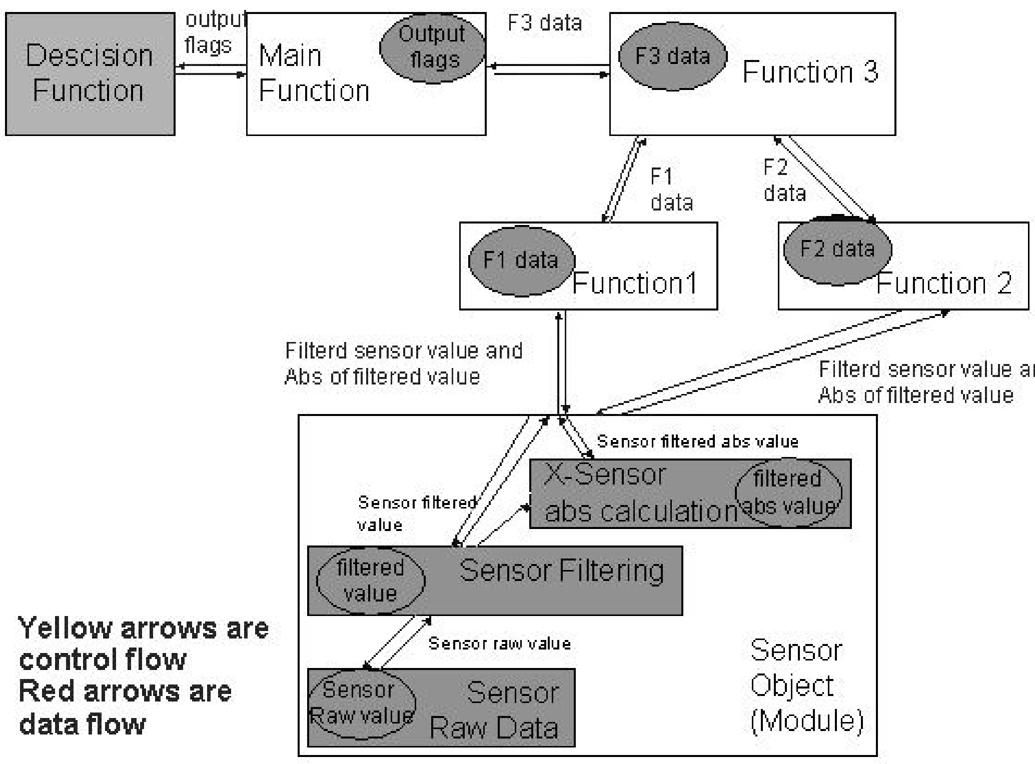
Synchronizing of Control and Data Flow

A picture tells more than thousand words in this case. Below you see a part of a typical design of microcontroller software. This is done the "classical" way i.e. in an "open" design using global variables and not using the interfaces of the C-functions.



This is the way a lot of microcontroller software is still done today. As you can see there are mainly global variables, and they are accessed form various functions. Can you imagine what happens if the sequence of the function calls is modified? This code is hard to maintain and even harder to test and may have a lot of surprises for you, up to field re-calls of your products. A surprise you certainly don't want to have!

In this picture you can see basically the same software following the principles of encapsulation and synchronized control and data flow :



How you can achieve this?

1. Follow the ideas of object orientation.
2. Make use of the encapsulation principles as outlined above.
3. Use the C-function interfaces and - use only C-functions as interfaces.
4. Make your system data flow driven. This means that e.g. if you expect an output of your system you should call the output generating function. This function needs data to make decisions and this data it gets by calling other functions in turn which e.g. do calculations and evaluation of intermediate results. The intermediate results are aquired by again calling other functions which aquire and prepare peripheral inputs e.g. sensors, etc.. Of course there are still a lot of things to consider even is a design as we just outlined, but in the end it will be stable and understandable, and even resource saving.

2.7.1 Role of Software Architecture Views

Q15. Explain the role of software architecture.

Ans :

Generally speaking, architecture of a system provides a very high level view of the parts of the system and how they are related to form the whole system. That is, architecture partitions the system in logical parts such that each part can be comprehended independently, and then describes the system in terms of these parts and the relationship between these parts.

Any complex system can be partitioned in many di erent ways, each providing a useful view and each having di erent types of logical parts. The same holds true for a software system—there is no unique structure of the system that can be described by its architecture; there are many possible structures.

Due to this possibility of having multiple structures, one of the most widely accepted definitions of software architecture is that the software architecture of a system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. This definition implies that for elements in architecture, we are only interested in those abstractions that specify those properties that other elements can assume to exist and that are needed to specify relationships. Details on how these properties are supported are not needed for architecture. This is an important capability that allows architecture descriptions to represent a complex system in a succinct form that is easily comprehended.

An architecture description of a system will therefore describe the di erent structures of the system. The next natural question is why should a team building a software system for some customer be interested in creating and documenting the structures of the proposed system. Some of the important uses that software architecture descriptions play are [6, 23, 54]:

1. Understanding and communication

An architecture description is primarily to communicate the architecture to its various stakeholders, which include the users who will use the system, the clients who commissioned the system, the builders who will build the system, and, of course, the architects. Through this description the stakeholders gain an understanding of some macro properties of the system and how the system intends to ful-fill the functional and quality requirements. As the description provides a common language between stakeholders, it also becomes the vehicle for negotiation and agreement among the stakeholders, who may have conflicting goals.

2. Reuse

The software engineering world has, for a long time, been working toward a discipline where software can be assembled from parts that are developed by di erent people and are available for others to use. If one wants to build a software product in which existing components may be reused, then architecture becomes the key point at which reuse at the highest level is decided. The architecture has to be chosen in a manner such that the components which have to be reused can fit properly and together with other components that may be developed. Architecture also facilitates reuse among products that are similar and building product families such that the common parts of these di erent but similar products can be reused. Architecture helps specify what is fixed and what is variable in these diferent products, and can help minimize the set of variable elements such that di erent products can share software parts to the maximum. Again, it is very hard to achieve this type of reuse at a detail level.

3. Construction and Evolution

As architecture partitions the system into parts, some architecture-provided partitioning can naturally be used for constructing the system, which also requires that the system be broken into parts such that di erent teams (or individuals) can separately work on dif erent parts. A suitable partitioning in the architecture can provide the project with the parts that need to be built to build the system. As, almost by definition, the parts specified in an architecture are relatively independent (the dependence between parts coming through their relationship), they can be built independently.

4. Analysis

It is highly desirable if some important properties about the behavior of the system can be determined before the system is actually built. This will allow the designers to consider alternatives and select the one that will best suit the needs. Many engineering disciplines use models to analyze design of a product for its cost, reliability, performance, etc. Architecture opens such possibilities for software also. It is possible (though the methods are not fully developed or standardized yet) to analyze or predict the properties of the system being built from its architecture. For example, the reliability or the performance of the system can be analyzed. Such an analysis can help determine whether the system will meet the quality and performance requirements, and if not, what needs to be done to meet the requirements.

For example, while building a website for shopping, it is possible to analyze the response time or throughput for a proposed architecture, given some assumptions about the request load and hardware. It can then be decided whether the performance is satisfactory or not, and if not, what new capabilities should be added (for example, a di erent architecture or a faster server for the back end) to improve it to a satisfactory level.

Not all of these uses may be significant in a project and which of these uses is pertinent to a project depends on the nature of the project. In some projects communication may be very important, but a detailed performance analysis may be unnecessary (because the system is too small or is meant for only a few users). In some other systems, performance analysis may be the primary use of architecture.

2.8 Architecture Styles

Q16. What are the different views of Software Architecture?

Ans : **(Imp.)**

There is a general view emerging that there is no unique architecture of a system. The definition that we have adopted (given above) also expresses this sentiment. Consequently, there is no one architecture drawing of the system. The situation is similar to that of civil construction, a discipline that is the original user of the concept of architecture and from where the concept of software architecture has been borrowed.

For a building, if you want to see the floor plan, you are shown one set of drawings. If you are an electrical engineer and want to see how the electricity distribution has been planned, you will be shown another set of drawings. And if you are interested in safety and firefighting, another set of drawings is used. These drawings are not independent of each other they are all about the same building. However, each drawing provides a different view of the building, a view that focuses on explaining one aspect of the building and tries to a good job at that, while not divulging much about the other aspects. And no one drawing can express all the different aspects such a drawing will be too complex to be of any use.

Similar is the situation with software architecture. In software, the different drawings are called views. A view represents the system as composed of some types of elements and relationships between them. Which elements are used by a view, depends on what the view wants to highlight. Different views expose different properties and attributes, thereby allowing the stakeholders and analysts to properly evaluate those attributes for the system. By focusing only on some aspects of the system, a view reduces the complexity that a reader has to deal with at a time, thereby aiding system understanding and analysis.

A view describes a structure of the system. We will use these two concepts views and structures interchangeably. We will also use the term architectural view to refer to a view. Many types of views have been proposed. Most of the proposed views generally belong to one of these three types:

i) Module

ii) Component and connector

iii) Allocation

i) Module

In a module view, the system is viewed as a collection of code units, each implementing some part of the system functionality. That is, the main elements in this view are modules. These views are code-based and do not explicitly represent any runtime structure of the system. Examples of modules are packages, a class, a procedure, a method, a collection of functions, and a collection of classes.

The relationships between these modules are also code-based and depend on how code of a module interacts with another module. Examples of relationships in this view are “is a part of” (i.e., module B is a part of module A), “uses or depends on” (a module A uses services of module B to perform its own functions and correctness of module A depends on correctness of module B), and “generalization or specialization” (a module B is a generalization of a module A).

ii) Component and Connector

In a component and connector (C&C) view, the system is viewed as a collection of runtime entities called components. That is, a component is a unit which has an identity in the executing system. Objects (not classes), a collection of objects, and a process are examples of components. While executing, components need to interact with others to support the system services. Connectors provide means for this interaction. Examples of connectors are pipes and sockets. Shared data can also act as a connector. If the components use some middleware to communicate and coordinate, then the middleware is a connector. Hence, the primary elements of this view are components and connectors.

iii) Allocation

An allocation view focuses on how the di erent software units are allocated to resources like the hardware, file systems, and people. That is, an allocation view specifies the relationship between software elements and elements of the environments in which the software system is executed. They expose structural properties like which processes run on which processor, and how the system files are organized on a file system.

An architecture description consists of views of different types, with each view exposing some structure of the system. Module views show how the software is structured as a set of implementation units, C&C views show how the software is structured as interacting runtime elements, and allocation views show how software relates to non-software structures. These three types of view of the same system form the architecture of the system.

Note that the different views are not unrelated. They all represent the same system. Hence, there are relationships between elements in one view and elements in another view. These relationships may be simple or may be complex. For example, the relationship between modules and components may be one to one in that one module implements one component. On the other hand, it may be quite complex with a module being used by multiple components, and a component using multiple modules. While creating the different views, the designers have to be aware of this relationship.

The next question is what are the standard views that should be expressed for describing the architecture of a system? To answer this question, the analogy with buildings may again help. If one is building a simple small house, then perhaps there is no need to have a separate view describing the emergency and the fire system. Similarly, if there is no air conditioning in the building, there need not be any view for that. On the other hand, an office building will perhaps require both of these views, in addition to other views describing plumbing, space, wiring, etc.

However, despite the fact that there are multiple drawings showing di erent views of a building, there is one view that predominates in construction - that of physical structure. This view forms the basis of other views in that other views cannot really be completed unless this view can be done. Other views may or may not be needed for constructing a building, depending on the nature of the project. Hence, in a sense, the view giving the building structure may be considered as the primary view in that it is almost always used, and other views rely on this view substantially. The view also captures perhaps the most important property to be analyzed in the early stages, namely, that of space organization.

The situation with software architecture is also somewhat similar. As we have said, depending on what properties are of interest, different views of the software architecture are needed. However, of these views, the C&C view has become the defacto primary view, one which is almost always prepared when an architecture is designed (some definitions even view architecture only in terms of C&C views). In this chapter, we will focus primarily on the C&C view. A note about relationship between architecture and design is in order.

As partitioning a system into smaller parts and composing the system from these parts is also a goal of design, a natural question is what is the difference between a design and architecture as both aim to achieve similar objectives and seem to fundamentally rely on the divide and conquer rule? First, it should be clear that architecture is a design in that it is in the solution domain and talks about the structure of the proposed system. Furthermore, an architecture view gives a high-level view of the system, relying on abstraction to convey the meaning - something which design also does. So, architecture is design. We can view architecture as a very high-level design, focusing only on main components, and the architecture activity as the first step in design. What we term as design is really about the modules that will eventually exist as code. That is, they are a more concrete representation of the implementation (though not yet an implementation).

Consequently, during design lower-level issues like the data structures, files, and sources of data, have to be addressed, while such issues are not generally significant at the architecture level. We also take the view that design can be considered as providing the module view of the architecture of the system.

The boundaries between architecture and high-level design are not fully clear. The way the field has evolved, we can say that the line between architecture and design is really up to the designer or the architect. At the architecture level, one needs to show only those parts that are needed to perform the desired evaluation. The internal structure of these parts is not important. On the other hand, during design, designing the structure of the parts that can lead to constructing them is one of the key tasks. However, which parts of the structure should be examined and revealed during architecture and which parts during design is a matter of choice. Generally speaking, details that are not needed to perform the types of analysis we wish to do at the architecture time are unnecessary and should be left for design to uncover.

2.9 Documenting Architecture Design

Q17. Explain in brief the need and reasons for documenting the software architecture.

Ans :

Documenting architecture is an important part of software development. Architecture must be documented in a good amount of detail and should be presented in an accessible form for many different stakeholders. A simple search shows lots of material. However, much of it tends to be pretty complex in nature. Much like there is not one definition around what an architect does, there is not one standard precise way of documenting architecture.

Let's look at some of the reasons around why we need to document architecture.

- ➢ Whiteboard designs are not persistent!
- ➢ Teams grow in size, and explaining the principles of architecture to a wider and wider audience gets difficult.
- ➢ The various decisions that drive the design could be forgotten and documenting them could help us get some rationale.
- ➢ People don't stay forever.
- ➢ Stakeholders have different concerns. This is a solid way to address those concerns ahead of time.
- ➢ This is an easy tool to visualize and plan for many different needs.
- ➢ Distributed systems and approaches like SOA have increased the overall design scope and complexity. No longer is the idea of a single system or a monolith accomplishing all the needs. Invariably, any modern system is build using multiple components and systems, and the interaction of these components together is what collectively accomplishes the business task. It becomes important to document their interaction.

These reasons pose several different questions:

- ➢ What should we document?
- ➢ How do we document?
- ➢ What approach or view should we use?
- ➢ How do we communicate clearly?

This article will attempt to answer some of these. However, the key point to remember is that there is not one right or wrong way; the answer depends on different things. It is important to consistently and periodically review and correct.

Another related question that could drive some of these concerns is the role of the architect. There are different views, ranging from the extremes of the architect being totally disconnected from the technology to architects who are totally immersed in technology working on bleeding edge research. The reality tends to be in the middle, where the architect is still expected to be technical and equipped to have skills to converse with non-technical stakeholders.

Documentation Misnomers

Let us first start with the misnomers that seem to be common :

- There are architects who specialize in particular diagramming tools or diagrams. They get so excited in the process of creating these artifacts that all their time and effort is spent focused on producing more and more diagrams. These are the architects whom sometimes I refer funnily as enterprise artists rather than enterprise architects. We have to remember that these diagrams are means to an end and not the end itself. There is nothing wrong in being artistic, but we are not here to produce art but architecture! It's important to balance out the artistic urge and its okay for some of the diagrams to be less perfect in terms of coloring, shapes, etc.
- Similarly, there are another set of Architects who use all sort of complex Architecture Artifacts that's available in the industry. They simply love complexity and would like to talk things in abstract and enjoy the kind of hallow the mystery produces.

"Any intelligent fool can make things bigger and more complex... It takes a touch of genius and a lot of courage to move in the opposite direction."

However, they have complete disregard for any feedback on whether what they produce is useful to any stakeholders.

It's important to remember that doing actual architecture and addressing concerns of the stakeholders is the primary role of an architect and the tools and documentation play more of a support role that helps us accomplish the primary role. So, picking tools and techniques that would allow the architect focus on his or her primary job is important. Any tool that is over-engineered or makes people spend more time is an absolute no for me.

Many architects are convinced that documenting views alone is sufficient. The industry has come to accept that software architecture cannot be described in a simple one-dimensional model. There are multiple goals and stakeholders and there are different views to address the goals and concerns. Architecture views are representations of the overall architecture that are meaningful to one or more stakeholders in the system. The architect chooses and develops a set of views that will enable the architecture to be communicated to and understood by all the stakeholders and enable them to verify that the system will address their concerns. Documenting more than views including things like architecture decisions, quality attributes, etc. are also equally important.

The views could be produced using different levels of notations. It is not a rule of thumb that they always need to be produced only using formal notations.

- **Informal notations:** custom conventions (i.e., general diagramming tools).
- **Semi-formal notations:** some element of semantics (i.e., UML).
- **Formal notations:** precise semantics (i.e., ADL such as ArchiMate).

More formal notations tend to take more time but allow us to produce less ambiguous artifacts.

So, what should a good architecture document contain. There are no definite set of rules and these are some of the suggestions that make sense to me.

Identify Views

Here is a definition of architecture view from OpenGroup :

"Architecture views are representations of the overall architecture that are meaningful to one or more stakeholders in the system. The architect chooses and develops a set of views that will enable the architecture to be communicated to, and understood by, all the stakeholders, and enable them to verify that the system will address their concerns."

Overall, we have come to a common understanding that describing architecture in a one-dimensional model is pretty complex to represent as well as understand. One-dimensional models are usually hard to understand and hard to maintain and end up being very poor in satisfying all of the different Stakeholder's concerns. Just like how traditional engineering produces different diagrams (Ex: Floor Plans, Electricity, Security), the

IT Architect is also expected to produce different views to address the concerns of the StakeHolders. The approach of producing many different Views allows us to represent complex systems in a manageable and maintainable fashion and satisfies many of the concerns of business and technical stakeholders.

There are different approaches that support this concept and are listed below. In general, the idea is to separate and provide views that support the following or more in detail. There are different nomenclatures available and I am using what I see to be the best fits.

- **Module views:** Internal working of a component.
- **Logical views:** How different modules work together to provide the business functionality.
- **Physical views:** How the components are deployed.
- **Behavior view:** The dynamic aspects of the system behavior and the variations. While the views are concerned with the structure of the system, these views focus on the behavior of the system for various internal and external stimuli and other behavior-driven inputs.

Example

- Krutchen 4+1 Architectural View Point.
- Siemen's Four View Model.
- Rozanski Woods.
- Archimate View Points.
- UML 2.o.

It is important to also document which Views address which of our stakeholder concerns to allow stakeholders to focus on what is important for them. Also there are different tooling available to satisfy the different Views, each satisfying a variety of concerns. It would be a good idea to identify and decide on them across the spectrum to ensure consistency.

Document Interfaces

The many different elements that make up the System interact with each other through interfaces. They are the key to building the Systems and are important for many stakeholders like Developers, Testers, operational folks etc. The interface documentation should inform what the consumers should know to interact with it in combination with many other elements. Usually the interfaces tend to be documented along with the Module Views as a part of the Views. The interface documentation is not just limited to the REST or SOAP API documentation. Though they could be classified as interface documentation and could be produced using some of the standard tooling available, they very well could be a Java or a C# Interface.

These are some of the suggestions around organizing interfaces.

- Name and version the interface.
- Provide details of operations and their semantics
- Provide what variances are available with respect to consuming the interface.
- Provide expected error conditions and error handling details.
- Provide performance or reliability numbers.
- Provide behavior diagrams like "sequence diagrams" in case the interaction is complex.

Document Key Decisions

A key responsibility of the Architect is to make variety of decisions weighing in the different concerns and tradeoffs. The approach of understanding software architecture in terms of a variety of architecture decisions is widely recognized. The decisions if documented could help in the long term around why certain design decisions were made and the actual thinking behind these decisions. A simple approach is to maintain a collection of "architecturally significant" decisions in a decision log. These are the decisions that impact many of the architecture concerns like NFRs, interfaces, Implementation Concerns, Design etc. The decision log could include things like

- **Context or background.** Explain what the issue is about and the options that are available.

- **Decision.** This includes the decision that is taken and the rationale.
- **Status.** This involves whether the decision is proposed or accepted. There are various lifecycle events for a decision.
- **Impact.** What is the impact of the decision? What do we gain or lose? What are the tradeoffs?
- **Stakeholders.** Parties who are impacted by decisions.

Document NFRs and Quality Attributes

This involves software that does not address its quality concerns and won't meet needs. It is important to document the different quality attributes and other NFRs.

- The design approach should consider the nonfunctional requirements and related cost.
- Various factors like like performance compliance, PCI and governance requirements, etc. that impact the design are visible to the different stakeholders.
- Different architectural components providing a service have quality attributes defined in their interface documentation. They, in turn, depend on all requirements.
- The NFRs should show how different nonfunctional requirements are satisfied.
- It helps different stakeholders like quality engineers and operational engineers to plan in advance various tasks like load testing, operational alerts, etc.

2.9.1 Building Blocks of Document Architecture

Q18. How would you build software architecture document?

Ans : **(Imp.)**

Architecture is the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution (IEEE 1471).

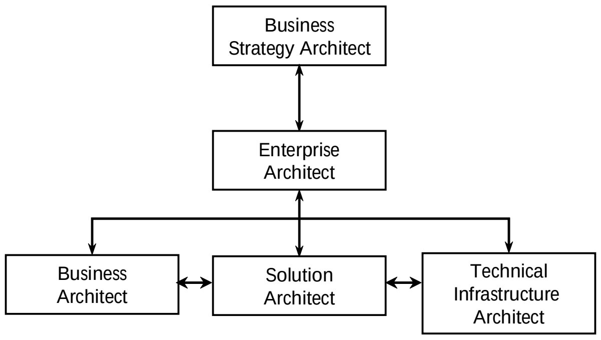
The definition suggested by IEEE (above) refers to a solution architect and/or software architect. However, as Microsoft suggests there are other kinds of architects such as a Business Strategy Architect. There are basically six types of Architects:

Business Strategy Architect

The purpose of this role is to change business focus and define the enterprise's to-be status. This role, he says, is about the long view and about forecasting.

- **Business Architect:** The mission of business architects is to improve the functionality of the business. Their job isn't to architect software but to architect the business itself and the way it is run.
- **Solution Architect:** Solution architect is a relatively new term, and it should refer also to an equally new concept. Sometimes, however, it doesn't; it tends to be used as a synonym for application architect.
- **Software Architect:** Software architecture is about architecting software meant to support, automate, or even totally change the business and the business architecture.
- **Infrastructure Architect:** The technical infrastructure exists for deployment of the solutions of the solution architect, which means that the solution architect and the technical infrastructure architect should work together to ensure safe and productive deployment and operation of the system
- **Enterprise Architect :** Enterprise Architecture is the practice of applying a comprehensive and rigorous method for describing a current and/or future structure and behaviour for an organization's processes, information systems, personnel and organizational subunits, so that they align with the organization's core goals and strategic

direction. Although often associated strictly with information technology, it relates more broadly to the practice of business optimization in that it addresses business architecture, performance management, and process architecture as well (Wikipedia).


Business Strategy Architect
Enterprise Architect
Business Architect
Solution Architect
Technical Infrastructure Architect

Solution Architect

As we are techies let's focus on Solution Architect role :

It tends to be used as a synonym for application architect. In an application-centric world, each application is created to solve a specific business problem, or a specific set of business problems. All parts of the application are tightly knit together, each integral part being owned by the application. An application architect designs the structure of the entire application, a job that's rather technical in nature. Typically, the application architect doesn't create, or even help create, the application requirements; other people, often called business analysts, less technically and more business-oriented than the typical application architect, do that.

So if you are asked to get on board and architecture a system based on a whole bunch of requirements, you are very likely to be asked to do solution architecture.

How to do that?

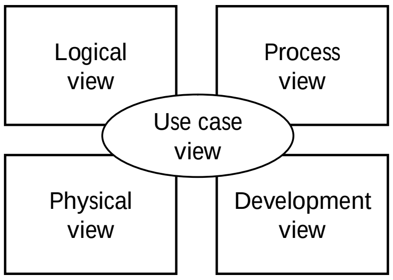
A while back a person who does not have a technical background, but he has money so he is the boss, was lecturing that in an ideal world no team member has to talk to other team members. At that time I was thinking that in my ideal world, which is very close to the Agile world, everybody can (or should) speak to everybody else. This points out that how you architecture a system is strongly tight to your methodology. It does not really make a big difference that which methodology you follow as long as you stick to the correct concepts. Likewise, he was saying that the Software Architecture Document is part of the BRD (Business Requirement Document) as if it was technical a business person (e.g. the stake holders) would not understand it. And I was thinking to me that: mate! There are different views being analyzed in a SAD. Some of them are technical, some of them are not.

What the above story points out to me is that solution architecture is the art of mapping the business stuff to technical stuff, or in the other words, it's actually speaking about technical things in a language which is understandable to business people.

A very good way to do this is to putting yourself in the stakeholders' shoes. There are several types of stakeholders in each project who have their own views and their own concerns. This is the biggest difference between the design and the architecture. A designer thinks very technically while an architect can think broadly and can look at a problem from different views. Designers usually make a huge mistake, which happens a lot in Australia: They put everything in one document. Where I am doing a solution architecture job now, I was given a 21-mega-byte MS Word document which included everything, from requirements to detailed class and database design.

Such a document is very unlikely to be understandable by the stakeholders and very hard to use by developers. I reckon that this happens because firstly designers don't consider the separation of stake holders and developers concerns. Second, because it's easier to write down everything in a document. But I have to say that this is wrong as SAD and design document (e.g. TSD) are built for different purposes and for different audiences (and in different phases if you are following a phase-based methodology such as RUP). If you put everything in a document, it's like you are cooking dinner and you put the ingredients along with the utensils in a pot and boil them!!

A very good approach for looking at the problem from the stakeholder's point of view is the 4+1 approach. At this model, scenarios (or Use Cases) are the base and we look at them from a logical view (what are the building blocks of the system), Process view (processes such as asynchronous operations), Development (aka Implementation) view and Physical (aka Deployment) view. There are also optional views such as Data View that you can use if you need to. Some of the views are technical and some of them are not, however they must match and there must be a consistency in the architecture so that technical views can cover business views (e.g. demonstration of a business process with a UML Activity Diagram and/or State Diagram).



I believe that each software project is like a spectrum that each stakeholder sees a limited part of it. The role of an architect is to see the entire spectrum. A good approach to do so (that I use a lot) is to include a business vision (this might not be a good term) in your SAD. It can be a billeted list, a diagram or both, which shows what the application looks like from a business perspective. Label each part of the business vision with a letter or a number. Then add an architectural overview and then map it to the items of business vision indicating that which part of the architecture is meant to address which part of the business vision.

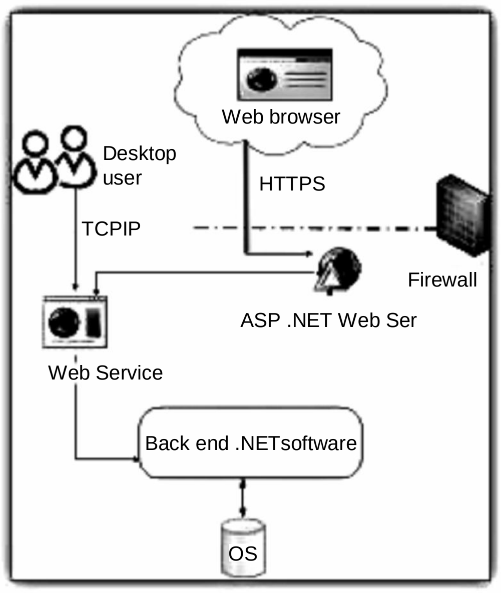
Q19. Explain the concept of Software Architecture Document (SAD).

Ans :

There are a whole bunch of SAD templates on the internet, such as the template offered by RUP. However the following items seem to be necessary for each architecture document:

- Introduction. This can include Purpose, Glossary, Background of the project, Assumptions, References etc. I personally suggest that you explain that what kind of methodology you are following? This will avoid lots of debates, I promise!
- It is very important to clear the scope of the document. Without a clear scope not only you will never know that when you are finished, you won't be able to convince the stakeholder that the architecture is comprehensive enough and addresses all their needs.
- Architectural goals and constraints: This can include the goals, as well as your business and architectural visions. Also explain the constraints (e.g. if the business has decided to develop the software system with Microsoft .NET, it is a constraint). I would suggest that you mention the components (or modules) of the system when you mention your architectural vision. For example say that it will include Identity Management, Reporting etc. And explain what your strategy to address them is. As this section is intended to help the business people to understand your architecture, try to include clear and well-organized diagrams.
- A very important item that you want to mention is the architectural principles that you are following. This is even more important when the client organization maintains a set of architectural principles.
  - **Quality of service requirements:** Quality of service requirements address the quality attributes of the system, such as performance, scalability, security etc. These items must not be mentioned in a technical language and must not contain any details (e.g. the use of Microsoft Enterprise Library 5).
  - **Use Case View:** Views basically come from 4+1 model so if you follow a different model you might not have it. However, it is very important that you detect key scenarios (or Use Cases) and mention them in a high-level. Again, diagrams, such as Use Case Diagram, help.
  - **Logical View :** Logical view demonstrates the logical decomposition of the system, such as packages the build it. It will help the business people and the designers to understand the system better.
  - **Process View:** Use activity diagrams as well as state diagrams (if necessary) to explain the key processes of the system (e.g. the process of approving a leave request).

- **Deployment View:** Deployment view demonstrates that how the system will work in a real production environment. I suggest that you put 2 types of diagrams: one (normal) human understandable diagram, such a Visio Diagram that shows the network, firewall, application server, database, etc. Also a UML deployment diagram that demonstrates the nodes and dependencies. This will again helps the business and technical people have same understanding of the physical structure of the system.
- **Implementation View:** This part is the most interesting section of the techies. I like to include the implementation options (e.g. Java and .NET) and provide a list of pros and cons for each of them. Again, technical pros and cons don't make much sense to business people. They are mostly interested in Cost of Ownership and availability of the resources and so on. If you suggest a technology or if it has already been selected, list the products and services that are needed on a production environment (e.g. IIS 7, SQL Server 2008). Also it'll be good to include a very high-level diagram of the system.



Also I like to explain the architectural patterns that I'm going to use. If you are including this section in the Implementation View, explain them enough so that a business person can quite understand what that pattern is for. For instance if you are using Lazy Loading patter, explain that what problem does it solve and why you are using it.

Needless to say that you have to also decide which kind of Architecture style you are suggesting, such as 3-Tier and N-Tier, Client-Server etc. Once you have declared that, explain the components of the system (Layers, Tiers and their relationships) by diagrams.

This part also must include your implementation strategy for addressing the Quality of Service Requirements, such as how will you address scaling out.

- **Data View:** If the application is data centric, explain the overall solution of data management (never put a database design in this part), your backup and restore strategy as well as disaster recovery strategy.
- **Be iterative:** It is suggested that the architecture (and in result the Software Architecture Document) be developed through two or more iterations. It's impossible to build a comprehensive architecture document in one iteration as not only Architecture has an impact on the requirements, but also architecture begins in an early stage and many of the scenarios are likely to change.

How to prove that ?

Now that after doing lots of endeavor you have prepared your SAD, how will you prove it to the stakeholders? I assume that many of business people do not have any idea about the content and structure of an SAD and the amount of information that you must include in it.

A good approach is to prepare a presentation about the mission of the system, scope, goals, visions and your approach. Invite the stakeholders to a meeting and present the architecture to them and explain that how the architecture covers their business needs. If they are not satisfied, your architecture is very likely to be incomplete.

UNIT III

Planning a Software Project: Effort Estimation, Project Schedule and staffing, Quality Planning, Risk Management Planning, Project Monitoring Plan, Detailed Scheduling.

Design: Design concepts, Function oriented Design, Object Oriented Design, Detailed Design, Verification, Metrics.

3.1 Planning a Software Project

Q1. What is a Software Project?

(OR)

Explain the planning a software project.

Ans :

Planning is the most important project management activity. It has two basic objectives-establish reasonable cost, schedule, and quality goals for the project, and to draw out a plan to deliver the project goals. A project succeeds if it meets its cost, schedule, and quality goals. Without the project goals being defined, it is not possible to even declare if a project has succeeded. And without detailed planning, no real monitoring or controlling of the project is possible.

Often projects are rushed toward implementation with not enough effort spent on planning. No amount of technical effort later can compensate for lack of careful planning. Lack of proper planning is a sure ticket to failure for a large software project. For this reason, we treat project planning as an independent chapter. Note that we also cover the monitoring phase of the project management process as part of planning, as how the project is to be monitored is also a part of the planning phase.

The inputs to the planning activity are the requirements specification and maybe the architecture description. A very detailed requirements document is not essential for planning, but for a good plan all the important requirements must be known, and it is highly desirable that key architecture decisions have been taken.

There are generally two main outputs of the planning activity: the overall project management plan document that establishes the project goals on the cost, schedule, and quality fronts, and defines the plans for managing risk, monitoring the project, etc.; and the detailed plan, often referred to as the detailed project schedule, specifying the tasks that need to be performed to meet the goals, the resources who will perform them, and their schedule. The overall plan guides the development of the detailed plan, which then becomes the main guiding document during project execution for project monitoring.

- How to estimate effort and schedule for the project to establish project goals and milestones and determine the team size needed for executing the project.
- How to establish quality goals for the project and prepare a quality plan.
- How to identify high-priority risks that can threaten the success of the project, and plan for their mitigation.
- How to plan for monitoring a project using measurements to check if a project is progressing as per the plan.
- How to develop a detailed task schedule from the overall estimates and other planning tasks done such that, if followed, the overall goals of the project will be met.

3.2 EFFORT ESTIMATION

Q2. Explain about Effort and Schedule estimates of Software Projects.

Ans : **(Imp.)**

For a software development project, overall effort and schedule estimates are essential prerequisites for planning the project. These estimates are needed before development is initiated, as they establish the cost and schedule goals of the project.

A more practical use of these estimates is in bidding for software projects, where cost and schedule estimates must be given to a potential client for the development contract. (As the bulk of the cost of software development is due to the human effort, cost can easily be determined from effort by using a suitable person-month cost value.) Effort and schedule estimates are also required for determining the staffing level for a project during different phases, for the detailed plan, and for project monitoring.

The accuracy with which effort can be estimated clearly depends on the level of information available about the project. The more detailed the information, the more accurate the estimation can be. Of course, even with all the information available, the accuracy of the estimates will depend on the effectiveness and accuracy of the estimation procedures or models employed and the process. If from the requirements specifications, the estimation approach can produce estimates that are within 20% of the actual effort about two-thirds of the time, then the approach can be considered good. Here we discuss two commonly used approaches.

I) Top-Down Estimation Approach

Although the effort for a project is a function of many parameters, it is generally agreed that the primary factor that controls the effort is the size of the project. That is, the larger the project, the greater is the effort requirement. The topdown approach utilizes this and considers effort as a function of project size. Note that to use this approach, we need to first determine the nature of the function, and then to apply the function, we need to estimate the size of the project for which effort is to be estimated.

If past productivity on similar projects is known, then it can be used as the estimation function to determine effort from the size. If productivity is P KLOC/PM, then the effort estimate for the project will be SIZE/P person months. Note that as productivity itself depends on the size of the project (larger projects often have lower productivity), this approach can work only if the size and type of the project are similar to the set of projects from which the productivity P was obtained (and that in the new project a similar productivity can be obtained by following a process similar to what was used in earlier projects).

A more general function for determining effort from size that is commonly used is of the form:

EFFORT = a * SIZE^b,

where a and b are constants, and project size is generally in KLOC (size could also be in another size measure called function points which can be determined from requirements). Values for these constants for an organization are determined through regression analysis, which is applied to data about the projects that have been performed in the past. For example, Watson and Felix [81] analyzed the data of more than 60 projects done at IBM Federal Systems Division, ranging from 4000 to 467,000 lines of delivered source code, and found that if the SIZE estimate is in thousands of delivered lines of code (KLOC), the total effort, E, in person-months (PM) can be given by the equation E = 5.2(SIZE).91. In the COnstructive COst MOdel (COCOMO), for the initial estimate (also called nominal estimate) the equation for an organic project is

E = 3.9(SIZE)$^.$.

Though size is the primary factor affecting cost, other factors also have some effect. In the COCOMO model, after determining the initial estimate, some other factors are incorporated for obtaining the final estimate. To do this, COCOMO uses a set of 15 different attributes of a project called cost driver attributes. Examples of the attributes are required software reliability, product complexity, analyst capability, application experience, use of modern tools, and required development schedule. Each cost driver has a rating scale, and for each rating, a multiplying factor is provided. For example, for the reliability, the rating scale is very low, low, nominal, high, and very high; the multiplying factors for these ratings are .75, .88, 1.00, 1.15, and 1.40, respectively. So, if the reliability requirement for the project is judged to be low, then the multiplying factor is .75, while if it is judged to be very high, the factor is 1.40. The attributes and their multiplying factors for different ratings.

Cost Drivers	**Rating**				
	Very Low	**Low**	**Nominal**	**High**	**Very High**
Product Attributes					
RELY, required reliability	.75	.88	1.00	1.15	1.40
DATA, database size		.94	1.00	1.08	1.16
CPLX, product complexity	.70	.85	1.00	1.15	1.30
Computer Attributes					
TIME, execution time constraint			1.00	1.11	1.30
STOR, main storage constraint			1.00	1.06	1.21
VITR, virtual machine volatility		.87	1.00	1.15	1.30
TURN, computer turnaround time		.87	1.00	1.07	1.15
Personnel Attributes					
ACAP, analyst capability	1.46	1.19	1.00	.86	.74
AEXP, application exp.	1.29	1.13	1.00	.91	.82
PCAP, programmer capability	1.42	1.17	1.00	.86	.70
VEXP, virtual machine exp.	1.21	1.10	1.00	.90	
LEXP, prog. language exp.	1.14	1.07	1.00	.95	
Project Attributes					
MODP, modern prog. practices	1.24	1.10	1.00	.91	.82
TOOL, use of SW tools	1.24	1.10	1.00	.91	.83
SCHED, development schedule	1.23	1.08	1.00	1.04	1.10

The multiplying factors for all 15 cost drivers are multiplied to get the effort adjustment factor (EAF). The final effort estimate, E, is obtained by multiplying the initial estimate by the EAF. In other words, adjustment is made to the size-based estimate using the rating for these 15 different factors.

As an example, consider a system being built for supporting auctions in a university (some of the use cases of this were discussed in the previous chapter). From the use cases and other requirements, it is decided that the system will comprise a few different modules. The modules and their expected sizes are:

Login 200 LOC
Payment 200 LOC
Administrator interface 600 LOC
0Seller functions 200 LOC
Buyer functions 500 LOC
View and bookkeeping 300 LOC
TOTAL 2000 LOC

The total size of this software is estimated to be 2 KLOC. If we want to use COCOMO for estimation, we should estimate the value of the different cost drivers. Suppose we expect that the complexity of the system is high, the programmer capability is low, and the application experience of the team is low. All other factors have a nominal rating. From these, the effort adjustment factor (EAF) is

EAF = 1.15 * 1.17 * 1.13 = 1.52.

The initial effort estimate for the project is obtained from the relevant equations. We have

$Ei = 3.9 * 2^{.91} = 7.3$ PM.

Using the EAF, the adjusted effort estimate is

E = 1.52 * 7.3 = 11.1 PM.

From the overall estimate, estimates of the effort required for the different phases in the projects can also be determined. This is generally done by using an effort distribution among phases. The percentage of total effort spent in a phase varies with the type and size of the project, and can be obtained from data of similar projects in the past.

It should be noted that to use the top-down approach for estimation, even if we have a suitable function, we need to have an estimate of the project size. In other words, we have replaced the problem of effort estimation by size estimation. One may then ask, why not directly do effort estimation rather than size estimation? The answer is that size estimation is often easier than direct effort estimation. This is mainly due to the fact that the system size can be estimated from the sizes of its components (which is often easier to do) by adding the size estimates of all the components. Similar property does not hold for effort estimation, as effort for developing a system is not the sum of effort for developing the components (as additional effort is needed for integration and other such activities when building a system from developed components).

Clearly for top-down estimation to work well, it is important that good estimates for the size of the software be obtained. There is no known "simple" method for estimating the size accurately. When estimating software size, the best way may be to get as much detail as possible about the software to be developed and to be aware of our biases when estimating the size of the various components. By obtaining details and using them for size estimation, the estimates are likely to be closer to the actual size of the final software.

II) Bottom-Up Estimation Approach

A somewhat different approach for effort estimation is the bottom-up approach. In this approach, the project is first divided into tasks and then estimates for the different tasks of the project are obtained. From the estimates of the different tasks, the overall estimate is determined. That is, the overall estimate of the project is derived from the estimates of its parts. This type of approach is also called activity-based estimation. Essentially, in this approach the size and complexity of the project is captured in the set of tasks the project has to perform.

The bottom-up approach lends itself to direct estimation of effort; once the project is partitioned into smaller tasks, it is possible to directly estimate the effort required for them, especially if tasks are relatively small. One difficulty in this approach is that to get the overall estimate, all the tasks have to be enumerated. A risk of bottom-up methods is that one may omit some activities. Also, directly estimating the effort for some overhead tasks, such as project management, that span the project can be difficult.

If architecture of the system to be built has been developed and if past information about how effort is distributed over different phases is known, then the bottom-up approach need not completely list all the tasks, and a less tedious approach is possible. Here we describe one such approach used in a commercial organization.

In this approach, the major programs (or units or modules) in the software being built are first determined. Each program unit is then classified as simple, medium, or complex based on certain criteria. For each classification unit, an average effort for coding (and unit testing) is decided. This average coding effort can be based on past data from a similar project, from some guidelines, or on experience of people.

Once the number of units in the three categories of complexity is known and the estimated coding effort for each program is selected, the total coding effort for the project is known. From the total coding effort, the effort required for the other phases and activities in the project is determined as a percentage of coding effort. For this, from information about the past performance of the process, the likely distribution of effort in different phases of this project is determined. This distribution is then used to determine the effort for other phases and activities from the effort estimate of coding. From these estimates, the total effort for the project is obtained.

This approach lends itself to a judicious mixture of experience and data. If suitable past data are not available (for example, if launching a new type of project), one can estimate the coding effort using experience once the nature of the different types of units is specified. With this estimate, we can obtain the estimate for other activities by working with some reasonable or standard effort distribution. This strategy can easily account for activities that are sometimes difficult to enumerate early but do consume effort by budgeting effort for an "other" or "miscellaneous" category.

The procedure for estimation can be summarized as the following sequence of steps :

1. Identify modules in the system and classify them as simple, medium, or complex.
2. Determine the average coding effort for simple/medium/complex modules.
3. Gets the total coding effort using the coding effort of different types of modules and the counts for them?
4. Using the effort distribution for similar projects, estimate the effort for other tasks and the total effort.
5. Refine the estimates based on project-specific factors.

This procedure uses a judicious mixture of past data (in the form of distribution of effort) and experience of the programmers. This approach is also simple and similar to how many of us plan any project. For this reason, for small projects, many people find this approach natural and comfortable.

Note that this method of classifying programs into a few categories and using an average coding effort for each category is used only for effort estimation. In detailed scheduling, when a project manager assigns each unit to a member of the team for coding and budgets time for the activity, characteristics of the unit are taken into account to give more or less time than the average.

3.3 Project Schedule and Staffing

Q3. What is Staffing? Explain Project Schedule.

Ans : **(Imp.)**

After establishing a goal on the effort front, we need to establish the goal for delivery schedule. With the effort estimate (in person-months), it may be tempting to pick any project duration based on convenience and then fix a suitable team size to ensure that the total effort matches the estimate. However, as is well known now, person and months are not fully interchangeable in a software project. Person and months can be interchanged arbitrarily only if all the tasks in the project can be done in parallel, and no communication is needed between people performing the tasks. This is not true for software projects—there are dependencies between tasks (e.g., testing can only be done after coding is done), and a person performing some task in a project needs to communicate with others performing other tasks. As Brooks has pointed out man and months are interchangeable only for activities that require no communication among men, like sowing wheat or reaping cotton. This is not even approximately true of software ...."

However, for a project with some estimated effort, multiple schedules (or project duration) are indeed possible. For example, for a project whose effort estimate is 56 person-months, a total schedule of 8 months is possible with 7 people. A schedule of 7 months with 8 people is also possible, as is a schedule of approximately 9 months with 6 people. (But a schedule of 1 month with 56 people is not possible. Similarly, no one would execute the project in 28 months with 2 people.) In other words, once the effort is fixed, there is some flexibility in setting the schedule by appropriately staffing the project, but this flexibility is not unlimited. Empirical data also suggests that no simple equation between effort and schedule fits well.

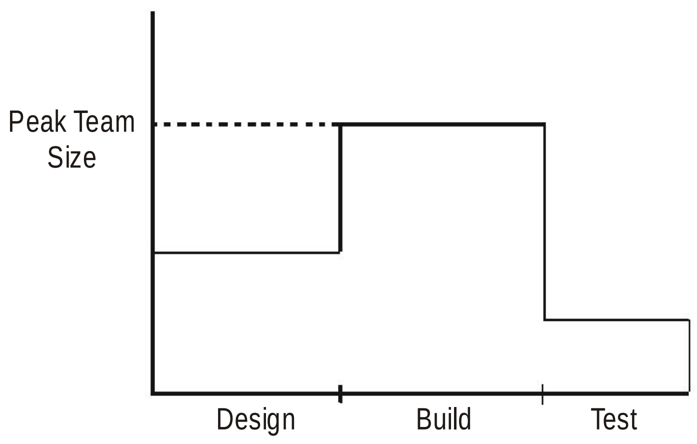
The objective is to fix a reasonable schedule that can be achieved (if suitable number of resources is assigned). One method to determine the overall schedule is to determine it as a function of effort. Such function can be determined from data from completed projects using statistical techniques like fitting a regression curve through the scatter plot obtained by plotting the effort and schedule of past projects. This curve is generally nonlinear because the schedule does not grow linearly with effort. Many models follow this approach [2, 12]. The IBM Federal Systems Division found that the total duration, M, in calendar months can be estimated by $M=4.1E^{.38}$. In COCOMO, the equation for schedule for an organic type of software is $M = 2.5E^{.38}$. As schedule is not a function solely of effort, the schedule determined in this manner is essentially a guideline.

Another method for checking a schedule for medium-sized projects is the rule of thumb called the square root check [58]. This check suggests that the proposed schedule can be around the square root of the total effort in person months. This schedule can be met if suitable resources are assigned to the project. For example, if the effort estimate is 50 person-months, a schedule of about 7 to 8 months will be suitable. From this macro estimate of schedule, we can determine the schedule for the major milestones in the project.

To determine the milestones, we must first understand the manpower ramp-up that usually takes place in a project. The number of people that can be gainfully utilized in a software project tends to follow the Rayleigh curve [71, 72]. That is, in the beginning and the end, few people are needed on the project; the peak team size (PTS) is needed somewhere near the middle of the project; and again fewer people are needed after that. This occurs because only a few people are needed and can be used in the initial phases of requirements analysis and design. The human resources requirement peaks during coding and unit testing, and during system testing and integration, again fewer people are required.

Often, the staffing level is not changed continuously in a project and approximations of the Rayleigh curve are used: assigning a few people at the start, having the peak team during the coding phase, and then leaving a few people for integration and system testing. If we consider design and analysis, build, and test as three major phases, the manpower ramp-up in projects typically resembles the function. For ease of scheduling, particularly for smaller projects, often the required people are assigned together around the start of the project. This approach can lead to some people being unoccupied at the start and toward the end. This slack time is often used for supporting project activities like training and documentation.

Given the effort estimate for a phase, we can determine the duration of the phase if we know the manpower ramp-up. For these three major phases, the percentage of the schedule consumed in the build phase is smaller than the percentage of the effort consumed because this phase involves more people. Similarly, the percentage of the schedule consumed in the design and testing phases exceeds their effort percentages. The exact schedule depends on the planned manpower ramp-up, and how many resources can be used effectively in a phase on that project. Generally speaking, design requires about a quarter of the schedule, build consumes about half, and integration and system testing consume the remaining quarter. COCOMO gives 19% for design, 62% for programming, and 18% for integration.



3.4 Quality Planning

Q4. Define Quality Planning. Write different Software Quality Factors.

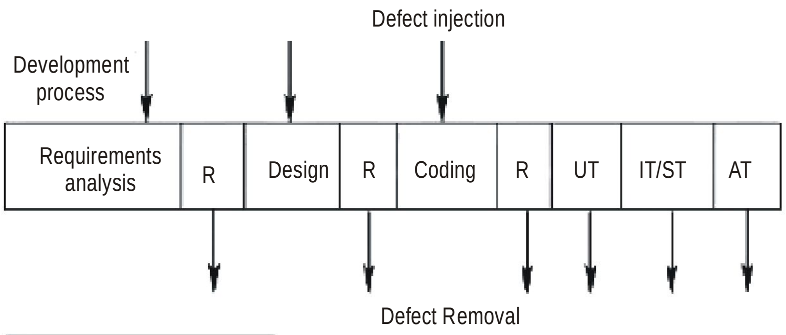
Ans :

For quality, even if we set the goal in terms of expected delivered defect density, it is not easy to plan for achieving this goal or for checking if a plan can meet these goals. Hence, often, quality goals are specified in terms of acceptance criteria— the delivered software should finally work for all the situations and test cases in the acceptance criteria. Further, there may even be an acceptance criterion on the number of defects that can be found during the acceptance testing. For example, no more than n defects are uncovered by acceptance testing.

The quality plan is the set of quality-related activities that a project plans to do to achieve the quality goal. To plan for quality, let us first understand the defect injection and removal cycle, as it is defects that determine the quality of the final delivered software.

Software development is a highly people-oriented activity and hence it is error-prone. In a software project, we start with no defects (there is no software to contain defects). Defects are injected into the software being built during the different phases in the project. That is, during the transformation from user needs to software to satisfy those needs, defects are injected in the transformation activities undertaken. These injection stages are primarily the requirements specification, the high-level design, the detailed design, and coding.

To ensure that high-quality software is delivered, these defects are removed through the quality control (QC) activities. The QC activities for defect removal include requirements reviews, design reviews, code reviews, unit testing, integration testing, system testing, acceptance testing, etc. Figure 4.2 shows the process of defect injection and removal.



R - Removal
UT - Unit Testing
IT - Integration Testing
ST - System Testing
AT - Acceptance Testing

As the final goal is to deliver software with low defect density, ensuring quality revolves around two main themes: reduce the defects being injected, and increase the defects being removed. The first is often done through standards, methodologies, following of good processes, etc., which help reduce the chances of errors by the project personnel. (There are specific techniques for defect prevention also.) The quality plan therefore focuses mostly on planning suitable quality control tasks for removing defects.

Reviews and testing are two most common QC activities utilized in a project. Whereas reviews are structured, human-oriented processes, testing is the process of executing software (or parts of it) in an attempt to identify defects. The most common approach for quality planning in a project is to specify the QC activities to be performed in the project, and have suitable guidelines for performing each of the QC tasks, such that the chances of meeting the quality goals are high. During project execution, these activities are carried out in accordance with the defined procedures.

When this approach is used for ensuring quality, making quantitative claims can be quite hard. For quantitative assessment of the quality processes, metrics-based analysis is necessary. That, however, is an advanced topic, beyond the scope of this book (and indeed many organizations). Hence, for ensuring quality, the reliance is primarily on applying suitable QC techniques at the right places in the process, and using experience to ensure that sufficient QC tasks are done in the project.

Hence, the quality plan for the project is largely a specification of which QC task is to be done and when, and what process and guidelines are to be used for performing the QC task. The choice depends on the nature and goals and constraints of the project. Typically, the QC tasks will be schedulable tasks in the detailed schedule of the project. For example, it will specify what documents will be inspected, what parts of the code will be inspected, and what levels of testing will be performed. The plan can be considerably enhanced if some expectations of defect levels that are expected to be found for the different quality control tasks are also mentioned—these can then aid the monitoring of quality as the project proceeds.

3.5 Risk Management Planning

Q5. What is Risk Management in Software Projects? Give brief ideas of Risk Assessment and Control.

Ans : **(Imp.)**

A software project is a complex undertaking. Unforeseen events may have an adverse impact on a project's ability to meet the cost, schedule, or quality goals. Risk management is an attempt to minimize the chances of failure caused by unplanned events. The aim of risk management is not to avoid getting into projects that have risks but to minimize the impact of risks in the projects that are undertaken.

A risk is a probabilistic event—it may or may not occur. For this reason, we frequently have an optimistic tendency to simply not see risks or to hope that they will not occur. Social and organizational factors also may stigmatize risks and discourage clear identification of them. This kind of attitude gets the project in trouble if the risk events materialize, something that is likely to happen in a large project. Not surprisingly, then, risk management is considered first among the best practices for managing large software projects. It first came to the forefront with Boehm's tutorial on risk management. Since then, several books have targeted risk management for software.

Risk Assessment

The goal of risk assessment is to prioritize the risks so that attention and resources can be focused on the more risky items. Risk identification is the first step in risk assessment, which identifies all the different risks for a particular project. These risks are project-dependent and identifying them is an exercise in envisioning what can go wrong. Methods that can aid risk identification include checklists of possible risks, surveys, meetings and brainstorming, and reviews of plans, processes, and work products.

Checklists of frequently occurring risks are probably the most common tool for risk identification—most organizations prepare a list of commonly occurring risks for projects, prepared from a survey of previous projects. Such a list can form the starting point for identifying risks for the current project.

Based on surveys of experienced project managers, Boehm has produced a list of the top 10 risk items likely to compromise the success of a software project. Some of these risks along with the techniques preferred by management for managing these risks. Top risks in a commercial software organization can be found in.

S.No.	Risk Item	Risk Management Techniques
1.	Personnel shortfalls	Staffing with top talent: Job matching; Team building; Key personnel agreements; Training Prescheduling key people
2.	Unrealistic Schedules and Budgets	Detailed cost and schedule estimation; Design to cost; Incremental development; Software reuse, requirements scrubbing.
3.	Developing the Wrong Software Functions.	Organization analysis; Machine analysis; User surveys; Prototyping; Early user's manuals
4.	Developing the Wrong User Interface	Prototyping Scenarios; Task analysis; User characterization.
5.	Gold Plating	Requirements scrubbing; Prototyping; Cost benefit analysis, Design to cost

The top-ranked risk item is personnel shortfalls. This involves just having fewer people than necessary or not having people with specific skills that a project might require. Some of the ways to manage this risk are to get the top talent possible and to match the needs of the project with the skills of the available personnel. Adequate training, along with having some key personnel for critical areas of the project, will also reduce this risk.

The second item, unrealistic schedules and budgets, happens very frequently due to business and other reasons. It is very common that high-level management imposes a schedule for a software project that is not based on the characteristics of the project and is unrealistic. Underestimation may also happen due to inexperience or optimism.

The next few items are related to requirements. Projects run the risk of developing the wrong software if the requirements analysis is not done properly and if development begins too early. Similarly, often improper user interface may be developed. This requires extensive rework of the user interface later or the software benefits are not obtained because users are reluctant to use it. Gold plating refers to adding features in the software that are only marginally useful. This adds unnecessary risk to the project because gold plating consumes resources and time with little return.

Risk identification merely identifies the undesirable events that might take place during the project, i.e., enumerates the "unforeseen" events that might occur. It does not specify the probabilities of these risks materializing nor the impact on the project if the risks indeed materialize. Hence, the next tasks are risk analysis and prioritization.

In risk analysis, the probability of occurrence of a risk has to be estimated, along with the loss that will occur if the risk does materialize. This is often done through discussion, using experience and understanding of the situation, though structured approaches also exist.

Once the probabilities of risks materializing and losses due to materialization of different risks have been analyzed, they can be prioritized. One approach for prioritization is through the concept of risk exposure (RE), which is sometimes called risk impact. RE is defined by the relationship

RE = Prob(UO) * Loss(UO)

where Prob(UO) is the probability of the risk materializing (i.e., undesirable outcome) andLoss(UO) is the total loss incurred due to the unsatisfactory outcome. The loss is not only the direct financial loss that might be incurred but also any loss in terms of credibility, future business, and loss of property or life. The RE is the expected value of the loss due to a particular risk. For risk prioritization using RE is, the higher the RE, the higher the priority of the risk item.

Risk Control

The main objective of risk management is to identify the top few risk items and then focus on them. Once a project manager has identified and prioritized the risks, the top risks can be easily identified. The question then becomes what to do about them. Knowing the risks is of value only if you can prepare a plan so that their consequences are minimal—that is the basic goal of risk management.

One obvious strategy is risk avoidance, which entails taking actions that will avoid the risk altogether, like the earlier example of shifting the building site to a zone that is not earthquake-prone. For some risks, avoidance might be possible.

For most risks, the strategy is to perform the actions that will either reduce the probability of the risk materializing or reduce the loss due to the risk materializing. These are called risk mitigation steps. To decide what mitigation steps to take, a list of commonly used risk mitigation steps for various risks is very useful here. For the risks mentioned suitable risk mitigation steps are also given.

Note that unlike risk assessment, which is largely an analytical exercise, risk mitigation comprises active measures that have to be performed to minimize the impact of risks. In other words, selecting a risk mitigation step is not just an intellectual exercise. The risk mitigation step must be executed (and monitored). To ensure that the needed actions are executed properly, they must be incorporated into the detailed project schedule.

Risk prioritization and consequent planning are based on the risk perception at the time the risk analysis is performed. Because risks are probabilistic events that frequently depend on external factors, the threat due to risks may change with time as factors change. Clearly, then, the risk perception may also change with time. Furthermore, the risk mitigation steps undertaken may affect the risk perception.

This dynamism implies that risks in a project should not be treated as static and must be monitored and reevaluated periodically. Hence, in addition to monitoring the progress of the planned risk mitigation steps, a project must periodically revisit the risk perception and modify the risk mitigation plans, if needed. Risk monitoring is the activity of monitoring the status of various risks and their control activities. One simple approach for risk monitoring is to analyze the risks afresh at each major milestone, and change the plans as needed.

A Practical Risk Management Planning Approach

Though the concept of risk exposure is rich, a simple practical way of doing risk planning is to simply categorize risks and the impacts in a few levels and then use it for prioritization. This approach is used in many organizations. Here we discuss a simple approach used in an organization . In this approach, the probability of a risk occurring is categorized as low, medium, or high. The risk impact can also be classified as low, medium, and high. With these ratings, the following simple method for risk prioritization can be specified :

S.No	Risk	Probability	Impact	Example	Mitigation
1.	Failure to meet the high performance				Study white papers and guidelines on perf. Train team on perf. tuning. Update review checklist to look for perf. pitfalls. Test application for perf. during system testing.
2.	Lock of people with right skills.	Medium	Medium	Medium	Train resources Review prototype with customers. Develop coding practices.
3.	Complexity of application	Medium	Medium	Medium	Ensure ongoing knowledge transfer. Deploy persons with prior experience wtih the domain.
4.	Manpower attrition.	Medium	Medium	Medium	Train a core group of four people. Rotates assignments among people. Identify backups for key roles.
5.	Unclear requirements	Medium	Medium	Medium	Review a prototype. Conduct a midstage review.

1. For each risk, rate the probability of its happening as low, medium, or high.
2. For each risk, assess its impact on the project as low, medium, or high.
3. Rank the risks based on the probability and effects on the project; for example, a high-probability, high-impact item will have higher rank than a risk item with a medium probability and high impact. In case of conflict, use judgment.
4. Select the top few risk items for mitigation and tracking.
5. An example of this approach is given in Table, which shows the various ratings and the risk mitigation steps. As we can see, the risk management plan, which is essentially this table, can be very brief and focused. For monitoring the risks, one way is to redo risk management planning at milestones, giving more attention to the risks listed in the project plan. During risk monitoring at milestones, reprioritization may occur and mitigation plans for the remainder of the project may change, depending on the current situation and the impact of mitigation steps taken earlier.

3.6 Project Monitoring Plan

Q6. What are different methods used for monitoring a project?

Ans :

A project management plan is merely a document that can be used to guide the execution of a project. Even a good plan is useless unless it is properly executed. And execution cannot be properly driven by the plan unless it is monitored carefully and the actual performance is tracked against the plan.

Monitoring requires measurements to be made to assess the situation of a project. If measurements are to be taken during project execution, we must plan carefully regarding what to measure, when to measure, and how to measure. Hence, measurement planning is a key element in project planning. In addition, how the measurement data will be analyzed and reported must also be planned in advance to avoid the situation of collecting data but not knowing what to do with it. Without careful planning for data collection and its analysis, neither is likely to happen.

(i) Measurements

The basic purpose of measurements in a project is to provide data to project management about the project's current state, such that they can effectively monitor and control the project and ensure that the project goals are met. As project goals are established in terms of software to be delivered, cost, schedule, and quality, for monitoring the state of a project, size, effort, schedule, and defects are the basic measurements that are needed. Schedule is one of the most important metrics because most projects are driven by schedules and deadlines. Only by monitoring the actual schedule can we properly assess if the project is on time or if there is a delay. It is, however, easy to measure because calendar time is usually used in all plans.

Effort is the main resource consumed in a software project. Consequently, tracking of effort is a key activity during monitoring; it is essential for evaluating whether the project is executing within budget. For effort data some type of timesheet system is needed where each person working on the project enters the amount of time spent on the project. For better monitoring, the effort spent on various tasks should be logged separately. Generally effort is recorded through some on-line system, which allows a person to record the amount of time spent against a particular activity in a project. At any point, total effort on an activity can be aggregated.

Because defects have a direct relationship to software quality, tracking of defects is critical for ensuring quality. A large software project may include thousands of defects that are found by different people at different stages. Just to keep track of the defects found and their status, defects must be logged and their closure tracked. If defects found are being logged, monitoring can focus on how many defects have been found so far, what percentages of defects are still open, and other issues. Defect tracking is considered one of the best practices for managing a project.

Size is another fundamental metric because it represents progress toward delivering the desired functionality, and many data (for example, delivered defect density) are normalized with respect to size. The size of delivered software can be measured in terms of LOC (which can be determined through the use of regular editors and line counters) or function points. At a more gross level, just the number of modules or number of features might suffice.

For effective monitoring, a project must plan for collecting these measurements. Most often, organizations provide tools and policy support for recording this basic data, which is then available to project managers for tracking.

(ii) Project Monitoring and Tracking

The main goal of project managers for monitoring a project is to get visibility into the project execution so that they can determine whether any action needs to be taken to ensure that the project goals are met. As project goals are in terms of effort, schedule, and quality, the focus of monitoring is on these aspects. Different levels of monitoring might be done for a project. The three main levels of monitoring are activity level, status reporting, and milestone analysis. Measurements taken on the project are employed for monitoring.

Activity-level monitoring ensures that each activity in the detailed schedule has been done properly and within time. This type of monitoring may be done daily in project team meetings or by the project manager checking the status of all the tasks scheduled to be completed on that day. A completed task is often marked as 100% complete in detailed schedule this is used by tools like the Microsoft Project to track the percentage completion of the overall project or a higher-level task. This monitoring is to ensure that the project continues to proceed as per the planned schedule.

Status reports are often prepared weekly to take stock of what has happened and what needs to be done. Status reports typically contain a summary of the activities successfully completed since the last status report, any activities that have been delayed, any issues in the project that need attention, and if everything is in place for the next week. Again, the purpose of this is to ensure that the project is proceeding as per the planned schedule.

The milestone analysis is done at each milestone or every few weeks, if milestones are too far apart, and is more elaborate. Analysis of actual versus estimated for effort and schedule is often included in the milestone analysis. If the deviation is significant, it may imply that the project may run into trouble and might not meet its objectives. This situation calls for project managers to understand the reasons for the variation and to apply corrective and preventive actions if necessary. Defects found by different quality control tasks, and the number of defects fixed may also be reported. This report monitors the progress of the project with respect to all the goals.

3.7 Detailed Scheduling

Q7. Explain the concept of detailed scheduling.

Ans : **(Imp.)**

- For the detailed schedule, the major phases identified during effort and schedule estimation, are broken into small schedulable activities in a hierarchical manner.
- For example, the detailed design phase can be broken into tasks for developing the detailed design for each module, review of each detailed design, fixing of defects found, and so on. For each detailed task, the project manager estimates the time required to complete it and assigns a suitable resource so that the overall schedule is met, and the overall effort also matches. In addition to the engineering tasks that are the outcome of the development process, the QC tasks identified in the quality plan, the monitoring activities defined in the monitoring plan, and the risk mitigation activities should also be scheduled.
- At each level of refinement, the project manager determines the effort for the overall task from the detailed schedule and checks it against the effort estimates. If this detailed schedule is not consistent with the overall schedule and effort estimates, the detailed schedule must be changed. If it is found that the best detailed schedule cannot match the milestone effort and schedule, then the earlier estimates must be revised. Thus, scheduling is an iterative process.
- Generally, the project manager refines the tasks to a level so that the lowest-level activity can be scheduled to occupy no more than a few days from a single resource. Activities related to tasks such as project management, coordination, database management, and configuration management may also be listed in the schedule, even though these activities have less direct effect on determining the schedule because they are ongoing tasks rather than schedulable activities. Nevertheless, they consume resources and hence are often included in the project schedule.

- Rarely will a project manager complete the detailed schedule of the entire project all at once. Once the overall schedule is fixed, detailing for a phase may only be done at the start of that phase.
- For detailed scheduling, tools like Microsoft Project or a spreadsheet can be very useful. For each lowest-level activity, the project manager specifies the effort, duration, start date, end date, and resources. Dependencies between activities, due either to an inherent dependency (for example, you can conduct a unit test plan for a program only after it has been coded) or to a resource related dependency (the same resource is assigned two tasks), may also be specified. From these tools the overall effort and schedule of higher-level tasks can be determined.
- A detailed project schedule is never static. Changes may be needed because the actual progress in the project may be different from what was planned, because newer tasks are added in response to change requests, or because of other unforeseen situations. Changes are done as and when the need arises. The final schedule, frequently maintained using some suitable tool, is often the most "live" project plan document. During the project, if plans must be changed and additional activities must be done, after the decision is made, the changes must be reflected in the detailed schedule, as this reflects the tasks actually planned to be performed. Hence, the detailed schedule becomes the main document that tracks the activities and schedule.
- It should be noted that only the number of resources is decided during the overall project planning. However, detailed scheduling can be done effectively only after actual assignment of people has been done, as task assignment needs information about the capabilities of the team members. In our discussion above, we have implicitly assumed that the project's team is led by a project manager, who does the planning and task assignment. This form of hierarchical team organization is fairly common, and was earlier called the Chief Programmer Team.

Example

As an example, consider the example of a project from [58]. The overall effort estimate for this project is 501 person-days, or about 24 person-months (this estimation was done using the bottom-up approach discussed earlier). The customer gave approximately 5.5 months to finish the project. Because this is more than the square root of effort in person-months, this schedule was accepted. Hence, these define the effort and schedule goals of the project.

The milestones are determined by using the effort estimates for the phases and an estimate of the number of resources available. Table shows the highlevel schedule of the project. This project uses the RUP process in which initial requirement and design is done in two iterations and the development is done in three iterations. The overall project duration with these milestones is 140 days.

Task	Duration (days)	Work (person - days)	Start	End
Project initiation	33.78	24.2	5/4/00	6/23/00
Regular activities	87.11	35.13	6/5/00	10/16/00
Training	95.11	49.37	5/8/00	9/29/00
Knowledge sharing tasks	78.22	19.56	6/2/00	9/30/00
Inception phase	26.67	22.67	4/3/00	5/12/00
Elaboration Iteration 1	27.56	55.16	5/15/00	6/23/00
Elaboration Iteration 2	8.89	35.88	6/26/00	7/7/00

Construction Iteration 1	8.89	24.63	7/10/00	7/21/00
Construction Iteration 2	6.22	28.22	7/20/00	7/28/00
Construction Iteration 3	6.22	27.03	7/31/00	8/8/00
Transition phase	56	179.62	8/9/00	11/3/00
Back-end work	4.44	6.44	8/14/00	8/18/00

This high-level schedule is an outcome of the overall project planning, and is not suitable for assigning resources and detailed planning.

For detailed scheduling, these tasks are broken into schedulable activities. In this way, the schedule also becomes a checklist of tasks for the project. As mentioned above, this exploding of top-level activities is not done fully at the start but rather takes place many times during the project.

The detailed schedule of the construction-iteration 1 phase of the project. For each activity, the table specifies the activity by a short name, the module to which the activity is contributing, and the effort (the duration may also be specified). For each task, how much is completed is given in the % Complete column. This information is used for activity tracking. The detailed schedule also specifies the resource to which the task is assigned (specified by initials of the person). Sometimes, the predecessors of the activity (the activities upon which the task depends) are also specified. This information helps in determining the critical path and the critical resources. This project finally had a total of about 325 schedulable tasks.

3.8 Design Concept

Q8. Explain briefly about software design and different types of approaches.

Ans :

Meaning

Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.

For assessing user requirements, an SRS (Software Requirement Specification) document is created whereas for coding and implementation, there is a need of more specific and detailed requirements in software terms. The output of this process can directly be used into implementation in programming languages.

Software design is the first step in SDLC (Software Design Life Cycle), which moves the concentration from problem domain to solution domain. It tries to specify how to fulfill the requirements mentioned in SRS.

Software Design Levels

Software design yields three levels of results:

➢ **Architectural Design -** The architectural design is the highest abstract version of the system. It identifies the software as a system with many components interacting with each other. At this level, the designers get the idea of proposed solution domain.

➢ **High-level Design-** The high-level design breaks the 'single entity-multiple component' concept of architectural design into less-abstracted view of sub-systems and modules and depicts their interaction with each other. High-level design focuses on how the system along with all of its components can be implemented in forms of modules. It recognizes modular structure of each sub-system and their relation and interaction among each other.

➢ **Detailed Design-** Detailed design deals with the implementation part of what is seen as a system and its sub-systems in the previous two designs. It is more detailed towards modules and their implementations. It defines logical structure of each module and their interfaces to communicate with other modules.

Approaches

(i) Modularization

Modularization is a technique to divide a software system into multiple discrete and independent modules, which are expected to be capable of carrying out task(s) independently. These modules may work as basic constructs for the entire software. Designers tend to design modules such that they can be executed and/or compiled separately and independently.

Modular design unintentionally follows the rules of 'divide and conquer' problem-solving strategy this is because there are many other benefits attached with the modular design of a software.

Advantage

- Smaller components are easier to maintain
- Program can be divided based on functional aspects
- Desired level of abstraction can be brought in the program
- Components with high cohesion can be re-used again
- Concurrent execution can be made possible
- Desired from security aspect

(ii) Concurrency

Back in time, all software are meant to be executed sequentially. By sequential execution we mean that the coded instruction will be executed one after another implying only one portion of program being activated at any given time. Say, software has multiple modules, then only one of all the modules can be found active at any time of execution.

In software design, concurrency is implemented by splitting the software into multiple independent units of execution, like modules and executing them in parallel. In other words, concurrency provides capability to the software to execute more than one part of code in parallel to each other.

It is necessary for the programmers and designers to recognize those modules, which can be made parallel execution.

Example

The spell check feature in word processor is a module of software, which runs along side the word processor itself.

(iii) Coupling and Cohesion

When a software program is modularized, its tasks are divided into several modules based on some characteristics. As we know, modules are set of instructions put together in order to achieve some tasks. They are though, considered as single entity but may refer to each other to work together. There are measures by which the quality of a design of modules and their interaction among them can be measured. These measures are called coupling and cohesion.

Cohesion

Cohesion is a measure that defines the degree of intra-dependability within elements of a module. The greater the cohesion, the better is the program design.

There are seven types of cohesion, namely –

- **Co-incidental cohesion -** It is unplanned and random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularization. Because it is unplanned, it may serve confusion to the programmers and is generally not-accepted.
- **Logical cohesion -** When logically categorized elements are put together into a module, it is called logical cohesion.

- **Temporal Cohesion -** When elements of module are organized such that they are processed at a similar point in time, it is called temporal cohesion.
- **Procedural cohesion -** When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called procedural cohesion.
- **Communicational cohesion -** When elements of module are grouped together, which are executed sequentially and work on same data (information), it is called communicational cohesion.
- **Sequential cohesion -** When elements of module are grouped because the output of one element serves as input to another and so on, it is called sequential cohesion.
- **Functional cohesion -** It is considered to be the highest degree of cohesion, and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well-defined function. It can also be reused.

Coupling

Coupling is a measure that defines the level of inter-dependability among modules of a program. It tells at what level the modules interfere and interact with each other. The lower the coupling, the better the program. There are five levels of coupling, namely -

- **Content coupling -** When a module can directly access or modify or refer to the content of another module, it is called content level coupling.
- **Common coupling-** When multiple modules have read and write access to some global data, it is called common or global coupling.
- **Control coupling-** Two modules are called control-coupled if one of them decides the function of the other module or changes its flow of execution.
- **Stamp coupling-** When multiple modules share common data structure and work on different part of it, it is called stamp coupling.
- **Data coupling-** Data coupling is when two modules interact with each other by means of passing data (as parameter). If a module passes data structure as parameter, then the receiving module should use all its components.

  Ideally, no coupling is considered to be the best.

(iv) Design Verification

The output of software design process is design documentation, pseudo codes, detailed logic diagrams, process diagrams, and detailed description of all functional or non-functional requirements.

The next phase, which is the implementation of software, depends on all outputs mentioned above.

It is then becomes necessary to verify the output before proceeding to the next phase. The early any mistake is detected, the better it is or it might not be detected until testing of the product. If the outputs of design phase are in formal notation form, then their associated tools for verification should be used otherwise a thorough design review can be used for verification and validation.

By structured verification approach, reviewers can detect defects that might be caused by overlooking some conditions. A good design review is important for good software design, accuracy and quality.

3.8.1 Data Design in Software Engineering

Q9. Write short notes on data design?

Ans :

Data design is the first design activity, which results in fewer complexes, modular and efficient program structure. The information domain model developed during analysis phase is transformed into data structures needed for implementing the software. The data objects, attributes, and relationships depicted in entity relationship diagrams and the information stored in data dictionary provide a base for data design activity. During the data design process, data types are specified along with the integrity rules required for the data. For specifying and designing efficient data structures, some principles should be followed.

These principles are listed below.

(i) The data structures needed for implementing the software as well-as the operations that can be applied on them should be identified.

(ii) A data dictionary should be developed to depict how different data objects interact with each other and what constraints are to be imposed on the elements of data structure.

(iii) Stepwise refinement should be used in data design process and detailed design decisions should be made later in the process.

(iv) Only those modules that need to access data stored in a data structure directly should be aware of the representation of the data structure.

(v) A library containing the set of useful data structures along with the operations that can be performed on them should be maintained.

(vi) Language used for developing the system should support abstract data types.

The structure of data can be viewed at three levels, namely, program component level, application level, and business level. At the program component level, the design of data structures and the algorithms required to manipulate them is necessary, if high-quality software is desired. At the application level, it is crucial to convert the data model into a database so that the specific business objectives of a system could be achieved. At the business level, the collection of information stored in different databases should be reorganized into data warehouse, which enables data mining that has an influential impact on the business.

3.8.2 Function Oriented Design, Object Oriented Design, Detailed Design

Q10. What are the strategies followed in designing software?

Ans : **(Imp.)**

Software design is a process to conceptualize the software requirements into software implementation. Software design takes the user requirements as challenges and tries to find optimum solution. While the software is being conceptualized, a plan is chalked out to find the best possible design for implementing the intended solution.

There are multiple variants of software design. Let us study them briefly:

(i) Structured Design

Structured design is a conceptualization of problem into several well-organized elements of solution. It is basically concerned with the solution design. Benefit of structured design is, it gives better understanding of how the problem is being solved. Structured design also makes it simpler for designer to concentrate on the problem more accurately.

Structured design is mostly based on 'divide and conquer' strategy where a problem is broken into several small problems and each small problem is individually solved until the whole problem is solved.

The small pieces of problem are solved by means of solution modules. Structured design emphasis that these modules be well organized in order to achieve precise solution.

These modules are arranged in hierarchy. They communicate with each other. A good structured design always follows some rules for communication among multiple modules, namely-

➢ **Cohesion** - grouping of all functionally related elements.

➢ **Coupling** - communication between different modules.

A good structured design has high cohesion and low coupling arrangements.

(ii) Function Oriented Design

In function-oriented design, the system is comprised of many smaller sub-systems known as functions. These functions are capable of performing significant task in the system. The system is considered as top view of all functions.

Function oriented design inherits some properties of structured design where divide and conquer methodology is used.

This design mechanism divides the whole system into smaller functions, which provides means of abstraction by concealing the information and their operation.. These functional modules can share information among themselves by means of information passing and using information available globally.

Another characteristic of functions is that when a program calls a function, the function changes the state of the program, which sometimes is not acceptable by other modules. Function oriented design works well where the system state does not matter and program/functions work on input rather than on a state.

Design Process

- The whole system is seen as how data flows in the system by means of data flow diagram.
- DFD depicts how functions changes data and state of entire system.
- The entire system is logically broken down into smaller units known as functions on the basis of their operation in the system.
- Each function is then described at large.

(iii) Object Oriented Design

Object oriented design works around the entities and their characteristics instead of functions involved in the software system. This design strategies focuses on entities and its characteristics. The whole concept of software solution revolves around the engaged entities.

Let us see the important concepts of Object Oriented Design :

- **Objects -** All entities involved in the solution design are known as objects. For example, person, banks, company and customers are treated as objects. Every entity has some attributes associated to it and has some methods to perform on the attributes.
- **Classes -** A class is a generalized description of an object. An object is an instance of a class. Class defines all the attributes, which an object can have and methods, which defines the functionality of the object.
- In the solution design, attributes are stored as variables and functionalities are defined by means of methods or procedures.
- **Encapsulation -** In OOD, the attributes (data variables) and methods (operation on the data) are bundled together is called encapsulation. Encapsulation not only bundles important information of an object together, but also restricts access of the data and methods from the outside world. This is called information hiding.
- **Inheritance -** OOD allows similar classes to stack up in hierarchical manner where the lower or sub-classes can import, implement and re-use allowed variables and methods from their immediate super classes. This property of OOD is known as inheritance. This makes it easier to define specific class and to create generalized classes from specific ones.
- **Polymorphism -** OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned same name. This is called polymorphism, which allows a single interface performing tasks for different types. Depending upon how the function is invoked, respective portion of the code gets executed.

(iv) Design Process

Software design process can be perceived as series of well-defined steps. Though it varies according to design approach (function oriented or object oriented, yet It may have the following steps involved:

- A solution design is created from requirement or previous used system and/or system sequence diagram.
- Objects are identified and grouped into classes on behalf of similarity in attribute characteristics.
- Class hierarchy and relation among them is defined.
- Application framework is defined.

3.9 Verification

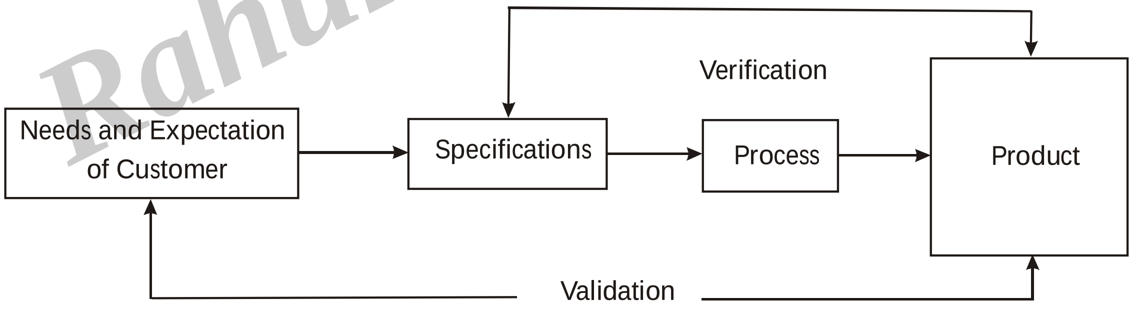
Q11. What is software verification?

Ans :

Verification makes sure that the product is designed to deliver all functionality to the customer.

- Verification is done at the starting of the development process. It includes reviews and meetings, walk-throughs, inspection, etc. to evaluate documents, plans, code, requirements and specifications.

Suppose you are building a table. Here the verification is about checking all the parts of the table, whether all the four legs are of correct size or not. If one leg of table is not of the right size it will imbalance the end product. Similar behavior is also noticed in case of the software product or application. If any feature of software product or application is not up to the mark or if any defect is found then it will result into the failure of the end product. Hence, verification is very important. It takes place at the starting of the development process.



Software verification and validation

- It answers the questions like: **Am I building the product right?**
- Am I accessing the data right (in the right place; in the right way).
- It is a Low level activity
- Performed during development on key artifacts, like walk throughs, reviews and inspections, mentor feedback, training, checklists and standards.
- Demonstration of consistency, completeness, and correctness of the software at each stage and between each stage of the development life cycle.

According to the Capability Maturity Model (CMM) we can also define verification as the process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

Advantages of Software Verification:

1. Verification helps in lowering down the count of the defect in the later stages of development.
2. Verifying the product at the starting phase of the development will help in understanding the product in a better way.
3. It reduces the chances of failures in the software application or product.
4. It helps in building the product as per the customer specifications and needs.

3.10 Metrics

Q12. What are metrics, Measurements & Models of Project Management & Software Management?

Ans : **(Imp.)**

Once measures are collected they are converted into metrics for use. IEEE defines metric as ‘a quantitative measure of the degree to which a system, component, or process possesses a given attribute.’ The goal of software metrics is to identify and control essential parameters that affect software development. Other objectives of using software metrics are listed below.

- ➢ Measuring the size of the software quantitatively.
- ➢ Assessing the level of complexity involved.
- ➢ Assessing the strength of the module by measuring coupling.
- ➢ Assessing the testing techniques.
- ➢ Specifying when to stop testing.
- ➢ Determining the date of release of the software.
- ➢ Estimating cost of resources and project schedule.

Software metrics help project managers to gain an insight into the efficiency of the software process, project, and product. This is possible by collecting quality and productivity data and then analyzing and comparing these data with past averages in order to know whether quality improvements have occurred. Also, when metrics are applied in a consistent manner, it helps in project planning and project management activity. For example, schedule-based resource allocation can be effectively enhanced with the help of metrics.

Difference in Measures, Metrics, and Indicators

Metrics is often used interchangeably with measure and measurement. However, it is important to note the differences between them. Measure can be defined as quantitative indication of amount, dimension, capacity, or size of product and process attributes. Measurement can be defined as the process of determining the measure. Metrics can be defined as quantitative measures that allow software engineers to identify the efficiency and improve the quality of software process, project, and product.

To understand the difference, let us consider an example. A measure is established when a number of errors is (single data point) detected in a software component. Measurement is the process of collecting one or more data points. In other words, measurement is established when many components are reviewed and tested individually to collect the measure of a number of errors in all these components. Metrics are associated with individual measure in some manner. That is, metrics are related to detection of errors found per review or the average number of errors found per unit test.

Once measures and metrics have been developed, indicators are obtained. These indicators provide a detailed insight into the software process, software project, or intermediate product. Indicators also enable software engineers or project managers to adjust software processes and improve software products, if required. For example, measurement dashboards or key indicators are used to monitor progress and initiate change. Arranged together, indicators provide snapshots of the system's performance.

Measured Data

Before data is collected and used, it is necessary to know the type of data involved in the software metrics. Table lists different types of data, which are identified in metrics along with their description and the possible operations that can be performed on them.

Type of data	Possible operations	Description of data
Nominal	$=, \neq$	Categories
Ordinal	$<, >$	Ranking
Interval	$+, -$	Differences
Ratio	/	Absolute zero

- **Nominal data:** Data in the program can be measured by placing it under a category. This category of program can be a database program, application program, or an operating system program. For such data, operation of arithmetic type and ranking of values in any order (increasing or decreasing) is not possible. The only operation that can be performed is to determine whether program 'X' is the same as program 'Y'.
- **Ordinal data:** Data can be ranked according to the data values. For example, experience in application domain can be rated as very low, low, medium, or high. Thus, experience can easily be ranked according to its rating.
- **Interval data:** Data values can be ranked and substantial differences between them can also be shown. For example, a program with complexity level 8 is said to be 4 units more complex than a program with complexity level 4.
- **Ratio data:** Data values are associated with a ratio scale, which possesses an absolute zero and allows meaningful ratios to be calculated. For example, program lines expressed in lines of code.
- It is desirable to know the measurement scale for metrics. For example, if metrics values are used to represent a model for a software process, then metrics associated with the ratio scale may be preferred.

Guidelines for Software Metrics

Although many software metrics have been proposed over a period of time, ideal software metric is the one which is easy to understand, effective, and efficient. In order to develop ideal metrics, software metrics should be validated and characterized effectively. For this, it is important to develop metrics using some specific guidelines, which are listed below.

- **Simple and computable:** Derivation of software metrics should be easy to learn and should involve average amount of time and effort.
- **Consistent and objective:** Unambiguous results should be delivered by software metrics.
- **Consistent in the use of units and dimensions:** Mathematical computation of the metrics should involve use of dimensions and units in a consistent manner.
- **Programming language independent:** Metrics should be developed on the basis of the analysis model, design model, or program's structure.
- **High quality:** Effective software metrics should lead to a high-quality software product.
- **Easy to calibrate:** Metrics should be easy to adapt according to project requirements.
- **Easy to obtain:** Metrics should be developed at a reasonable cost.
- **Validation:** Metrics should be validated before being used for making any decisions.
- **Robust:** Metrics should be relatively insensitive to small changes in process, project, or product.
- **Value:** Value of metrics should increase or decrease with the value of the software characteristics they represent. For this, the value of metrics should be within a meaningful range. For example, metrics can be in a range of 0 to 5.

Q13. What are the different metrics used in software engineering in various stages?

Ans :

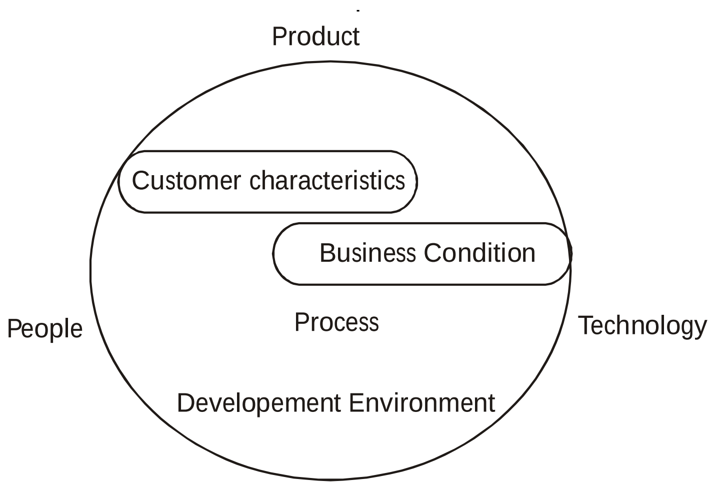
Measurement is done by metrics. Three parameters are measured: process measurement through process metrics, product measurement through product metrics, and project measurement through project metrics.

Process metrics assess the effectiveness and quality of software process, determine maturity of the process, effort required in the process, effectiveness of defect removal during development, and so on. Product metrics is the measurement of work product produced during different phases of software development. Project metrics illustrate the project characteristics and their execution.

1) Process Metrics

To improve any process, it is necessary to measure its specified attributes, develop a set of meaningful metrics based on these attributes, and then use these metrics to obtain indicators in order to derive a strategy for process improvement.

Using software process metrics, software engineers are able to assess the efficiency of the software process that is performed using the process as a framework. Process is placed at the centre of the triangle connecting three factors (product, people, and technology), which have an important influence on software quality and organization performance. The skill and motivation of the people, the complexity of the product and the level of technology used in the software development have an important influence on the quality and team performance. The process triangle exists within the circle of environmental conditions, which includes development environment, business conditions, and customer /user characteristics.



To measure the efficiency and effectiveness of the software process, a set of metrics is formulated based on the outcomes derived from the process. These outcomes are listed below.

- Number of errors found before the software release
- Defect detected and reported by the user after delivery of the software
- Time spent in fixing errors
- Work products delivered
- Human effort used
- Time expended
- Conformity to schedule
- Wait time
- Number of contract modifications
- Estimated cost compared to actual cost.

Note that process metrics can also be derived using the characteristics of a particular software engineering activity. For example, an organization may measure the effort and time spent by considering the user interface design.

It is observed that process metrics are of two types, namely, private and public. Private Metrics are private to the individual and serve as an indicator only for the specified individual(s). Defect rates by a software module and defect errors by an individual are examples of private process metrics. Note that some process metrics are public to all team members but private to the project. These include errors detected while performing formal technical reviews and defects reported about various functions included in the software.

Public metrics include information that was private to both individuals and teams. Project-level defect rates, effort and related data are collected, analyzed and assessed in order to obtain indicators that help in improving the organizational process performance.

Process Metrics Etiquette

Process metrics can provide substantial benefits as the organization works to improve its process maturity. However, these metrics can be misused and create problems for the organization. In order to avoid this misuse, some guidelines have been defined, which can be used both by managers and software engineers. These guidelines are listed below.

- Rational thinking and organizational sensitivity should be considered while analyzing metrics data.
- Feedback should be provided on a regular basis to the individuals or teams involved in collecting measures and metrics.
- Metrics should not appraise or threaten individuals.
- Since metrics are used to indicate a need for process improvement, any metric indicating this problem should not be considered harmful.
- Use of single metrics should be avoided.

  As an organization becomes familiar with process metrics, the derivation of simple indicators leads to a stringent approach called Statistical Software Process Improvement (SSPI). SSPI uses software failure analysis to collect information about all errors (it is detected before delivery of the software) and defects (it is detected after software is delivered to the user) encountered during the development of a product or system.

2. Product Metrics

In software development process, a working product is developed at the end of each successful phase. Each product can be measured at any stage of its development. Metrics are developed for these products so that they can indicate whether a product is developed according to the user requirements. If a product does not meet user requirements, then the necessary actions are taken in the respective phase.

Product metrics help software engineer to detect and correct potential problems before they result in catastrophic defects. In addition, product metrics assess the internal product attributes in order to know the efficiency of the following.

- Analysis, design, and code model
- Potency of test cases
- Overall quality of the software under development.

  Various metrics formulated for products in the development process are listed below.

(i) **Metrics for analysis model:** These address various aspects of the analysis model such as system functionality, system size, and so on.

(ii) **Metrics for design model:** These allow software engineers to assess the quality of design and include architectural design metrics, component-level design metrics, and so on.

(iii) **Metrics for source code:** These assess source code complexity, maintainability, and other characteristics.

(iv) **Metrics for testing:** These help to design efficient and effective test cases and also evaluate the effectiveness of testing.

(v) **Metrics for maintenance:** These assess the stability of the software product.

(i) Metrics for the Analysis Model

There are only a few metrics that have been proposed for the analysis model. However, it is possible to use metrics for project estimation in the context of the analysis model. These metrics are used to examine the analysis model with the objective of predicting the size of the resultant system. Size acts as an indicator of increased coding, integration, and testing effort; sometimes it also acts as an indicator of complexity involved in the software design. Function point and lines of code are the commonly used methods for size estimation.

Function Point (FP) Metric

The function point metric, which was proposed by A.J Albrecht, is used to measure the functionality delivered by the system, estimate the effort, predict the number of errors, and estimate the number of components in the system. Function point is derived by using a relationship between the complexity of software and the information domain value. Information domain values used in function point include the number of external inputs, external outputs, external inquires, internal logical files, and the number of external interface files.

Lines of Code (LOC)

Lines of code (LOC) is one of the most widely used methods for size estimation. LOC can be defined as the number of delivered lines of code, excluding comments and blank lines. It is highly dependent on the programming language used as code writing varies from one programming language to another. Example, lines of code written (for a large program) in assembly language are more than lines of code written in C++.

From LOC, simple size-oriented metrics can be derived such as errors per KLOC (thousand lines of code), defects per KLOC, cost per KLOC, and so on. LOC has also been used to predict program complexity, development effort, programmer performance, and so on. For example, Hasltead proposed a number of metrics, which are used to calculate program length, program volume, program difficulty, and development effort.

Metrics for Specification Quality

To evaluate the quality of analysis model and requirements specification, a set of characteristics has been proposed. These characteristics include specificity, completeness, correctness, under-standability, verifiability, internal and external consistency, &achievability, concision, traceability, modifiability, precision, and reusability.

Most of the characteristics listed above are qualitative in nature. However, each of these characteristics can be represented by using one or more metrics. For example, if there are n_r requirements in a specification, then n_r can be calculated by the following equation.

$n_r = n_f + n_{rf}$

Where

n_f = number of functional requirements

n_{nf} = number of non-functional requirements.

In order to determine the specificity of requirements, a metric based on the consistency of the reviewer's understanding of each requirement has been proposed. This metric is represented by the following equation.

$Q_1 = n_{ui}/n_r$

Where n_{ui} = number of requirements for which reviewers have same understanding, Q_1 = specificity.

Ambiguity of the specification depends on the value of Q. If the value of Q is close to 1 then the probability of having any ambiguity is less.

Completeness of the functional requirements can be calculated by the following equation.

$Q_2 = n_u / [n_j * n_s]$

Where

n_u = number of unique function requirements

n_i = number of inputs defined by the specification

n_s = number of specified state.

Q_2 in the above equation considers only functional requirements and ignores non-functional requirements. In order to consider non-functional requirements, it is necessary to consider the degree to which requirements have been validated. This can be represented by the following equation.

$Q_3 = n_c / [n_c + n_{nv}]$

Where

n_c = number of requirements validated as correct

n_{nv} = number of requirements, which are yet to be validated.

(ii) Metrics for Software Design

The success of a software project depends largely on the quality and effectiveness of the software design. Hence, it is important to develop software metrics from which meaningful indicators can be derived. With the help of these indicators, necessary steps are taken to design the software according to the user requirements. Various design metrics such as architectural design metrics, component-level design metrics, user-interface design metrics, and metrics for object-oriented design are used to indicate the complexity, quality, and so on of the software design.

Architectural Design Metrics

These metrics focus on the features of the program architecture with stress on architectural structure and effectiveness of components (or modules) within the architecture. In architectural design metrics, three software design complexity measures are defined, namely, structural complexity, data complexity, and system complexity.

In hierarchical architectures (call and return architecture), say module 'j', structural complexity is calculated by the following equation.

$S(j) = f^2_{out}(j)$

Where

$f_{out}(j)$ = fan-out of module 'j' [Here, fan-out means number of modules that are subordinating module j].

Complexity in the internal interface for a module 'j' is indicated with the help of data complexity, which is calculated by the following equation.

$D(j) = V(j) / [f_{out}(j)+l]$

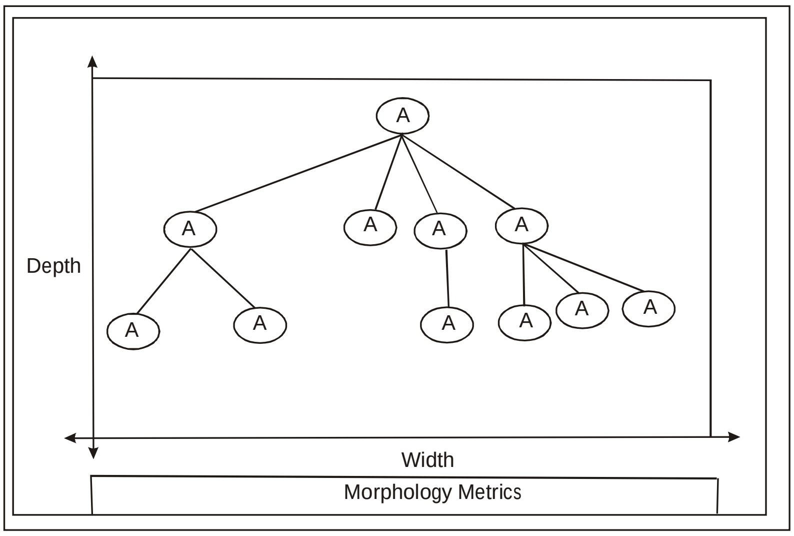
Where

$V(j)$ = number of input and output variables passed to and from module 'j'.

System complexity is the sum of structural complexity and data complexity and is calculated by the following equation.

$C(j) = S(j) + D(j)$

The complexity of a system increases with increase in structural complexity, data complexity, and system complexity, which in turn increases the integration and testing effort in the later stages.



In addition, various other metrics like simple morphology metrics are also used. These metrics allow comparison of different program architecture using a set of straightforward dimensions. A metric can be developed by referring to call and return architecture. This metric can be defined by the following equation.

Size = n+a

Where n = number of nodes, a= number of arcs.

For example, there are 11 nodes and 10 arcs. Here, Size can be calculated by the following equation.

Size = n+a = 11+10+21.

Depth is defined as the longest path from the top node (root) to the leaf node and width is defined as the maximum number of nodes at any one level.

Coupling of the architecture is indicated by arc-to-node ratio. This ratio also measures the connectivity density of the architecture and is calculated by the following equation.

r=a/n

Quality of software design also plays an important role in determining the overall quality of the software. Many software quality indicators that are based on measurable design characteristics of a computer program have been proposed. One of them is Design Structural Quality Index (DSQI), which is derived from the information obtained from data and architectural design. To calculate DSQI, a number of steps are followed, which are listed below.

1. To calculate DSQI, the following values must be determined.
   - Number of components in program architecture (S_1)
   - Number of components whose correct function is determined by the Source of input data (S_2)
   - Number of components whose correct function· depends on previous processing (S_3)
   - Number of database items (S_4)
   - Number of different database items (S_5)
   - Number of database segments (S_6)
   - Number of components having single entry and exit (S_7).
2. Once all the values from S_1 to S_7 are known, some intermediate values are calculated, which are listed below.

   Program structure (D_1): If discrete methods are used for developing architectural design then D_1= 1, else D_1 = 0

   Module independence (D_2): D_2 = 1-(S_2/S_1)

   Modules not dependent on prior processing (D_3): D_3 = 1-(S_3/S_1)

   Database size (D_4): D_4 = 1-(S_5/S_4)

   Database compartmentalization (D_5):D_5 = 1-(S_6/S_4)

   Module entrance/exit characteristic (D_6): D_6 = 1-(S_7/S_1)
3. Once all the intermediate values are calculated, OSQI is calculated by the following equation.

   DSQI = “W_iD_i

Where

i = 1 to 6

"W_i = 1 (W_i is the weighting of the importance of intermediate values).

In conventional software, the focus of component – level design metrics is on the internal characteristics of the software components; The software engineer can judge the quality of the component-level design by measuring module cohesion, coupling and complexity; Component-level design metrics are applied after procedural design is final. Various metrics developed for component-level design are listed below.

- **Cohesion metrics:** Cohesiveness of a module can be indicated by the definitions of the following five concepts and measures.
- **Data slice:** Defined as a backward walk through a module, which looks for values of data that affect the state of the module as the walk starts
- **Data tokens:** Defined as a set of variables defined for a module
- **Glue tokens:** Defined as a set of data tokens, which lies on one or more data slice
- **Superglue tokens:** Defined as tokens, which are present in every data slice in the module
- **Stickiness:** Defined as the stickiness of the glue token, which depends on the number of data slices that it binds.
- **Coupling Metrics:** This metric indicates the degree to which a module is connected to other modules, global data and the outside environment. A metric for module coupling has been proposed, which includes data and control flow coupling, global coupling, and environmental coupling.

Complexity Metrics: Different types of software metrics can be calculated to ascertain the complexity of program control flow. One of the most widely used complexity metrics for ascertaining the complexity of the program is cyclomatic complexity.

Many metrics have been proposed for user interface design. However, layout appropriateness metric and cohesion metric for user interface design are the commonly used metrics. Layout Appropriateness (LA) metric is an important metric for user interface design. A typical Graphical User Interface (GUI) uses many layout entities such as icons, text, menus, windows, and so on. These layout entities help the users in completing their tasks easily. In to complete a given task with the help of GUI, the user moves from one layout entity to another.

Appropriateness of the interface can be shown by absolute and relative positions of each layout entities, frequency with which layout entity is used, and the cost of changeover from one layout entity to another.

Cohesion metric for user interface measures the connection among the on screen contents. Cohesion for user interface becomes high when content presented on the screen is from a single major data object (defined in the analysis model). On the other hand, if content presented on the screen is from different data objects, then cohesion for user interface is low.

In addition to these metrics, the direct measure of user interface interaction focuses on activities like measurement of time required in completing specific activity, time required in recovering from an error condition, counts of specific operation, text density, and text size. Once all these measures are collected, they are organized to form meaningful user interface metrics, which can help in improving the quality of the user interface.

(iii) Metrics for Object-oriented Design

In order to develop metrics for object-oriented (OO) design, nine distinct and measurable characteristics of OO design are considered, which are listed below.

- **Complexity:** Determined by assessing how classes are related to each other.
- **Coupling:** Defined as the physical connection between OO design elements.
- **Sufficiency:** Defined as the degree to which an abstraction possesses the features required of it.
- **Cohesion:** Determined by analyzing the degree to which a set of properties that the class possesses is part of the problem domain or design domain.
- **Primitiveness:** Indicates the degree to which the operation is atomic.
- **Similarity:** Indicates similarity between two or more classes in terms of their structure, function, behavior, or purpose.
- **Volatility:** Defined as the probability of occurrence of change in the OO design.
- **Size:** Defined with the help of four different views, namely, population, volume, length, and functionality. Population is measured by calculating the total number of OO entities, which can be in the form of classes or operations. Volume measures are collected dynamically at any given point of time. Length is a measure of interconnected designs such as depth of inheritance tree. Functionality indicates the value rendered to the user by the OO application.

(iv) Metrics for Coding

Halstead proposed the first analytic laws for computer science by using a set of primitive measures, which can be derived once the design phase is complete and code is generated. These measures are listed below.

n_l = number of distinct operators in a program

n_2 = number of distinct operands in a program

N_1 = total number of operators

N_2 = total number of operands.

By using these measures, Halstead developed an expression for overall program length, program volume, program difficulty, development effort, and so on.

Program length (N) can be calculated by using the following equation.

$N = n_1 \log_2 n_l + n_2 \log_2 n_2$.

Program volume (V) can be calculated by using the following equation.

$V = N \log_2 (n_1 + n_2)$.

Note that program volume depends on the programming language used and represents the volume of information (in bits) required to specify a program. Volume ratio (L)can be calculated by using the following equation.

$$L = \frac{\text{Volume of the most compact form of a program}}{\text{Volume of the actual program}}$$

Where, value of L must be less than 1. Volume ratio can also be calculated by using the following equation.

$L = (2/n_1)^* \; (n_2/N_2)$.

Program difficulty level (D) and effort (E)can be calculated by using the following equations.

$D = (n_1/2)*(N_2/n_2)$.

$E = D * V$.

(v) Metrics for Software Testing

Majority of the metrics used for testing focus on testing process rather than the technical characteristics of test. Generally, testers use metrics for analysis, design, and coding to guide them in design and execution of test cases.

Function point can be effectively used to estimate testing effort. Various characteristics like errors discovered, number of test cases needed, testing effort, and so on can be determined by estimating the number of function points in the current project and comparing them with any previous project.

Metrics used for architectural design can be used to indicate how integration testing can be carried out. In addition, cyclomatic complexity can be used effectively as a metric in the basis-path testing to determine the number of test cases needed.

Halstead measures can be used to derive metrics for testing effort. By using program volume (V) and program level (PL),Halstead effort (e)can be calculated by the following equations.

$e = V/ PL$ Where

$PL = 1/ [(n_1/2) * (N_2/n_2)]$ … (1)

For a particular module (z), the percentage of overall testing effort allocated can be calculated by the following equation.

Percentage of testing effort (z) = e(z)/”e(i)

Where, e(z) is calculated for module z with the help of equation (1). Summation in the denominator is the sum of Halstead effort (e) in all the modules of the system.

For developing metrics for object-oriented (OO) testing, different types of design metrics that have a direct impact on the testability of object-oriented system are considered. While developing metrics for OO testing, inheritance and encapsulation are also considered. A set of metrics proposed for OO testing is listed below.

- **Lack of cohesion in methods (LCOM):** This indicates the number of states to be tested. LCOM indicates the number of methods that access one or more same attributes. The value of LCOM is 0, if no methods access the same attributes. As the value of LCOM increases, more states need to be tested.
- **Percent public and protected (PAP):** This shows the number of class attributes, which are public or protected. Probability of adverse effects among classes increases with increase in value of PAP as public and protected attributes lead to potentially higher coupling.
- **Public access to data members (PAD):** This shows the number of classes that can access attributes of another class. Adverse effects among classes increase as the value of PAD increases.
- **Number of root classes (NOR):** This specifies the number of different class hierarchies, which are described in the design model. Testing effort increases with increase in NOR.
- **Fan-in (FIN):** This indicates multiple inheritances. If value of FIN is greater than 1, it indicates that the class inherits its attributes and operations from many root classes. Note that this situation (where FIN> 1) should be avoided.

UNIT IV

Coding and Unit Testing: Programming Principles and Guidelines, Incrementally developing code, managing evolving code, unit testing, code inspection, Metrics.

Testing: Testing Concepts, Testing Process, Black Box testing, White box testing, Metrics.

4.1 Coding

Q1. Explain various Programming Practices used in Coding. What is meant by Information Hiding?

Ans : **(Imp.)**

The primary goal of the coding phase is to translate the given design into source code in a given programming language, so that code is simple, easy to test, and easy to understand and modify. Simplicity and clarity are the properties that a programmer should strive for. All designs contain hierarchies, as creating a hierarchy is a natural way to manage complexity. Most design methodologies for software also produce hierarchies.

In a top down implementation, the implementation starts from the top of the hierarchy and proceeds to the lower levels. First the main module is implemented, then its subordinates are implemented, and their subordinates, and so on. In a bottom up implementation, the process is the reverse.

The development starts with implementing the modules at the bottom of the hierarchy and proceeds through the higher levels unit until it reaches the top. We want to build the system in parts, even though the design of the entire system has been done. This is necessitated by the fact that for large systems it is simply not feasible or desirable to build the whole system and then test it.

When we proceed top down, for testing a set of modules at the top of the hierarchy, stubs will have to be written for the lower level modules that the set of modules under testing invoke. On the other hand when we proceed bottom up all modules that are lower in the hierarchy have been developed and driver modules are needed to invoke these modules under testing. In practice in large systems, a combination of the two approaches is used during coding.

The top modules of the system generally contain the overall view of the system and may even contain the user interfaces. On the other hand, the bottom level modules typically form the service routines that provide the basic operations used by higher level modules. A program has a static structure as well as a dynamic structure. The static structure is the structure of the text of the program, which is usually just a linear organization of statement of the program. The dynamic structure of the program is the sequence of statements executed during the execution of the program.

The goal of structured programming is to ensure that the static structure and the dynamic structures are the same. That is, statements executed during the execution of a program are same as the sequence of statements in the text of that program. As the statements in a program text are linearly organized, the objective of structured programming becomes developing programs whose control flow during execution is linearized and follows the linear organization of the program text.

In structured programming a statement is not a simple assignment statement, it is a structured statement. The key property of a structured statement is that it has a single entry and single exit. That is, during execution, the execution of the (structured) statement starts from one defined point and the execution terminates at alone defined point. With single entry and single exit statements, we can view a program as a sequence of statements.

And if all statements are structured statements, then during execution, the sequence of execution of these statements will be the same as the sequence in the program text. Hence, by using single entry and single exit statement, the correspondence between the static and dynamic structures can be obtained.

Structured programming practice forms a good basis and guideline for writing programs clearly. A software solution to a problem always contains data structures that are means to represent information in the problem domain. That is when software is developed to solve a problem; the software uses some data structures to capture the information in the problem domain.

With the problem information represented internally as data structures, the required functionality of the problem domain, which is in terms of information in that domain, can be implemented as software operations on the data structures. Hence, any software solution to a problem contains data structures that represent information in the problem domain.

When the information is represented as data structures, the same principle should be applied, and only some defined operations should be performed on the data structures. This essentially, is the principle of information hiding. The information captured in the data structures that represent the operations performed on the information should be visible. Information hiding can reduce the coupling between modules and make the system more maintainable.

4.1.1 Programming Principles

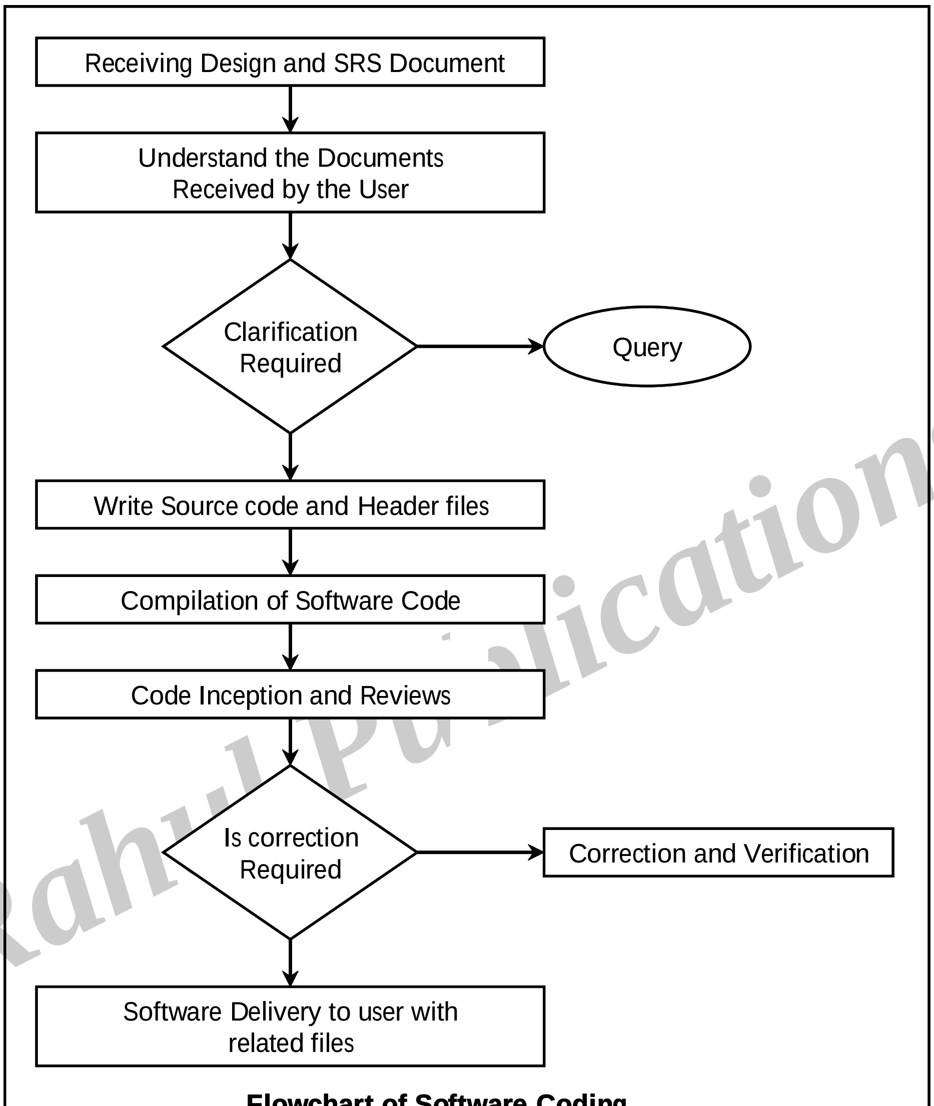
Q2. Explain the methodology of Coding in Software Engineering. What are its Features.

Ans :

The methodology refers to a set of well-documented procedures and guidelines used in the analysis, design, and implementation of programs. Coding methodology includes a diagrammatic notation for documenting the results of the procedure. It also includes an objective set (ideally quantified) of criteria for determining whether the results of the procedure are of the desired quality. The steps to use coding' methodology are listed below.

1. The software development team begins its work by reviewing and understanding the design and requirements specification documents. These documents are essential for understanding user requirements and creating a framework for the software code.
2. In case the software development team is unable to understand user requirements correctly and further clarification is required, the queries are sent back to the user. In addition, the software development team also returns the requirements that are understood by them.
3. After the requirements are clearly understood by the software development team, the design and specifications are implemented in source code, supporting files, and the header files. Note that while writing the software code, the coding style guidelines should be followed. In some cases, there may be a proposal of change in hardware or software specifications. However, the requests for change are implemented only after the approval of the user.
4. When the software code is completely written, it is compiled along with other required files.
5. Code inspection and reviews are conducted after the compilation. These methods are used to correct and verify errors in the software code.
6. Software testing is carried out to detect and correct errors in each module of the software code.
7. After the software code is tested, the software is delivered to the user along with the relevant code files, header files, and documentation files.
8. In Software Coding Process further change and clarifications are required in the design or SRS, the software development team raises a query, which is sent to the user with

the document containing what the software development team understood from the documents sent by the user. Changes are made only when the user has a positive response to the queries raised by the software development team.


Receiving Design and SRS Document
Understand the Documents Received by the User
Clarification Required
Query
Write Source code and Header files
Compilation of Software Code
Code Inception and Reviews
Is correction Required
Correction and Verification
Software Delivery to user with related files

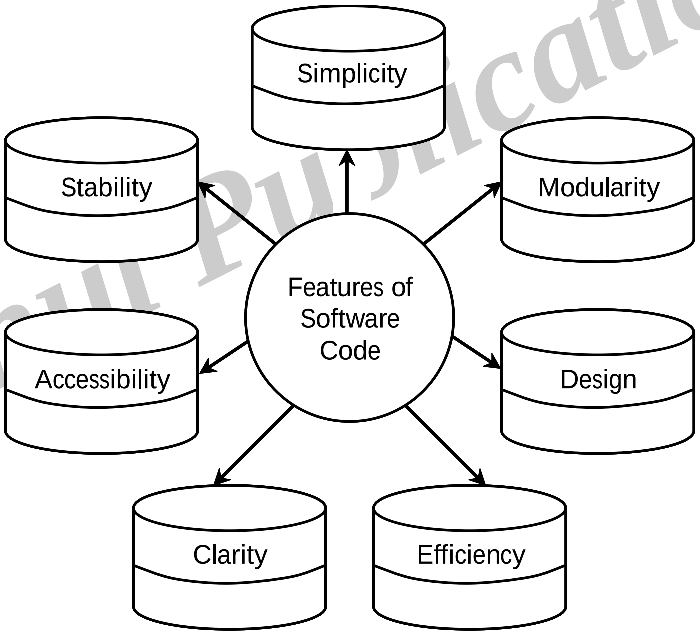
Flowchart of Software Coding

Features

The code written for software should be according to the requirements of the users. A program is said to be good if the software code is flawless or contains minimum errors. For the effective performance of the software, some particular features are required in almost all languages that are used to write the software code. These features are listed below.

- **Implicitly:** Software code should be written in a simple and concise manner. Simplicity should be maintained in the organization, implementation, and design of the software code.
- **Modularity:** Breaking the software into several modules not only makes it easy to understand but also easy to debug. With the modularity feature, the same code segment can be reused in one or more software programs.

- **Design:** Software code is properly designed if it is presented in a proper manner. The design of the software should be decided before beginning to write the software code. Writing the software code in a specific, consistent style helps other software developers in reviewing it.
- **Efficiency:** A program is said to be efficient if it makes optimal use of the available resources.
- **Clarity:** Software codes should be clear so that developers are able to understand the program without any complexity. Clarity can be achieved by using features such as simplicity, readability and modularity. Note that clarity comprises clarity of code, clarity of design, and clarity of purpose so that one knows what occurs at each level in the software program.
- **Accessibility:** Software codes should be written in a way that the software components (for example, files and functions) are easily available and accessible. For the files and functions to be accessible, they should have meaningful names as well as supporting captions and text for each image. Similarly, there should be hyperlinks and navigation aids to assist the users in searching information from different sections of the software code.
- **Stability:** Software codes are said to be stable if they are able to work correctly on different platforms without affecting their layout and consistency. To check for stability, software codes should be tested for errors and inconsistency.



4.1.2 Guidelines

Q3. What are the Guidelines of Coding?

(OR)

Describe the Coding Guidelines in Software Engineering.

Ans :

Writing an efficient software code requires a thorough knowledge of programming. This knowledge can be implemented by following a coding style which comprises several guidelines that help in writing the software code efficiently and with minimum errors. These guidelines, known as coding guidelines, are used to implement individual programming language constructs, comments, formatting, and so on. These guidelines, if followed, help in preventing errors, controlling the complexity of the program, and increasing the readability and understandability of the program.

A set of comprehensive coding guidelines encompasses all aspects of code development. To ensure that all developers work in a harmonized manner (the source code should reflect a harmonized style as a single developer had written the entire code in one session), the developers should be aware of the coding guidelines before starting a software project. Moreover, coding guidelines should state how to deal with the existing code when the software incorporates it or when maintenance is performed.

Since there are numerous programming languages for writing software codes, each having different features and capabilities, coding style guidelines differ from one language to another. However, there are some basic guidelines which are followed in all programming languages. These include naming conventions, commenting conventions, and formatting conventions.

There are certain rules for naming variables, functions and methods in the software code. These naming conventions help software; developers in understanding the use of a particular variable or function. The guidelines used to assign a name to any variable, function, and method are listed below.

All the variables, functions, and methods should be assigned names that make the code more understandable to the reader. By using meaningful names, the code can be self-explanatory, thus, minimizing the effort of writing comments for variables. For example, if two variables are required to refer to 'sales tax' and 'income tax', they should be assigned names such as 'sales Tax' and 'income Tax'.

- For names, a full description in a commonly spoken language (for example, English) should be used. In addition, the use of abbreviations should be avoided. For example, variable names like 'contact Number' and 'address' should be used instead of 'cno' and 'add'.
- Short and clear names should be assigned in place of long names. For 'example, 'multiply The Two Numbers' can be shortened to 'multiply Numbers' as it is clear and short enough to be expressed in reasonable length.
- In every programming language, there is a different naming convention for variables and constants in the software code. The commonly used conventions for naming variables and constants are listed in Table.

(i) Variable Naming Conventions

- The variable names should be in camel case letters starting with a lower case letter. For example, use 'total Amount' instead of 'Total Amount'.
- The temporary storage variables that are restricted to a segment of code should be short. For example, the variable 'temp' can be used for a temporary variable. It is important to note that a single temporary variable should not be reused in the same program. For example, variables 'i', j', or 'k' are declared while using loops.
- The use of numbers in naming variables should be avoided. For example, 'first Number' should be used instead of 'number1'.

(ii) Constant Naming Conventions

- All the names of constants should be in upper case. In case the name of constant is too long, it should be separated by an underscore. For example, sales tax rate should be written as 'SALES_TAX_RATE'.
- The use of literal should be avoided. Literal numbers such as '15'used in the software code confuses the reader. These numbers are counted as integers and result in wrong output of the program. However, the numbers '0' and '1' can be used as constants.

As with variables and constants, there are some guidelines that should be followed while naming functions in the software code. These conventions are listed below.

- The names of functions should be meaningful and should describe the purpose of the function with clarity and briefness. Like variables, the names should be self-explanatory so that no additional description about the task of that function is required.
- The function name should begin with a verb. For example, the verb 'display' can be used for the function that displays the output on the screen. In case the verb itself is not descriptive, an additional noun or adjective can be used with the verb.
- In case the function returns a Boolean value, the helping verbs 'is' and 'has' should be used as prefixes for the function name. For example, the function name 'is Deposited' or has Deposited' should be used for functions that return true or false values.
- Comments are helpful in proper understanding of the code segment used in program. Commenting conventions should be used efficiently to make the code easy to grasp. Generally, two types of commenting conventions are used: file header comments and trailing comments.
- **File header comments** are useful in providing information related to a file as a whole and comprise identification information such as date of creation, Dame of the creator, and a brief description of the software code.
- **Trailing comments** are used to provide explanation of a single line of code. These comments are used to clarify the complex code. These also specify the function of the abbreviated variable names that are not clear. In some languages, trailing comments are used with the help of a double slash (//).The commenting conventions that are commonly followed in the software code are listed below.
  - o Comments should not be used to include information that is clearly understandable from the software.
  - o Comments should be used with important segments of code and code segments that are difficult to understand.
  - o Comments should be separated from the code to enhance readability of the software code.
  - o Formatting (way of arranging a program in order to enhance readability) consists of indentation, alignment, and use of white spaces in the program. Consistency plays an important role while formatting a program in an organized way. A program with consistent formatting makes the code easier to read and understand. The commonly used formatting conventions are listed below.
- **Indentation:** This refers to one or more spaces left at the beginning of statements in the program. Indentation is useful in making the code easily readable. However, the spaces used for indentation should be followed in the entire program. The guidelines that are commonly followed while indenting a program are listed below. White spaces: These improve readability by minimizing the compactness of the code. Some of the guidelines for proper usage of spaces within the code are listed below.

  Indentation should be used to highlight a nested block. Some nested blocks can be made with the help of 'if-else' and 'do-while' loops.

  - – Indentation is required if the statement is large enough to fit in a single line.
  - – Indentation should be consistent at the beginning and at the end of the braces in the program.
  - – There should be a space after placing a comma between two function arguments.
  - – There should be no space between a function name and parenthesis.
  - – There should be spaces to align the operators vertically to emphasize program structure and semantics.

Q4. Explain the Implementing Coding Guidelines.

Ans :

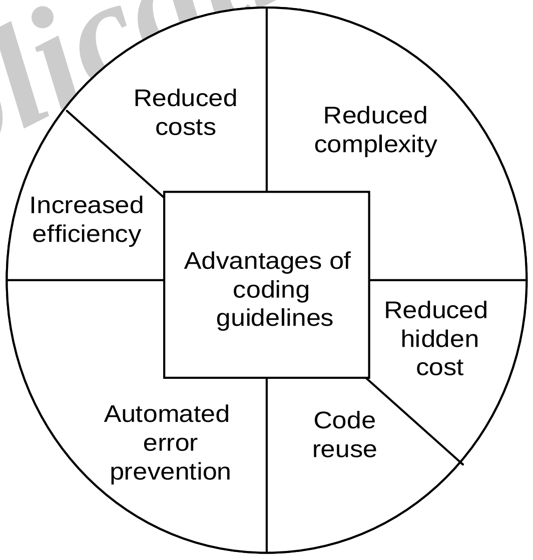
If coding guidelines are used in a proper manner, errors can be detected at the time of writing the software code. Such detection in early stages helps in increasing the performance of the software as well as reducing the additional and unplanned costs of correcting and removing errors. Moreover, if a well-defined coding guideline is applied, the program yields a software system that is easy to comprehend and maintain. Some of the coding guidelines that are followed in a programming language are listed below.

- All the codes should be properly commented before being submitted to the review team.
- All curly braces should start from a new line.
- All class names should start with the abbreviation of each group. For example, AA and CM can be used instead of academic administration and course management, respectively.
- Errors should be mentioned in the following format: [error code]: [explanation]. For example, 0102: null pointer exception, where 0102 indicates the error code and null pointer exception is the name of the error.
- Every 'if statement should be followed by a curly braces even if there exists only a single statement.
- Every file should contain information about the author of the file, modification date, and version information.
- Similarly, some of the commonly used coding guidelines in a database (organized collection of information that is systematically organized for easy access and analysis) are listed below.
- Table names should start with TBL. For example, TBL_STUDENT.
- If table names contain one word, field names should start with the first three characters of the name of the table. For example, STU_FIRSTNAME.
- Every table should have a primary key.
- Long data type (or database equivalent) should be used for the primary key.

Q5. State the Advantages of Coding Guidelines.

Ans : **(Imp.)**

Coding guidelines supplement the language standard by defining acceptable and unacceptable usage of the programming language used. Acceptable usage avoids troublesome situations while unacceptable usage is conducive to errors or leads to misunderstanding of the written code. Properly implemented coding guidelines help the developer to limit program complexity, establish the basis for code review, and guard against compiler and common programming errors. Other advantages associated with coding guidelines are listed below and depicted.



(i) Increased Efficiency: Coding guidelines can be used effectively to save time spent on gathering unnecessary details. These guidelines increase the efficiency of the software team while the software development phase is carried out. An efficient software code is fast and economical. Software coding guidelines are used to increase efficiency by making the team productive, thus, ensuring that the software is delivered to the user on time.

(ii) **Reduced Costs:** Coding guidelines are beneficial in reducing the cost incurred on the software project. This is possible since coding guidelines help in detecting errors in the early stages of the software development. Note that if errors are discovered after the software is delivered to the user, the process of rectifying them becomes expensive as additional costs are incurred on late detection, rework, and retesting of the entire software code.

(iii) **Reduced Complexity:** The written software code can be either simple or complex. Generally, it is observed that a complex segment of software code is more susceptible to errors than a segment containing a simple software code. This is because a complex software code reduces readability as well as understandability. In addition, the complex software code may be inefficient in functioning and use of resources. However, if the code is written using the given coding guidelines, the problem of complexity can be significantly avoided as the probability of error occurrence reduces substantially.

(iv) **Reduced Hidden Costs:** Coding guidelines, if adhered to in a proper manner, help to achieve a high-quality software code. The software quality determines the efficiency of the software. Software quality is the degree to which user requirements are accomplished in the software along with conformity to standards. Note that if quality is not considered while developing the software, the cost for activities such as fixing errors, redesigning the software, and providing technical support increases considerably.

(v) **Code Reuse:** Using coding guidelines, software developers are able to write a code that is more robust and create individual modules of the software code. The reason for making separate code segment is to enable reusability of the modules used in the software. A reusable module can be used a number of times in different modules in one or more software.

(vi) **Automated Error Prevention:** The coding guidelines enable Automated Error Prevention (AEP). This assures that each time error occurs in software, the software development activity is improved to prevent similar errors in future. AEP begins with detecting errors in the software, isolating its cause, and then searching the cause of error generation. Coding guidelines are useful in preventing errors as they allow implementation of requirements that prevent the most common and damaging errors in the software code.

In addition to the above mentioned advantages, coding guidelines define appropriate metric thresholds. These thresholds help in reducing complexity, thus, minimizing the occurrence of errors. Software developers face increasing demands to demonstrate that development practices meet the accepted coding guidelines. This is essential for companies developing safety-critical software as well as those seeking CMM and ISO certification.

4.2 Unit Testing

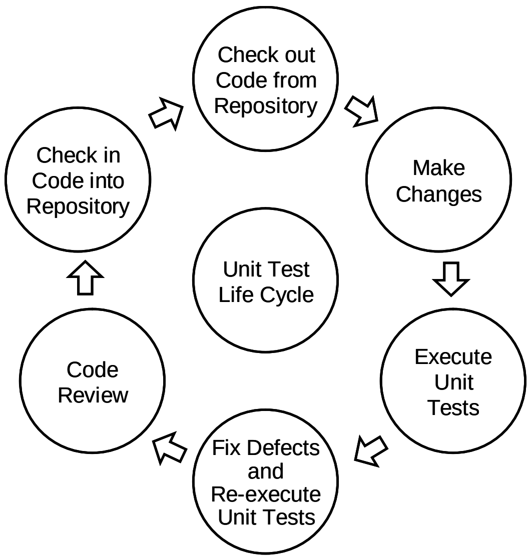
Q6. What is Unit Testing? State the benefits of unit tests.

Ans :

As a client, unit testing is not something you will see in an end product it's a type of testing performed during the coding stage. You can always specify that a potential developer you hire uses unit testing during the development phase to ensure your product has reduced logical bugs.

Without getting too technical, unit testing is a form of coding that breaks your software down into specific functions then tests each individually for any logic flaws. The goal is to find bugs before your customers find them.

A unit test would find any logic flaws (e.g., empty values that throw errors, input that throws errors, or any inappropriate values) that your human testers may not think to enter. From this type of testing, developers can have the confidence that their code is functional and, if there are any errors, that they can be quickly found. This type of assurance can be crucial when rolling out large enterprise applications, and can ultimately reduce costs associated with buggy software.


Check out Code from Repository
Make Changes
Execute Unit Tests
Fix Defects and Re-execute Unit Tests
Code Review
Check in Code into Repository
Unit Test Life Cycle

In many scenarios, the first thing a developer cuts out when a deadline is looming is unit tests. This is because unit tests are separate programs that are written to test for any logic errors and functions, and this code has to be maintained throughout the lifecycle of your software project. But cutting out this part of the testing process can lead to big problems down the line.

There are several applications on the market that plug into the coder's code and help them determine if any bugs exist for any specific function.

Benefits

1. Unit testing increases confidence in changing/ maintaining code. If good unit tests are written and if they are run every time any code is changed, we will be able to promptly catch any defects introduced due to the change. Also, if codes are already made less interdependent to make unit testing possible, the unintended impact of changes to any code is less.

2. Codes are more reusable. In order to make unit testing possible, codes need to be modular. This means that codes are easier to reuse.

3. Development is faster. How? If you do not have unit testing in place, you write your code and perform that fuzzy 'developer test' (You set some breakpoints, fire up the GUI, provide a few inputs that hopefully hit your code and hope that you are all set.) If you have unit testing in place, you write the test, write the code and run the test. Writing tests takes time but the time is compensated by the less amount of time it takes to run the tests; You need not fire up the GUI and provide all those inputs. And, of course, unit tests are more reliable than 'developer tests'. Development is faster in the long run too. How? The effort required to find and fix defects found during unit testing is very less in comparison to the effort required to fix defects found during system testing or acceptance testing.

4. The cost of fixing a defect detected during unit testing is lesser in comparison to that of defects detected at higher levels. Compare the cost (time, effort, destruction, humiliation) of a defect detected during acceptance testing or when the software is live.

5. Debugging is easy. When a test fails, only the latest changes need to be debugged. With testing at higher levels, changes made over the span of several days/ weeks/ months need to be scanned.

6. Codes are more reliable. Why? I think there is no need to explain this to a sane person.

4.3 Code Metrics Values

Q7. What are the metrics of Coding?

Ans : **(Imp.)**

Code metrics is a set of software measures that provide developers better insight into the code they are developing. By taking advantage of code metrics, developers can understand which types and/ or methods should be reworked or more thoroughly tested. Development teams can identify potential risks, understand the current state of a project, and track progress during software development.

Software Measurements

1. Maintainability Index

Calculates an index value between 0 and 100 that represents the relative ease of maintaining the code. A high value means better maintainability. Color coded ratings can be used to quickly identify trouble spots in your code. A green rating is between 20 and 100 and indicates that the code has good maintainability. A yellow rating is between 10 and 19 and indicates that the code is moderately maintainable. A red rating is a rating between 0 and 9 and indicates low maintainability.

2. Cyclomatic Complexity

Measures the structural complexity of the code. It is created by calculating the number of different code paths in the flow of the program. A program that has complex control flow will require more tests to achieve good code coverage and will be less maintainable.

3. Depth of Inheritance

Indicates the number of class definitions that extend to the root of the class hierarchy. The deeper the hierarchy the more difficult it might be to understand where particular methods and fields are defined or/and redefined.

4. Class Coupling

Measures the coupling to unique classes through parameters, local variables, return types, method calls, generic or template instantiations, base classes, interface implementations, fields defined on external types, and attribute decoration. Good software design dictates that types and methods should have high cohesion and low coupling. High coupling indicates a design that is difficult to reuse and maintain because of its many interdependencies on other types.

5. Lines of Code

Indicates the approximate number of lines in the code. The count is based on the IL code and is therefore not the exact number of lines in the source code file. A very high count might indicate that a type or method is trying to do too much work and should be split up. It might also indicate that the type or method might be hard to maintain.

Code Metrics Values

- The new home for Visual Studio documentation is Visual Studio 2017 Documentation on docs.microsoft.com.
- The latest version of this topic can be found at Code Metrics Values.
- Code metrics is a set of software measures that provide developers better insight into the code they are developing. By taking advantage of code metrics, developers can understand which types and/or methods should be reworked or more thoroughly tested. Development teams can identify potential risks, understand the current state of a project, and track progress during software development.

Software Measurements

The following list shows the code metrics results that Visual Studio calculates:

1. Maintainability Index

Calculates an index value between 0 and 100 that represents the relative ease of maintaining the code. A high value means better maintainability. Color coded ratings can be used to quickly identify trouble spots in your code. A green rating is between 20 and 100 and indicates that the code has good maintainability. A yellow rating is between 10 and 19 and indicates that the code is moderately maintainable. A red rating is a rating between 0 and 9 and indicates low maintainability.

2. Cyclomatic Complexity

Measures the structural complexity of the code. It is created by calculating the number of different code paths in the flow of the program. A program that has complex control flow will require more tests to achieve good code coverage and will be less maintainable.

3. Depth of Inheritance

Indicates the number of class definitions that extend to the root of the class hierarchy. The deeper the hierarchy the more difficult it might be to understand where particular methods and fields are defined or/and redefined.

4. Class Coupling

Measures the coupling to unique classes through parameters, local variables, return types, method calls, generic or template instantiations, base classes, interface implementations, fields defined on external types, and attribute decoration. Good software design dictates that types and methods should have high cohesion and low coupling. High coupling indicates a design that is difficult to reuse and maintain because of its many interdependencies on other types.

5. Lines of Code

Indicates the approximate number of lines in the code. The count is based on the IL code and is therefore not the exact number of lines in the source code file. A very high count might indicate that a type or method is trying to do too much work and should be split up. It might also indicate that the type or method might be hard to maintain.

6 Anonymous Methods

An anonymous method is just a method that has no name. Anonymous methods are most frequently used to pass a code block as a delegate parameter. Metrics results for an anonymous method that is declared in a member, such as a method or access, are associated with the member that declares the method. They are not associated with the member that calls the method.

Generated Code

Some software tools and compilers generate code that is added to a project and that the project developer either does not see or should not change. Mostly, Code Metrics ignores generated code when it calculates the metrics values. This enables the metrics values to reflect what the developer can see and change.

Code generated for Windows forms is not ignored, because it is code that the developer can see and change.

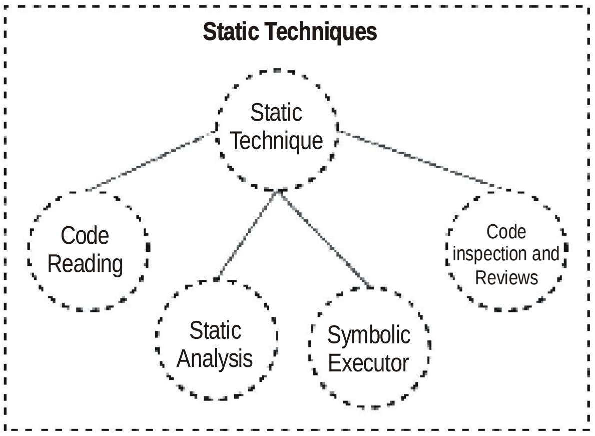
4.4 Code Validation or Code Verification Techniques in Software Engineering

Q8. Explain in detail the Code verification Techniques in Software Engineering.

Ans : **(Imp.)**

Code verification is the process used for checking the software code for errors introduced in the coding phase. The objective of code verification process is to check the software code in all aspects. This process includes checking the consistency of user requirements with the design phase. Note that code verification process does not concentrate on proving the correctness of programs. Instead, it verifies whether the software code has been translated according to the requirements of the user.

The code verification techniques are classified into two categories, namely, dynamic and static. The dynamic technique is performed by executing some test data. The outputs of the program are tested to find errors in the software code. This technique follows the conventional approach for testing the software code. In the static technique, the program is executed conceptually and without any data. In other words, the static technique does not use any traditional approach as used in the dynamic technique. Some of the commonly used static techniques are code reading, static analysis, symbolic execution, and code inspection and reviews.



I) Code Reading

Code reading is a technique that concentrates on how to read and understand a computer program. It is essential for a software developer to know code reading. The process of reading a software program in order to understand it is known as code reading or program reading. In this process, attempts are made to understand the documents, software specifications, or software designs. The purpose of reading programs is to determine the correctness and consistency of the code. In addition, code reading is performed to enhance the software code without entirely changing the program or with minimal disruption in the current functionality of' the program. Code reading also aims at inspecting the code and removing (fixing) errors from it.

Code reading is a passive process and needs concentration. An effective code reading activity primarily focuses on reviewing 'what is important'. The general conventions that can be followed while reading the software code are listed below.

(i) Figure out what is important: While reading the code, emphasis should be on finding graphical techniques (bold, italics) or positions (beginning or end of the section). Important comments may be highlighted in the introduction or at the end of the software code. The level of details should be according to the requirements of the software code.

(ii) Read what is important: Code reading should be done with the intent to check syntax and structure such as brackets, nested loops, and functions rather than the non-essentials such as name of the software developer who has written the software code.

II) Static Analysis

Static analysis comprises a set of methods used to analyze the source code or object code of the software to understand how the software functions and to set up criteria to check its correctness. Static analysis studies the source code without executing it and gives information about the structure of model used, data and control flows, syntactical accuracy, and much more. Due to this, there are several kinds of static analysis methods, which are listed below.

➢ **Control Flow Analysis**

This examines the control structures (sequence, selection, and repetition) used in the code. It identifies incorrect and inefficient constructs and also reports unreachable code, that is, the code to which the control never reaches.

➢ **Data Analysis**

This ensures that-proper operations are applied to data objects (for example, data structures and linked lists). In addition, this method also ensures that the defined data is properly used. Data analysis comprises two methods, namely, data dependency and data-flow analysis. Data dependency (which determines the dependency of one variable on another) is essential for assessing the accuracy of synchronization across multiple processors. Dataflow analysis checks the definition and references of variables.

➢ **Fault/failure Analysis**

This analyzes the fault (incorrect model component) and failure (incorrect behavior of a model component) in the model. This method uses input-output transformation descriptions to identify the conditions that are the cause for the failure. To determine the failures in certain conditions, the model design specification is checked.

➢ **Interface Analysis**

This verifies and validates the interactive and distributive simulations to check the software code. There are two basic techniques for the interface analysis, namely, model interface analysis and user interface analysis. Model interface analysis examines the sub-model interfaces and determines the accuracy of the interface structure. User interface analysis examines the user interface model and checks for precautionary steps taken to prevent errors during the user's interaction with the model'. This method also concentrates on how accurately the interface is integrated into. the overall model and simulation.

III) Symbolic Execution

Symbolic execution concentrates on assessing the accuracy of the model by using symbolic values instead of actual data values for input. Symbolic execution, also known as symbolic evaluation, is performed by providing symbolic inputs, which produce expressions for the output.

Symbolic execution uses a standard mathematical technique for representing the arbitrary program inputs (variables) in the form of symbols. To perform the calculation, a machine is employed to perform algebraic manipulation on the symbolic expressions. These expressions include symbolic data meant for execution. The symbolic execution is represented as a symbolic state symbol consisting of variable symbolic values, path, and the path conditions. The symbolic state for each step in the arbitrary input is updated. The steps that are commonly followed for updating the symbolic state considering all possible paths are listed below.

(i) The read or the input symbol is created.

(ii) The assignment creates a symbolic value expression.

(iii) The conditions in symbolic state add constraints to the path condition.

The output of symbolic execution is represented in the form of a symbolic execution tree. The branches of the tree represent the paths of the model. There is a decision point to represent the nodes of the tree. This node is labeled with the symbolic values of the data at that junction. The leaves of the tree are complete paths through the model and they represent the output of symbolic execution. Symbolic execution helps in showing the correctness of the paths for all computations. Note that in this method the symbolic execution tree increases in size and creates complexity with growth in the model.

IV) Code Inspection and Reviews

This technique is a formal and systematic examination of the source code to detect errors. During this process, the software is presented to the project managers and the users for a comment of approval. Before providing any comment, the inspection team checks the source code for errors. Generally, this team consists of the following.

(i) **Moderator:** Conducts inspection meetings, checks errors-, and ensures that the inspection process is followed.

(ii) **Reader:** Paraphrases the operation of the software code.

(iii) **Recorder:** Keeps record of each error in the software code. This frees the task of other team members to think deeply about the software code.

(iv) **Author:** Observes the code inspection process silently and helps only when explicitly required. The role of the author is to understand the errors found in the software code.

As mentioned above, the reader paraphrases the meaning of small sections of code during the code inspection process. In other words, the reader translates the sections of code from a computer language to a commonly spoken language (such as English). The inspection process is carried out to check whether the implementation of the software code is done according to the user requirements. Generally, to conduct code inspections the following steps are performed.

(i) **Planning:** After the code is compiled and there are no more errors and warning messages in the software code, the author submits the findings to the moderator who is responsible for forming the inspection team. After the inspection team is formed, the moderator distributes the listings as well as other related documents like design documentation to each team member. The moderator plans the inspection meetings and coordinates with the team members.

(ii) **Overview:** This is an optional step and is required only when the inspection team members are not aware of the functioning of the project. To familiarize the team members, the author provides details to make them understand the code.

(iii) Preparation: Each inspection team member individually examines the code and its related materials. They use a checklist to ensure that each problem area is checked. Each inspection team member keeps a copy of this checklist, in which all the problematic areas are mentioned.

(iv) Inspection Meeting: This is carried out with all team members to review the software code. The moderator discusses the code under review with the inspection team members.

There are two checklists for recording the result of the code inspection, namely, code inspection checklist and inspection error list. The code inspection checklist contains a summary of all the errors of different types found in the software code. This checklist is used to understand the effectiveness of inspection process. The inspection error list provides the details of each error that requires rework. Note that this list contains details only of those errors that require the whole coding process to be repeated.

All errors in the checklist are classified as major or minor. An error is said to be major if it results in problems and later comes to the knowledge of the user. On the other hand, minor errors are spelling errors and non-compliance with standards. The classification of errors is useful when the software is to be delivered to the user and there is little time to review all the errors present in the software code.

At the conclusion of the inspection meeting, it is decided whether the code should be accepted in the current form or sent back for rework. In case the software code needs reworking, the author makes all the suggested corrections and then compiles the code. When the code becomes error-free, it is sent back to the moderator. The moderator checks the code that has been reworked. If the moderator is completely satisfied with the software code, inspection becomes formally complete and the process of testing the software code begins.

4.5 Software Testing Techniques

Q9. Explain the concept of Software Testing.

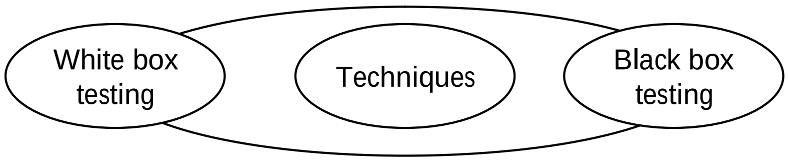
Ans :

Once the software is developed it should be tested in a proper manner before the system is delivered to the user. For this, two techniques that provide systematic guidance for designing tests are used. These techniques are discussed here.

1. Once the internal working of software is known, tests are performed to ensure that all internal operations of the software are performed according to specifications. This is referred to as white box testing.

2. Once the internal working of software is known, tests are performed to ensure that all internal operations of the software are performed according to specifications. This is referred to as white box testing.

3. Once the specified function for which the software has been designed is known, tests are performed to ensure that each function is working properly. This is referred to as black box testing.



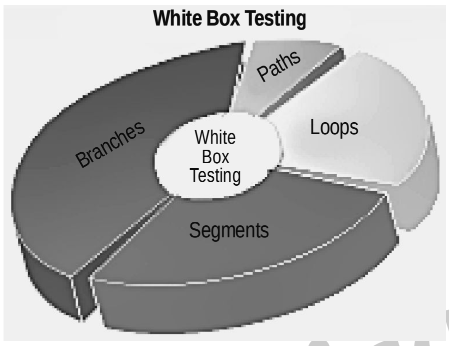
Testing Techniques

4.5.1 White Box Testing

Q10. Describe white box testing in software engineering.

Ans : **(Imp.)**

White box testing (also called structural testing or glass box testing) is performed to test the program internal structure. To perform white box testing, the tester should have a thorough knowledge of the program internals along with the purpose of developing the software. During this testing, the entire software implementation is also included with the specification. This helps in detecting errors even with unclear or incomplete software specification.



The goal of white box testing is to ensure that the test cases (developed by software testers by using white box testing) exercise each path through a program. That is, test cases ensure that all internal structures in the program are developed according to design specifications. The test cases also ensure the following.

1. All independent paths within the program have been exercised at least once.
2. All internal data structures have been exercised.
3. All loops (simple loops, concatenated loops, and nested loops) have been executed at and within their specified boundaries.
4. All segments present between the control structures (like ‘switch’ statement) have been executed at least once.
5. Each branch (like ‘case’ statement) has been exercised at least once.
6. All the logical conditions as well as their combinations have been executed at least once for both true and false paths.
7. Various advantages and disadvantages of white box testing are listed in Table.

Advantages

Covers the larger part of the program code while testing.

1. Uncovers typographical errors.
2. Detects design errors that occur when incorrect assumptions are made about execution paths.

Disadvantages

Tests that cover most of the program code may not be good for assessing the functionality of surprise (unexpected) behaviors and other testing goals.

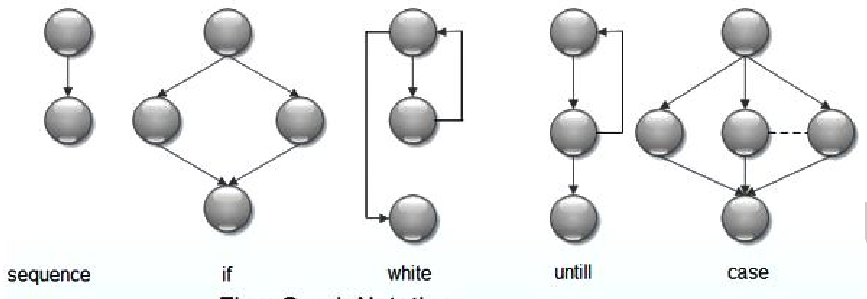
1. Tests based on design may miss other system problems.
2. Tests cases need to be changed if implementation changes.

The effectiveness of white box testing is usually measured in terms bf test or code coverage metrics, that is, the fraction of code exercised by test cases. Various types of testing, which occur as part of white box testing are basis path testing, control structure testing, and mutation testing.

Basis path testing enables to generate test cases such that every path of the program has been exercised at least once. This technique is used to specify the basic set of execution paths that are required to execute all the statements present in the program. Note that with the increase in the size of the software the number of execution paths also increase, thereby degrading the effectiveness of basis path testing.

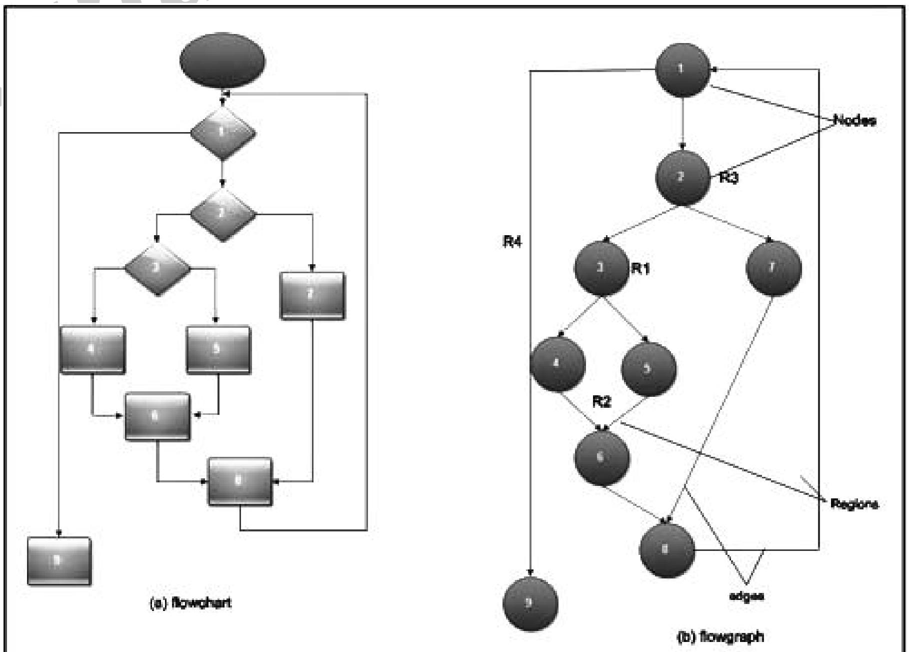
Creating Flow Graph

A flow graph represents the logical control flow within a program. For this, it makes use of a notation.



Flow Graph Notation

A flow graph uses different symbols, namely, circles and arrows to represent various statements and flow of control within the program. Circles represent nodes, which are used to depict the procedural statements present in the program. A sequence of process boxes and a decision box used in a flowchart can be easily mapped into a single node. Arrows represent edges or links, which are used to depict the flow of control within the program. It is necessary far every edge to end in a node irrespective of whether it represents a procedural statement. In a flaw graph, the area bounded by edges and nodes is known as a region. In addition, the area outside the graph is also counted as a region while counting regions. A flow graph can be easily understood with the help of a diagram.



Creating Flowgraph

Note that a node that contains a condition is known as predicated node, which contains one or more edges emerging out of it.

Finding Independent Paths

A path through the program, which specifies a new condition or a minimum of one new set of processing statements, is known as an independent path. For example, in nested 'if' statements there are several conditions that represent independent paths. Note that a set of all independent paths within a program is known as its basis set.

A test case is developed to ensure that while testing all statements of the program get exercised at least once.

P1: 1-9

P2: 1-2-7-8-1-9

P3: 1-2-3-4-6-8-1-9

P4: 1-2-3-5-6-8-1-9

Where P1, P2, P3, and P4 represent different independent paths present in the program.

To determine the number of independent paths through a program, the cyclomatic complexity metric is used that provides a quantitative measure of the logical complexity of a program. The value of this metric defines the number of test cases that should be developed to' ensure that all statements in the program get exercised at least once during testing.

Cyclomatic complexity of a program can be computed using any of the following three methods.

➢ By counting the total number of regions in the flow graph of a program. For example, there are four regions represented by R1, R2, R3, and R4; hence, the cyclomatic complexity is four.

➢ By using the following formula.

CC = E - N + 2 Where

CC = the cyclomatic complexity of the program

E = the number of edges in the flaw graph

N = the number of nodes in the flaw graph.

➢ For example, E = 11, N = 9.

Therefore, CC = 11 - 9 + 2 = 4.

➢ By using the following formula.

CC= P + 1

Where

P= the number of predicate nodes in the flow graph.

For example, P = 3.

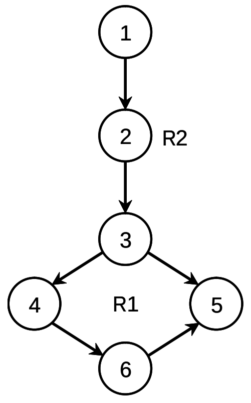
Therefore, CC = 3 + 1 = 4.

Deriving Test Cases

In this, basis path testing is presented as a series of steps and test cases are developed to ensure that all statements within the program get exercised at least once while performing testing. While performing basis path testing, initially the basis set (independent paths in the program) is derived. The basis set can be derived using the steps listed below.

1. Draw the Flow Graph of the Program

A flow graph is constructed using symbols previously discussed. For example, a program to find the greater of two numbers is given below.



For graph to Find the Greater between Two Number

```
procedure greater;
integer: x, y, z = 0;
enter the value of x;
enter the value of y;
if x > y then
z = x;
else
z = y;
end greater
```

2. Compute the cyclomatic complexity

The cyclomatic complexity of the program can be computed using the flow graph depicted.

CC = 2 as there two regions R1 and R2 or

CC 6 edges - 6 nodes + 2 =2 or

CC 1 predicate node + 1 = 2.

3. Determine all independent paths through the program

For the flow graph depicted the independent paths are listed below.

P1: 1-2-3-4-6

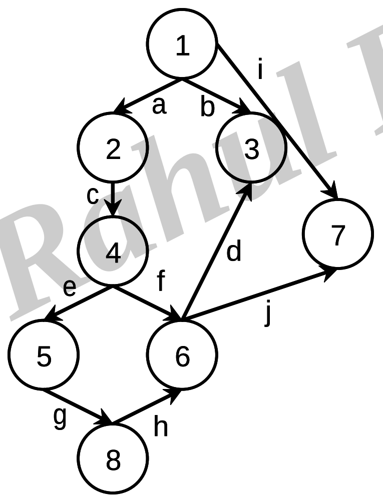
P2: 1-2-3-5-6

4. Prepare test cases:

Test cases are prepared to implement the execution of all independent paths in the basis set. The program is then tested for each test case and the produced output is compared with the desired output.

Graph matrix is used to develop software tool that in turn helps in carrying 'out basis path testing. It is defined as a data structure used to represent the flow graph of a program in a tabular form. This matrix is also used to evaluate the control structures present in the program during testing.

Graph matrix is a square matrix of the size NxN, where Nis the number of nodes in the flow graph. An entry is made in the matrix at the intersection of ith row and jth column if there exists an edge between ith and jth node in the flow graph. Every entry in the graph matrix is assigned some value known as link weight. Adding link weights to each entry makes the graph matrix a useful tool for evaluating the control structure of the program during testing.



(a) Flow graph

	1	2	3	4	5	6	7	8
1		a	b				i	
2				c				
3								
4					e	f		
5								g
6			d				j	
7								
8						h		

(b) Graph matrix

In the flow graph, numbers and letters are used to identify each node and edge respectively. A letter entry is made if there is an edge between two nodes of the flow graph. For example, node 3 is connected to the node 6 by edge d and node 4 is connected to node 2 by edge c, and so on.

Control Structure Testing

Control structure testing is used to enhance the coverage area by testing various control structures (which include logical structures and loops) present in the program. Note that basis path testing is used as one of the techniques for control structure testing. Various types of testing performed under control structure testing are condition testing, data-flow testing, and loop testing.

Condition Testing

In condition testing, the test cases are derived to determine whether the logical conditions and decision statements are free from errors. The errors presenting logical conditions can be incorrect Boolean operators, missing parenthesis in a Boolean expression, error in relational operators, arithmetic expressions, and so on.

The common types of logical conditions that are tested using condition testing are listed below.

1. A relational expression such as E1 op E2, where E1 and E2 are arithmetic expressions and op is an operator.
2. A simple condition such as any relational expression preceded by a NOT (~) operator for example, (~E1), where E1 is an arithmetic expression.
3. A compound condition, which is formed by combining two or more simple conditions using Boolean operators. For example, (E1 & E2) | (E2 & E3), where El, E2, and E3 are arithmetic expressions and & and | represent AND and OR operators respectively.
4. A Boolean expression consisting of operands and a Boolean operator such as AND, OR, or NOT. For example, A|B is a Boolean expression, where A and B are operands and | represents OR operator.

Condition testing is performed using different strategies, namely, branch testing, domain testing, and branch and relational operator testing. Branch testing executes each branch (like 'if statement) present in the module of a program at least once to detect all the errors present in the branch. Domain testing tests relational expressions present in a program. For this, domain testing executes all statements of the program that contain relational expressions. Branch and relational operator testing tests the branches present in the module of a program using condition constraints. For example,

```
if a > 10
then
print big
```

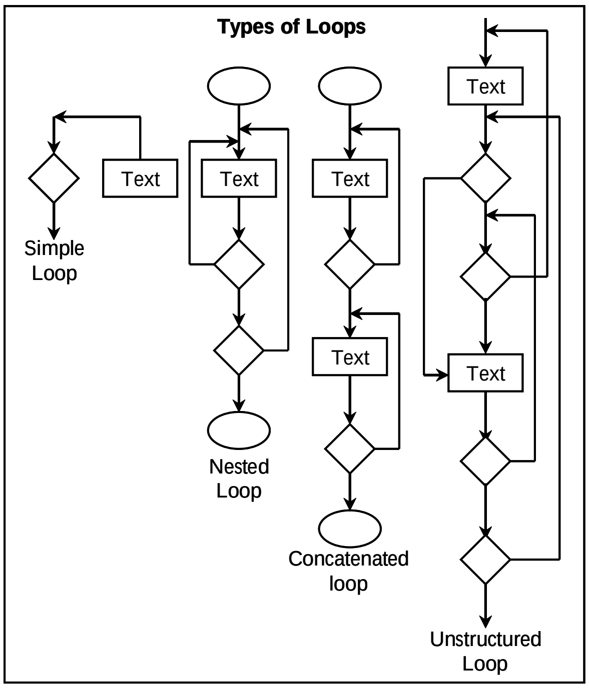
In this case, branch and relational operator testing verifies when the above code is executed, it produces the output 'big' only if the value of variable a is greater than 10.

In data flow, testing, test cases are derived to determine the validity of variables definitions and their uses in the program. This testing ensures that all variables are used properly in a program. To specify test cases, data-flow-based testing uses information such as location at which the variables are defined and used in the program.

For performing data-flow testing, a definition-use graph is built by associating the program variables with nodes and edges of the control flow graph. Once these variables are attached, test cases can easily determine which variable is used in which part of a program and how data is flowing in the program. Thus, data-flow of a program can be tested easily using specified test cases.

Loop Testing

Loop testing is used to check the validity of loops present in the program modules. Generally, there exist four types of loops, namely, simple loop, nested loops, concatenated loops, and unstructured loops.



Simple Loop

Refers to a loop that has no other loops in it. Consider a simple loop of size n. Size n of the loop indicates that the loop can be traversed n times, that is, n number of passes are made through the loop. The steps followed for testing simple loops are listed below.

1. Skip the entire loop.
2. Traverse the loop only once.
3. Traverse the loop two times.
4. Make ill number of passes through the loop, where ill < n.
5. Traverse the loop n - I, n, n + 1 times.

Nested Loops

Loops within loops are known as nested loops. The numbers of tests required for testing nesting loops depends on the level of nesting. More is the level of nesting, more will be the number of tests required. The steps followed for testing nested loops are listed below.

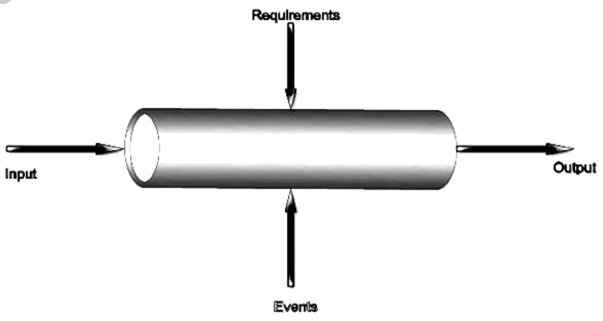
1. Start with the inner loop and set values of all the outer to minimum.
2. Test the inner loop using the steps followed for testing simple loops while keeping the iteration parameters of the outer loops at their minimum values. Add other tests for values that are either out-of-range or are eliminated.
3. Move outwards and perform tests for the next loop while holding other nested loops to 'typical' values and the iteration parameters of the outer loops at their minimum values.
4. Continue performing tests until all the loops are tested.
5. Concatenated loops: The loops containing several loops that may be dependent or independent In case the loops are dependent on each other, the steps in nested loops are followed. On the other hand, if the loops are independent of each other, the steps in simple loops are followed.
6. Unstructured loops: Such loops of difficult to test; therefore, they should be redesigned so that the use of structured programming constructs can be reflected.
7. Mutation testing is a white box method where errors are 'purposely' inserted into a program (under test) to verify whether the existing test case is able to detect the error. In this testing, mutants of the program are created by making some changes in the original program. The objective is to check whether each mutant produces an output that is different from the output produced by the original program.

4.5.2 Black Box Testing

Q11. Explain Black Box Testing in detail.

Ans : **(Imp.)**

Black box (or functional) testing checks the functional requirements and examines the input and output data of these requirements. When black box testing is performed, only the sets of 'legal' input and corresponding outputs should be known to the tester and not the internal logic of the program to produce that output. Hence to determine the functionality, the outputs produced for the given sets of input are observed.



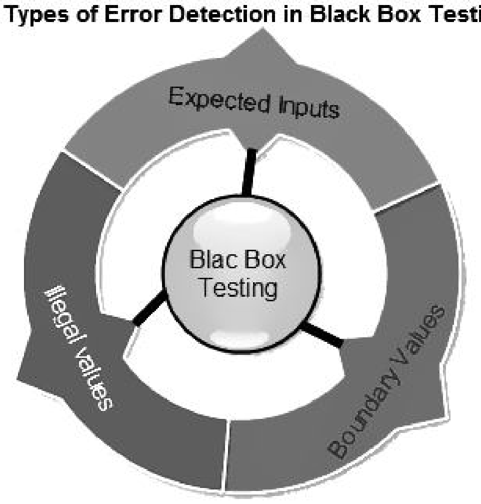
Black Box Testing

The black box testing is used to find the errors listed below.

1. Interface errors such as functions, which are unable to send or receive data to/from other software.
2. Incorrect functions that lead to undesired output when executed.
3. Missing functions and erroneous data structures.
4. Erroneous databases, which lead to incorrect outputs when the software uses the data present in these databases for processing.

5. Incorrect conditions due to which the functions produce incorrect outputs when they are executed.
6. Termination errors such as certain conditions due to which a function enters a loop that forces it to execute indefinitely.

Types of Error Detection in Black Box Testing



In this testing, tester derives various test cases to exercise the functional requirements of the software without considering implementation details of the code. Then, the software is run for the specified sets of input and the outputs produced for each input set is compared against the specifications to conform the correctness. If they are a specified by the user, then the software is considered to be correct else the software is tested for the presence of errors in it. The advantages and disadvantages associated with black box testing are listed in Table.

Advantages

Tester does not require knowledge of internal logic of the program and the programming language used.

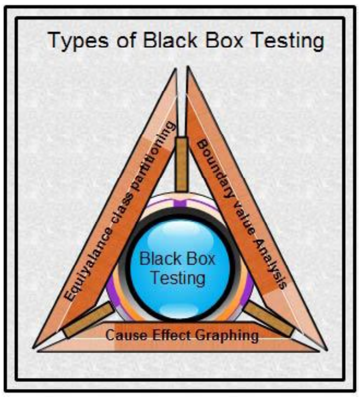
1. Reveals any ambiguities and inconsistencies in the functional specifications.
2. Efficient when used to larger systems.
3. A non-technical person can also perform black box testing.

Disadvantages

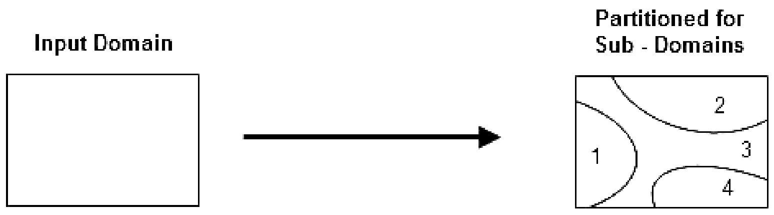
Exercising software for every possible test case requires a lot of time, thus, only a small number of test cases are used to test the functional requirements.

1. As test cases are developed without looking at the internal logic of program, testing may leave many paths in the program unexercised.
2. There may be duplication of test cases if tester is unaware of the test cases that have already been tried by the software developer
3. Cannot be targeted towards particular code segments, hence is more error prone.

Various methods used in black box testing are equivalence class partitioning, boundary value analysis, and cause-effect graphing. In equivalence class partitioning, the test inputs are classified into equivalence classes such that one input checks (validates) all the input values in that class. In boundary value analysis, the boundary values of the equivalence classes are considered and tested. In cause-effect graphing, cause-effect graphs are used to design test cases, which provides all the possible combinations of inputs to the program.



This method tests the validity of outputs by dividing the input domain into different classes of data (known as equivalence classes) using which test cases can be easily generated. Test cases are designed with the purpose of covering each partition at least once. A test case that is able to detect every error in a specified partition is said to be an ideal test case.

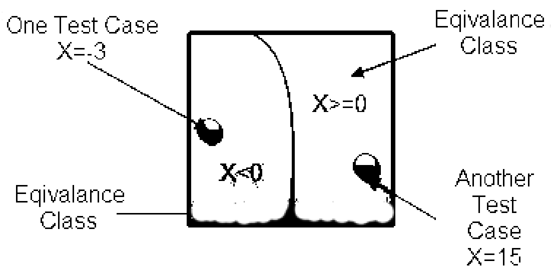


An equivalence class depicts valid or invalid states for the input condition. An input condition can be either a specific numeric value, a range of values, a Boolean condition, or a set of values. The general guidelines that are followed for generating the equivalence classes are listed in Table.

Input Condition	Number of equivalence classes	Description
Boolean	Two	One valid and one invalid
Specific numeric value	Three	One valid and two invalid
Range	Three	One valid and two invalid
Member of a set	Two	One valid and one invalid

To understand equivalence class partitioning properly, let us consider an example. This example is explained in the series of steps listed below.

1. Suppose that a program P takes an integer X as input.
2. Now, either $X < 0$ or $X > 0$.
3. In case $X < 0$, the program is required to perform task T1; otherwise, the task T2 is performed.
4. The input domain is as large as X and it can assume a large number of values. Therefore the input domain (p) is partitioned into two equivalence classes and all test inputs in the $X < 0$ and $X \geq 0$ equivalence classes are considered to be equivalent.
5. Now, as shown in Figure independent test cases are developed for $X < 0$ and $X \geq 0$.



Test Case and Equivalence Class

In boundary value analysis (BVA), test cases are derived on the basis of values that lie on an edge of the equivalence partitions. These values can be input or output values either at the edge or within the permissible range from the edge of an equivalence partition.

BVA is used since it has been observed that most errors occur at the boundary of input domain rather than at the middle of the input domain. Note that boundary value analysis complements the equivalence partitioning method. The only difference is that in BVA, test cases are derived for both input domain and output domain while in equivalence partitioning test cases are derived only for input domain.

Generally, the test cases are developed in boundary value analysis using certain guidelines, which are listed below.

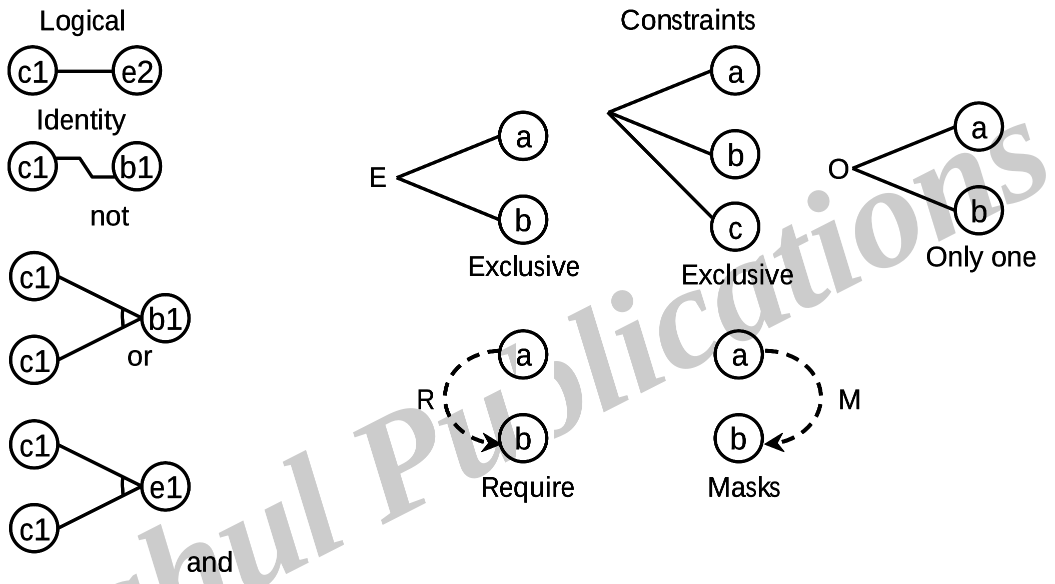
1. If an input condition specifies a range of values, test cases should be developed on the basis of both the values at the boundaries and the values that are just above and below the boundary values. For example, for the range -0.5d”Xd”0.5, the input values for a test case can be ‘-0.4’, -’0.5', ‘0.5’, ‘0.6’.
2. If an input condition specifies a number of values, test cases should be designed to exercise the minimum and maximum numbers as well as values just above and below these numbers.
3. If an input consists of certain data structures (like arrays), then the test case should be able to execute all the values present at the boundaries of the data structures such as the maximum and minimum value of an array.

Equivalence partitioning and boundary value analysis tests each input given to a program independently. It means none .of these consider the case of combinations of inputs, which may produce situations that need to be tested. This drawback is avoided in cause-effect graphing where combinations of inputs are used instead of individual inputs. In this technique, the causes (input conditions) and effects (output conditions) of the system are identified and a graph is created with each condition as the node of the graph. This graph is called cause-effect graph. This graph is then used to derive test cases. To use the cause-effect graphing method, a number of steps are followed, which are listed below.

1. List the cause (input conditions) and effects (outputs) of the program.
2. Create a cause-effect graph.
3. Convert the graph into a decision table.
4. Generate test cases from the decision table rules.

In order to generate test cases, all causes and effects are allocated unique numbers, which are used to identify them. After allocating numbers, the cause due to which a particular effect occurred is determined. Next, the combinations of various conditions that make the effect 'true' are recognized. A condition has two states, 'true' and 'false'. A condition is 'true' if it causes the effect to occur; otherwise, it is 'false'. The conditions are combined using Boolean operators such as 'AND' (&), 'OR' (|), and 'NOT' (~). Finally, a test case is generated for all possible combinations of conditions.

Various symbols are used in the cause-effect graph. The figure depicts various logical associations among causes c_i and effects e_i. The dashed notation in the figure indicates various constraint associations that can be applied to either causes or effects.

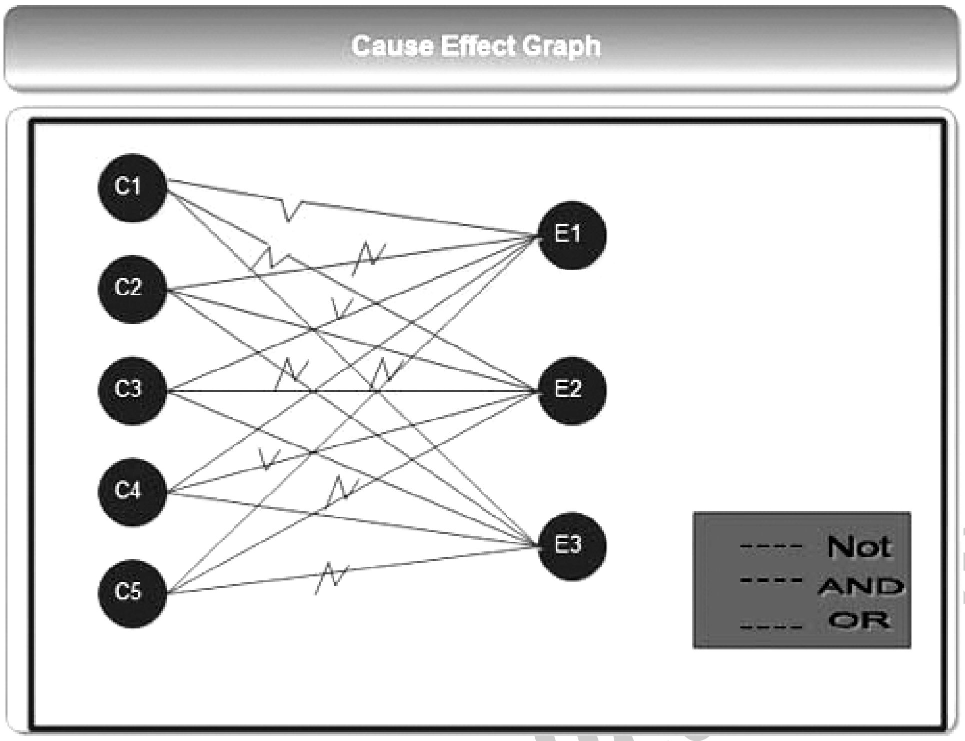


Logical and Constraints Associations

To understand cause-effect graphing properly, let us consider an example. Suppose a triangle is drawn with inputs x, y, and z. The values of these inputs are given between '0' and '100'. Using these inputs, three outputs are produced, namely, isosceles triangle, equilateral triangle or no triangle is made (if values of x, y, z are less than. 60°).

1. Using the steps of cause-effect graphing, initially the causes and effects of the problem are recognized, which are listed in Table.

Causes	Effects
C1: side x is less than the sum of sides y and z. C2: sides x, y, z are equal. C3: side x is equal to side y. C4: side y is equal to side z. C5: side x is equal to side z.	El: no triangle is formed. E2: equilateral triangle is formed. E3: isosceles triangle is formed.



2. A decision table (a table that shows a set of conditions and the actions resulting from them) is drawn as shown in Table.

Conditions					
C1: x < y + z	0	X	X	X	X
C2: x=y=z	X	1	X	X	X
C3: x=y	X	X	1	X	X
C4: y=z	X	X	X	1	X
C5: x=z	X	X	X	X	1
El: not a triangle	1				
E2: equilateral triangle		1			
E3: isosceles triangle			1	1	1

3. Each combination of conditions for an effect in Table is a test case.

4.5.3 White Box Testing Vs Black Box Testing

Q12. What are the differences between White Box and Black Box Testing?

Ans : **(Imp.)**

Although both the testing techniques are used together to test many programs, there are several considerations that make them different from each other. Black box testing detects errors of omission, which are errors occurring due to non-accomplishment of user requirements. On the other hand, white box testing detects errors of commission which are errors occurring due to non-implementation of some part of software code. Other differences between these two techniques are listed in Table.

S.No.	Basis	White Box Testing	Black Box Testing
1	**Purpose**	To test the internal structure of software. Test the software but does not ensure the complete implementation of all the specifications mentioned in user requirements. Addresses flow and control structure of a program.	To test the functionality of software. Concerned with testing the specifications and does not ensure that all the components of software that are implemented are tested. Addresses validity, behavior and performance of software
2	**Stage**	Performed in the early stages of testing.	Performed in the later stages of testing.
3	**Requirement**	Knowledge of the internal structure of a program is required for generating test case.	No knowledge of the internal structure of a program is required to generate test case.
4	**Test Cases**	Test cases are generated on the basis of the internal structure or code of the module to be tested.	Internal structure of modules or programs is not considered for selecting test cases.
5	**Example**	The inner software present inside the calculator (which is known by the developer only) is checked by giving inputs to the code.	-----------

In this testing, it is checked whether the calculator is working properly by giving inputs by pressing the buttons in the calculator.

4.6 Software Testing Metrics

Q13. What are Software Testing Metric?

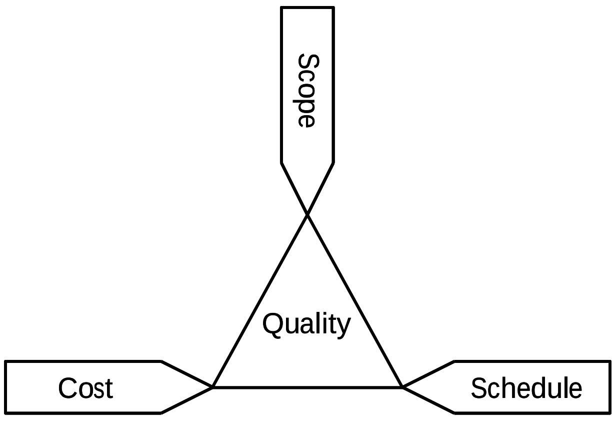
Ans : **(Imp.)**

In software testing, Metric is a quantitative measure of the degree to which a system, system component, or process possesses a given attribute.

In other words, metrics helps estimating the progress, quality and health of a software testing effort. The ideal example to understand metrics would be a weekly mileage of a car compared to its ideal mileage recommended by the manufacturer.

"Software testing metrics - Improves the efficiency and effectiveness of a software testing process."

Software testing metrics or software test measurement is the quantitative indication of extent, capacity, dimension, amount or size of some attribute of a process or product.



Steps to Test Metrics

S.No	Steps to test metrics	Example
1	Identify the key software testing processes to be measured	Testing progress tracking process
2	In this Step, the tester uses the data as baseline to define the metrics	The number of test cases planned to be executed per day
3	Determination of the information to be followed, frequency of tracking and the person responsible	The actual test execution per day will be captured by the test manager at the end of the day
4	Effective calculation, management and interpretation of the defined metrics	The actual test cases executed per day
5	Identify the areas of improvement depending on the interpretation of defined metrics	The Test Case execution falls below the goal set, we need to investigate the reason and suggest the improvement measures

Purpose

"We cannot improve what we cannot measure" and Test Metrics helps us to do exactly the same.

- Take decision for next phase of activities
- Evidence of the claim or prediction
- Understand the type of improvement required
- Take decision or process or technology change

Types of Metrics

(i) Process Metrics: It can be used to improve the process efficiency of the SDLC (Software Development Life Cycle).

(ii) Product Metrics: It deals with the quality of the software product.

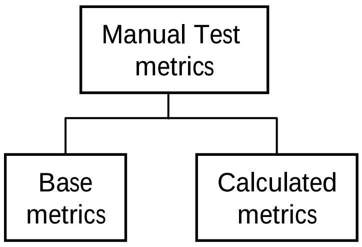
(iii) Project Metrics: It can be used to measure the efficiency of a project team or any testing tools being used by the team members.

Manual Test Metrics

Manual test metrics is classified into two classes

(i) Base Metrics

(ii) Calculated Metrics



(i) Base Metrics

Base metrics is the raw data collected by Test Analyst during the test case development and execution (# of test cases executed, # of test cases).

(ii) Calculated Metrics

While, calculated metrics is derived from the data collected in base metrics. Calculated metrics is usually followed by the test manager for test reporting purpose (% Complete, % Test Coverage).

Depending on the project or business model some of the important metrics are

- Test case execution productivity metrics
- Test case preparation productivity metrics
- Defect metrics
- Defects by priority
- Defects by severity
- Defect slippage ratio

UNIT V

Maintenance and Re-engineering: Software Maintenance, supportability, Reengineering, Business process Reengineering, Software reengineering, Reverse engineering; Restructuring, Forward engineering, Economics of Reengineering.

Software Process Improvement: Introduction, SPI process, CMMI, PCMM, Other SPI Frameworks, SPI return on investment, SPI Trends.

5.1 Software Maintenance, Supportability

Q1. What is Software Maintenance? Describe the types of Software Maintanance.

Ans : **(Imp.)**

Software maintenance is widely accepted part of SDLC now a days. It stands for all the modifications and updations done after the delivery of software product. There are number of reasons, why modifications are required, some of them are briefly mentioned below:

1. **Market Conditions**

   Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping, may trigger need for modification.

2. **Client Requirements**

   Over the time, customer may ask for new features or functions in the software.

3. **Host Modifications**

   If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.

4. **Organization Changes**

   If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software may arise.

Types

In a software lifetime, type of maintenance may vary based on its nature. It may be just a routine maintenance tasks as some bug discovered by some user or it may be a large event in itself based on maintenance size or nature. Following are some types of maintenance based on their characteristics:

1. **Corrective Maintenance**

   This includes modifications and updations done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.

2. **Adaptive Maintenance**

   This includes modifications and updations applied to keep the software product up-to date and tuned to the ever changing world of technology and business environment.

3. **Perfective Maintenance**

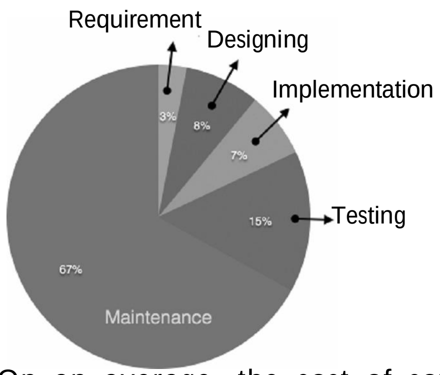
   This includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.

4. **Preventive Maintenance**

   This includes modifications and updations to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

Cost of Maintenance

Reports suggest that the cost of maintenance is high. A study on estimating software maintenance found that the cost of maintenance is as high as 67% of the cost of entire software process cycle.



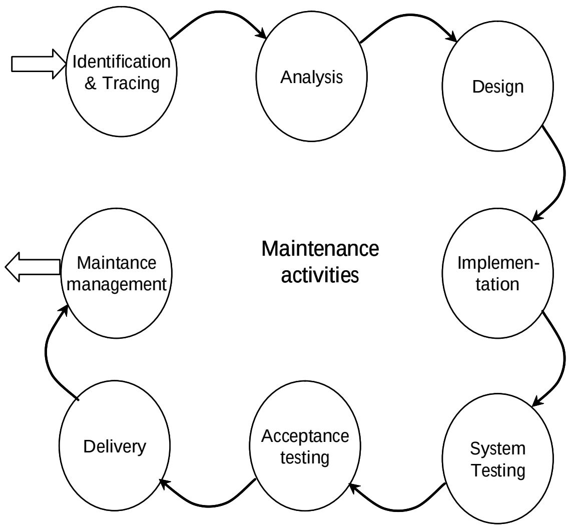
On an average, the cost of software maintenance is more than 50% of all SDLC phases. There are various factors, which trigger maintenance cost go high, such as:

Real-world Factors affecting Maintenance Cost

- The standard age of any software is considered up to 10 to 15 years.
- Older software's, which were meant to work on slow machines with less memory and storage capacity cannot keep themselves challenging against newly coming enhanced software's on modern hardware.
- As technology advances, it becomes costly to maintain old software.
- Most maintenance engineers are newbie and use trial and error method to rectify problem.
- Often, changes made can easily hurt the original structure of the software, making it hard for any subsequent changes.
- Changes are often left undocumented which may cause more conflicts in future.
- Software-end factors affecting Maintenance Cost
- Structure of Software Program
- Programming Language
- Dependence on external environment
- Staff reliability and availability

Maintenance Activities

IEEE provides a framework for sequential maintenance process activities. It can be used in iterative manner and can be extended so that customized items and processes can be included.



These activities go hand-in-hand with each of the following phase:

- **Identification & Tracing :** It involves activities pertaining to identification of requirement of modification or maintenance. It is generated by user or system may itself report via logs or error messages. Here, the maintenance type is classified also.
- **Analysis :** The modification is analyzed for its impact on the system including safety and security implications. If probable impact is severe, alternative solution is looked for. A set of required modifications is then materialized into requirement specifications. The cost of modification/maintenance is analyzed and estimation is concluded.
- **Design :** New modules, which need to be replaced or modified, are designed against requirement specifications set in the previous stage. Test cases are created for validation and verification.
- **Implementation :** The new modules are coded with the help of structured design created in the design step. Every programmer is expected to do unit testing in parallel.
- **System Testing :** Integration testing is done among newly created modules. Integration testing is also carried out between new modules and the system. Finally the system is tested as a whole, following regressive testing procedures.

- **Acceptance Testing :** After testing the system internally, it is tested for acceptance with the help of users. If at this state, user complaints some issues they are addressed or noted to address in next iteration.
- **Delivery** : After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system. The final testing takes place at client end after the software is delivered.

  Training facility is provided if required, in addition to the hard copy of user manual.
- **Maintenance management** : Configuration management is an essential part of system maintenance. It is aided with version control tools to control versions, semi-version or patch management.

5.2 Reengineering

Q2. What is Re-engineering? Describe steps followed by Reengineering.

Ans : **(Imp.)**

Meaning

Reengineering implies changes of various types and depth to a system, from a slight renovation to a total overhaul. Business process reengineering (BPR) began as a private, sector technique to help organizations fundamentally rethink how they do their work in order to dramatically improve customer service, cut operational costs, and become world-class competitors. A key stimulus for reengineering has been the continuing development and deployment of sophisticated information systems and networks.

Steps

There are six standard steps, which are useful to guide a firm in its reengineering procedure,

Step 1: Target Identification

This is the most important aspect of the overall reengineering procedure. It is very essential to identify which work or operation is required to be changed or improved. It is also important to identify the known range of potential improvement i.e. we should know and be aware of limitations or extent of scopes of improvements that can be made.

Step 2: Understand the work Sequence

Understanding the work sequence which is being evaluated is the second step in the reengineering procedure. The traditional gay of doing this is to make a detailed flow chart or process map of the various steps that are required for performing a particular activity. However in reengineering process only those steps that are capable of potential improvement are studied and alternative suggested. In situations where the alternatives suggested require capital commitment like installing of new machinery the return on investment is also taken in account.

Step 3 and 4: The Creative Aspect

A model of the activity which is being studied for improvement is created. This is done to identify best possible alternative design. Simultaneously the firm should also initiate steps to study and analyze external benchmarking in order to find out improved alternative approaches to the design. A final combination of the suggested alternative design of the activity would be a combination of both the internal as well as external perspectives.

Step 5: Evaluation the Modifications to the Activity

The fifth step involves evaluation the modifications to the activity which is being reviewed on the cost benefit basis. During the benchmarking exercise, various ideas would be generated. Care should be taken only to adopt those ideas, which are practical and meaningful. The focus of the evaluation should be on the accurate assessment of the expected benefits that will be accrued from the implementation of the modified activity.

Step 6: Implementation

Depending upon the extent of the proposed change, it may become necessary for the firm to resort to suitable training for its employees. How effective will be the implementation will depend upon the risk involved in adopting and managing the proposed change in the activity.

5.3 Business Process Re-engineering

Q3. What is the Business Process Re-engineering?

Ans :

The globalization of the economy and the liberalization of the trade markets have formulated new conditions in the market place which are characterized by instability and intensive competition in the business environment. Competition is continuously increasing with respect to price, quality and selection, service and promptness of delivery. Removal of barriers, international cooperation, technological innovations cause competition to intensify. All these changes impose the need for organizational transformation, where the entire processes, organization climate and organization structure are changed. Hammer and Champy provide the following definitions. Reengineering is the fundamental rethinking and radical redesign of business processes to achieve dramatic improvements in critical contemporary measures of performance such as cost, quality, service and speed. ¸ Process is a structured, measured set of activities designed to produce a specified output for a particular customer or market. It implies a strong emphasis on how work is done within an organization."

Business processes are characterized by three elements: the inputs, (data such customer inquiries or materials), the processing of the data or materials (which usually go through several stages and may necessary stops that turns out to be time and money consuming), and the outcome (the delivery of the expected result). The problematic part of the process is processing. Business process reengineering mainly intervenes in the processing part, which is reengineered in order to become less time and money consuming.

The term "Business Process Reengineering" has, over the past couple of year, gained Increasing circulation. As a result, many find themselves faced with the prospect of having to learn, plan, implement and successfully conduct a real Business Process Reengineering endeavor, whatever that might entail within their own business organization.

How Business Process Reengineering works:

Business Process Reengineering is a dramatic change initiative that contains five major steps. Managers should:

➢ Refocus company values on customer needs

➢ Redesign core processes, often using information technology to enable improvements

➢ Reorganize a business into cross-functional teams with end-to-end responsibility for a process.

➢ Rethink basic organizational and people issues.

➢ Improve business processes across the organization.

Companies use Business Process Reengineering

Companies use Business Process Reengineering to improve performance substantially on key processes that impact customers. Business Process Reengineering can:

- **Reduce Costs and Cycle Time:** Business Process Reengineering reduces costs and cycle times by eliminating unproductive activities and the employees who perform them. Reorganization by teams decreases the need for management layers, accelerates information flows, and eliminates the errors and rework caused by multiple handoffs.
- **Improve Quality:** Business Process Reengineering improves quality by reducing the fragmentation of work and establishing clear ownership of processes. Workers gain responsibility for their output and can measure their performance based on prompt feedback.

How to Re-Engineer the Corporation?

Information Availability: To fundamentally redesign a process, one must know the details involved. Details from internal and external sources must be captured and provided to the relevant people in the required time duration. This helps them to identify the bottlenecks and work around better ways of reaching the desired end.

Information Sharing

A BPR project is usually facilitated by a cross functional team. Most of the times, teams are spread across different geographic locations. Information needs to be successfully shared amongst various people to ensure the reengineering goes as planned and without hiccups.

Technology as the Solution

The new processes that are developed as a result of BPR initiatives deploy the latest technology to achieve the desired end results. Usually it is e-Commerce, automation or another technology driven solution that is implemented.

Business Process Re-engineering has become a very important buzzword in the BPM lexicon. Many corporations who were late in realizing the power and importance of BPM have to undergo re-engineering initiatives to ensure that they are still relevant to the marketplace. Re-engineering initiatives are however expensive and may require certain downtime. This is the reason they are resented by many corporations.

Q4. What are the benefits of Business Process Reengineering?

Ans :

The following are the benefits of reengineering of a Business Process:

1. By reengineering, an organisation can achieve radical changes in performance (as measured by cost, cycle time, service and quality).
2. It boosts competitiveness in the operations network through simpler, leaner and more productive processes.
3. Reengineering encourages organisations to abandon conventional approaches to problem solving and to "think big" (revolutionary thinking).
4. The slow, cautious process of incremental improvements leaves many organisations unprepared to compete in today's rapidly changing market place. Reengineering helps organisations make noticeable changes in the pace and quality of their response to customer needs (i.e. break-through improvements).
5. Through reengineering, an organisation can be transformed from a rule driven and job centred organisation structure to a marketing organisation structure that focusses directly on the customer.
6. Reengineering often results in radically new organisational designs that can help companies respond better to competitive pressures, increase market share and profitability and improve cycle times, cost ratios and quality (organisational renewal).
7. The major accomplishment of the reengineering effort is the change that occurs in the corporate culture and the basic principles by which departments operate. Workers at all levels are encouraged to make suggestions for improvement and to believe that management will listen to what they have

to say. Reengineering will eventually help the culture in the organisation to evolve from an insular one to one that accepts change and knows how to deal with it.

8. Reengineering has helped create more challenging and more rewarding jobs with broader responsibilities for employees (job redesign).

Q5. What are disadvantages of business process Reengineering?

Ans :

Business process reengineering is a program that systemically breaks down the process a business uses and starts over with new, more efficient methods basically a redesign or a reboot. A business process is a collection of procedures, steps or activities the business uses to get the product from development to the customer. Businesses use BPR for various reasons, including to cut costs and improve overall production, but the program also has its drawbacks.

- **Identifies Waste**

  The aim of BPR is to help businesses pinpoint obsolete steps, items or workers in a business process. For example, the business may discover during reengineering that only two workers can get the job done that four workers were performing. BPR encourages employee input and participation, as the workers who have familiarity with the processes under study can point out flaws and voice ideas for improvement.

- **Requires Investment**

  BPR typically requires an investment, particularly in technology. Outdated methods, such as doing a task by hand, face replacement by computer programs. The programs improve efficiency and reduce errors, but the company must invest in the software and training, a costly option for companies looking to cut expenses immediately. Not all business types benefit from BPR. For example, a manufacturing company may not have the option of redesigning processes without sacrificing safety or product quality.

- **Cuts Costs and Improves Functionality**

  Removing unnecessary steps cuts down on time and confusion among workers. Assigning tasks that multiple workers would typically handle to one worker gives customers a clear point of contact for help or service. Even by investing more money in technology at the start, companies typically save money over time with the redesigned methods. For example, improving or updating electronic components incurs an up-front cost, but saves money over time by eliminating errors due to outdated components.

- **May Lower Worker Morale**

  Some workers may not adapt to the BPR changes, and those assigned new responsibilities can become overwhelmed. Other workers become obsolete if their primary function is eliminated as part of a process overhaul. Management must provide support and guidance during BPR. Failure of the management team to assist workers and set an example during the BPR process may lead to failure, disorganization and staff problems.

5.4 Software Reengineering

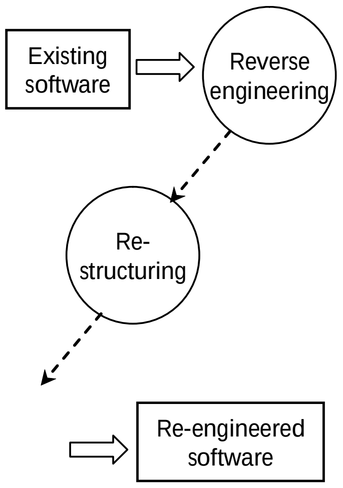
Q6. Explain the concept of Software Re-engineering.

Ans : **(Imp.)**

Legacy software cannot keep tuning with the latest technology available in the market. As the hardware become obsolete, updating of software becomes a headache. Even if software grows old with time, its functionality does not.

For example, initially Unix was developed in assembly language. When language C came into existence, Unix was re-engineered in C, because working in assembly language was difficult.

Other than this, sometimes programmers notice that few parts of software need more maintenance than others and they also need re-engineering.



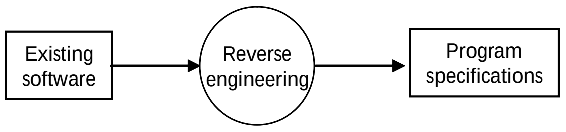
Re-Engineering Process

- **Decide** what to re-engineer. Is it whole software or a part of it?
- **Perform** Reverse Engineering, in order to obtain specifications of existing software.
- **Restructure Program** if required. For example, changing function-oriented programs into object-oriented programs.
- **Re-structure data** as required.
- **Apply Forward engineering** concepts in order to get re-engineered software.

There are few important terms used in Software re-engineering

Reverse Engineering

It is a process to achieve system specification by thoroughly analyzing, understanding the existing system. This process can be seen as reverse SDLC model, i.e. we try to get higher abstraction level by analyzing lower abstraction levels.



An existing system is previously implemented design, about which we know nothing. Designers then do reverse engineering by looking at the code and try to get the design. With design in hand, they try to conclude the specifications. Thus, going in reverse from code to system specification.

Program Restructuring

It is a process to re-structure and re-construct the existing software. It is all about re-arranging the source code, either in same programming language or from one programming language to a different one. Restructuring can have either source code-restructuring and data-restructuring or both.

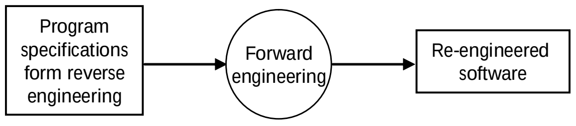
Re-structuring does not impact the functionality of the software but enhance reliability and maintainability. Program components, which cause errors very frequently can be changed, or updated with re-structuring.

The dependability of software on obsolete hardware platform can be removed via re-structuring.

Forward Engineering

Forward engineering is a process of obtaining desired software from the specifications in hand which were brought down by means of reverse engineering. It assumes that there was some software engineering already done in the past.

Forward engineering is same as software engineering process with only one difference – it is carried out always after reverse engineering.



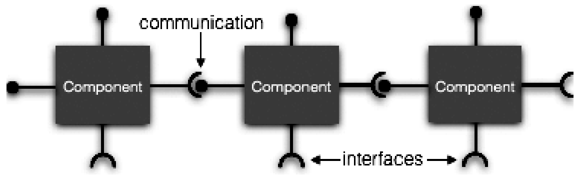
Component Reusability

A component is a part of software program code, which executes an independent task in the system. It can be a small module or sub-system itself.

Example

- The login procedures used on the web can be considered as components, printing system in software can be seen as a component of the software.

- Components have high cohesion of functionality and lower rate of coupling, i.e. they work independently and can perform tasks without depending on other modules.
- In OOP, the objects are designed are very specific to their concern and have fewer chances to be used in some other software.
- In modular programming, the modules are coded to perform specific tasks which can be used across number of other software programs.
- There is a whole new vertical, which is based on re-use of software component, and is known as Component Based Software Engineering (CBSE).



Re-use can be done at various levels

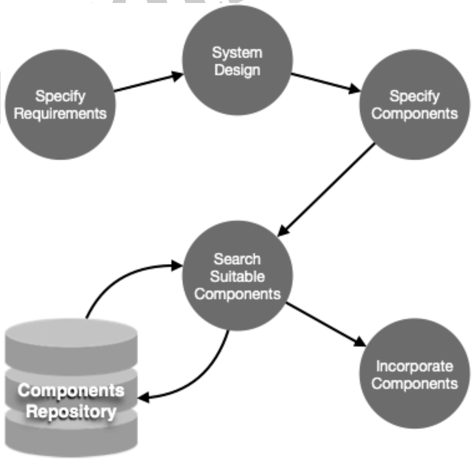
- **Application level:** Where an entire application is used as sub-system of new software.
- **Component level:** Where sub-system of an application is used.
- **Modules level** : Where functional modules are re-used.

Software components provide interfaces, which can be used to establish communication among different components.

Reuse Process

Two kinds of method can be adopted: either by keeping requirements same and adjusting components or by keeping components same and modifying requirements.

- **Requirement Specification** - The functional and non-functional requirements are specified, which a software product must comply to, with the help of existing system, user input or both.
- **Design** - This is also a standard SDLC process step, where requirements are defined in terms of software parlance. Basic architecture of system as a whole and its sub-systems are created.
- **Specify Components** - By studying the software design, the designers segregate the entire system into smaller components or sub-systems. One complete software design turns into a collection of a huge set of components working together.
- **Search Suitable Components** - The software component repository is referred by designers to search for the matching component, on the basis of functionality and intended software requirements..
- **Incorporate Components** - All matched components are packed together to shape them as complete software.



5.5 Reverse Engineering

Q7. Define Reverse Engineering. State the uses of Reengineering.

Ans :

Reverse engineering, the process of taking a software program's binary code and recreating it so as to trace it back to the original source code, is being widely used in computer hardware and software to enhance product features or fix certain bugs. For example, the programmer writes the code in a high-level language such as C, C++ etc. In short, the code written in high level language needs to be interpreted into low level or machine language.

Reverse engineering can be applied to several aspects of the software and hardware development activities to convey different meanings. In general, it is defined as the process of creating representations of systems at a higher level of abstraction and understanding the basic working principle and structure of the systems under study. With the help of reverse engineering, the software system that is under consideration can be examined thoroughly. There are two types of reverse engineering; in the first type, the source code is available, but high-level aspects of the program are no longer available. The efforts that are made to discover the source code for the software that is being developed is known as reverse engineering. In the second case, the source code for the software is no longer available; here, the process of discovering the possible source code is known as reverse engineering. To avoid copyright infringement, reverse engineering makes use of a technique called *clean room* design.

In the world of reverse engineering, we often hear about black box testing. Even though the tester has an API, their ultimate goal is to find the bugs by hitting the product hard from outside.

Uses

Reverse engineering is used in a variety of fields such as software design, software testing, programming etc.

- In software design, reverse engineering enables the developer or programmer to add new features to the existing software with or without knowing the source code. Different techniques are used to incorporate new features into the existing software.
- Reverse engineering is also very beneficial in software testing, as most of the virus programmers don't leave behind instructions on how they wrote the code, what they have set out to accomplish etc. Reverse engineering helps the testers to study the virus and other malware code. The field of software testing, while very extensive, is also interesting and requires vast experience to study and analyze virus code.
- The third category where reverse engineering is widely used is in software security. Reverse engineering techniques are used to make sure that the system does not have any major vulnerabilities and security flaws. The main purpose of reverse engineering is to make the system robust so as to protect it from spywares and hackers. Infact, this can be taken a step forward to Ethical hacking, whereby you try to hack your own system to identify vulnerabilities.
- While one needs a vast amount of knowledge to become a successful reverse engineer, he or she can definitely have a lucrative career in this field by starting off with the basics. It is highly suggested that you first become familiar with assembly level language and gain significant amount of practical knowledge in the field of software designing and testing to become a successful software engineer.

Tools

1. As mentioned above, reverse engineering is the process of analyzing the software to determine its components and their relationships. The process of reverse engineering is accomplished by making use of some tools that are categorized into debuggers or disassemblers, hex editors, monitoring and decompile tools:
2. **Disassemblers :** A disassembler is used to convert binary code into assembly code and also used to extract strings, imported and exported functions, libraries etc. The disassemblers convert the machine language into a user-friendly format. There are different dissemblers that specialize in certain things.
3. **Debuggers :** This tool expands the functionality of a disassembler by supporting the CPU registers, the hex duping of the program, view of stack etc. Using debuggers, the programmers can set breakpoints and edit the assembly code at run time. Debuggers analyse the binary in a similar way as the disassemblers and allow the reverser to step through the code by running one line at a time to investigate the results.

4. **Hex Editors :** These editors allow the binary to be viewed in the editor and change it as per the requirements of the software. There are different types of hex editors available that are used for different functions.

5. **PE and Resource Viewer :** The binary code is designed to run on a windows based machine and has a very specific data which tells how to set up and initialize a program. All the programs that run on windows should have a portable executable that supports the DLLs the program needs to borrow from.

Reverse engineering has developed significantly and taken a positive approach to creating descriptive data set of the original object. Today, there are numerous legitimate applications of reverse engineering. Due to the development of numerous digitizing devices, reverse engineering software enables programmers to manipulate the data into a useful form. The kind of applications in which reverse engineering is used ranges from mechanical to digital, each with its own advantages and applications. Reverse engineering is also beneficial for business owners as they can incorporate advanced features into their software to meet the demands of the growing markets.

5.6 Software Restructuring

Q8. Explain the Process of Implementing a code.

(OR)

Explain the concept of Restructuring.

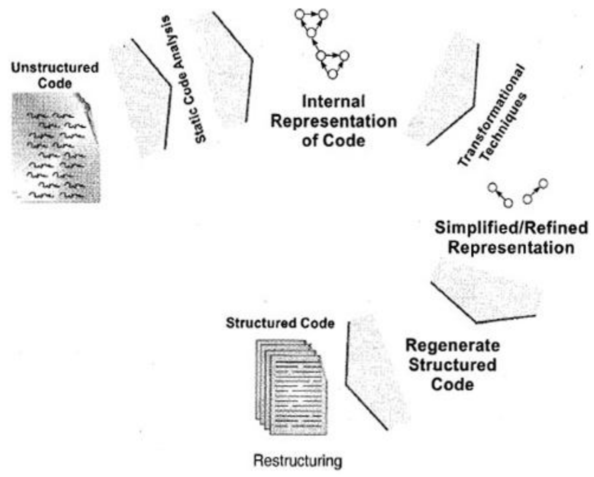
Ans :

Software restructuring is a form of perfective maintenance that modifies the structure of a program's source code. Its goal is increased maintainability to better facilitate other maintenance activities, such as adding new functionality to, or correcting previously undetected errors within a software system. Changes to the structure are introduced through the application of transformations. Manually transforming the source code may introduce undesirable as well as undetectable changes in the system's behaviour. It is very difficult to ensure that manual restructuring preserves functionality and guaranteeing it is almost impossible. The problems associated with manual restructuring can be addressed by using a restructuring tool to automatically apply transformations. The majority of restructuring tools apply transformations by manipulating abstract program representations and specify the conditions of the transformation in terms of the representation structure. The context entity graph as program representation was developed to support specific language constructs, but can be adapted to support a variety of programming languages. The implementation of a code abstraction transformation in terms of this structure is examined and various improvements are also suggested.

Restructuring

Restructuring involves the transformation of unstructured code into structured code thereby making it easier to understand and change. Restructuring involves the following steps.

1. Static analysis is performed, which provides information that is used to represent code as a directed graph or associative (semantic) network. The representation mayor may not be in a human readable form; thus, an automated tool is used.
2. Transformational techniques are used to refine (simplify) the representation.
3. Refined representation is interpreted and used to generate the structured code.


Unstructured Code
Static Code Analysis
Internal Representation of Code
Transformational Techniques
Simplified/Refined Representation
Regenerate Structured Code
Structured Code
Restructuring

5.7 Forward Reengineering

Q9. What is forward engineering in software?

Ans : **(Imp.)**

Forward engineering practice informal requirements are some how converted into a semi-formal specification using domain notations without underlying precise semantics like e.g. data-flow diagrams, entity-relationship diagrams, natural language descriptions, or other problem specific informal or semiformal notations. The program then is constructed manually (i.e. in an error prone way) from the specification by a creative agent, the programmer.

Hidden in this creative construction of the program from the specification are a set of obvious as well as non-obvious design decisions about how to encode certain parts of the specification in an efficient way using available implementation mechanisms to achieve performance criteria (the why of the design decisions). As an example, a specification fragment requiring associative retrieval using numeric keys may be implemented using hash tables, achieving good system reaction time. These decisions are usually not documented.

Over time the program code is modified to remove errors and to adapt the system to changed requirements. The requirements may change to allow usage of alphanumeric keys and to be able to handle large amounts of data, and the implementation revised to use disk-based B-trees. Unfortunately, often these changes take place without being reflected correctly in the specification. The gap between the original specification and the program becomes larger and larger. The result is a program code without a proper specification and with untrustworthy design information (such as comments describing the hash tables!). The code becomes difficult to understand and, thus, difficult to maintain.

To overcome this deficiency, it is important to change the specification first and then reflect the changes in the program code. A necessary precondition for this is to have reliable information about the relationship between the specification and the program code. The design and its rationale describe the how and why of this relationship; however, they are not documented in current practice.

There are two known approaches to reduce the gap between the specification and the program.

The first one is the development of software by stepwise refinement introducing intermediate descriptions of the system between the specification and the final program code. The intermediate descriptions should reflect major design decisions during the construction, which helps to understand the software and its design. However, this approach has some important drawbacks: the development steps are still manual, they are too large not to contain hidden design decisions, there is no rationale directly associated to the steps, the relation between the descriptions can only be established a posteriori, and there are a lot more documents to be maintained for incorporating changes of the system requirements.

Despite these drawbacks, the stepwise refinement approach is not that bad. Consequently attacking them leads us to the second approach: the transformational development of software where the fairly large manual development steps are replaced by smaller formal correctness-preserving transformations each of which reflects a small design decision or optimization.

A necessary prerequisite for this approach is to have a formal specification which may contain domain notations (e.g. for the general purpose domains of logical descriptions and data-flow diagrams and for special application domains like e.g. money management and loans) by giving them a precise underlying semantics. Based on this the program code can then be derived by making small implementation steps using formal correctness-preserving transformations leading from more abstract specifications to more concrete specifications. The final program code then is correct by construction with respect to the formal requirements specification. Each transformation falls into one of three categories: refinements (i.e. maps of concepts from an abstract level to a more concrete one), optimizations (i.e. maps to reduce the resources used at a level of abstraction according to some criteria), and jittering transformations (that eventually enable the application of refinements and optimizations). The decision to apply a particular transformation is thus crucial design information. The applied transformations coupled with the rationale for the choice of them is the design information that "explains" the code. It is called the transformational design of the code from the specification.

The selection of a transformation to apply to a certain specification in general is still a creative process but is guided by the performance criteria to be achieved. This guided selection of the transformations allows the development process to be supported by a semiautomatic system that contains a large repertoire of available transforms by applying them to achieve or at least approach the performance criteria. The transformations needed for practical software engineering comprise all those design decisions implicitly used by current software developers including techniques like the following ones:

- **Decomposition :** Most problems can be decomposed to subproblems which in general is possible in different ways. The actual hierarchical structure of the code represents only one particular choice from the set of possible decomposition.
- **Generalization/Specialization :** If different components are similar it may be possible to construct a more general version that comprises both of them as special cases. Vice versa, for efficiency reasons it may be a good choice to specialize a component for a certain (part of the) application.
- **Choice of Representation :** To be able to implement high-level data types in program code it is often necessary to change its representation. A common example is the representation of sets by lists.
- **Choice of Algorithm :** High-level concepts can be realized by an algorithm in many different ways. The choice between which may be enforced by badly documented or even undocumented performance criteria that have to be satisfied.
- **Interleaving :** For efficiency reasons it may be useful to realize different concepts in the same section of code or the same data structure.
- **Delocalization :** Certain high-level concepts may be spread throughout the whole code introducing distracting details in other concepts (see the study of the effects of delocalization on comprehension in [LS86]).
- **Resource Sharing :** Interleaved code often allows different concepts to share some resources like e.g. control conditions, intermediate data results, functions, names, and other computational resources.
- **Data caching/ Memorization :** If some data, that has to be computed, is needed often or its computation is expensive it is worth to cache the data, i.e. to memorize it for later reuse.

Optimizations

To achieve highly efficient code (due to the need of satisfying e.g. memory or reaction time constraints) many different optimizations are used. Such optimizations comprise

1. The folding and unfolding (in lining) program code
2. The use of algebraic properties of functions,
3. The merging of computations by composing or combining functions
4. Partial evaluation (often in the form of context-dependent simplification)
5. Finite differencing
6. Result accumulation
7. Recursion simplification and elimination
8. Loop fusion
9. Optimizations typically performed by good compilers (code motion, common sub-expression elimination, partial redundancy removal).
10 Domain specific optimizations like e.g. minimizing a finite state machine recognizing a certain language.

All these design decisions may overlap and may be delocalized during the construction of the program code from the specification. Whether these methods are carried out mechanically by a tool or informally by smart programmers, the resulting software systems are very difficult to understand.

The transformational software development approach has the advantage that it allows the automatic recording of the design decisions made during the derivation of the final program code from the formal specification. Provided the selection of the transformations that are applied during this development are guided by the performance criteria to be achieved and the relation between the selection and the performance criteria is recorded we get the complete design together with its rationale as the product of software development. The code itself is just a byproduct that is correct by construction.

Recording this design information allows us to modify the specification instead of the code and then to modify the design to get a new implementation of the modified specification. The formal nature of transformations makes such design modifications supportable by a semiautomatic tool that increases the reliability and reduces the cost of maintaining the system.

5.8 Economics of Re-Engineering

Q10. Explain the concept of Economics of Reengineering.

Ans : **(Imp.)**

Every unmaintainable program would be retired immediately, to be replaced by high-quality, reengineered applications developed using modern software engineering practices. But we live in a world of limited resources. Reengineering drains resources that can be used for other business purposes. Therefore, before an organization attempts to reengineer an existing application, it should perform a cost/benefit analysis.

A cost/benefit analysis model for reengineering has been proposed by Sneed . Nine parameters are defined.

- **P1** = current annual maintenance cost for an application.
- **P2** = current annual operation cost for an application.
- **P3** = current annual business value of an application.
- **P4** = predicted annual maintenance cost after reengineering.
- **P5** = predicted annual operations cost after reengineering.
- **P6** = predicted annual business value after reengineering.
- **P7** = estimated reengineering costs.
- **P8** = estimated reengineering calendar time.
- **P9** = reengineering risk factor (P9 = 1.0 is nominal).
- **L** = expected life of the system.

The cost associated with continuing maintenance of a candidate application (i.e., reengineering is not performed) can be defined as

Cmaint = [P3 - (P1 + P2)] x L

L The costs associated with reengineering are defined using the following relationship:

Creeng = [P6 - (P4 + P5) x (L - P8) - (P7 x P9)]

Using the costs presented in equations, the overall benefit of reengineering can be computed as

cost benefit = Creeng - Cmaint

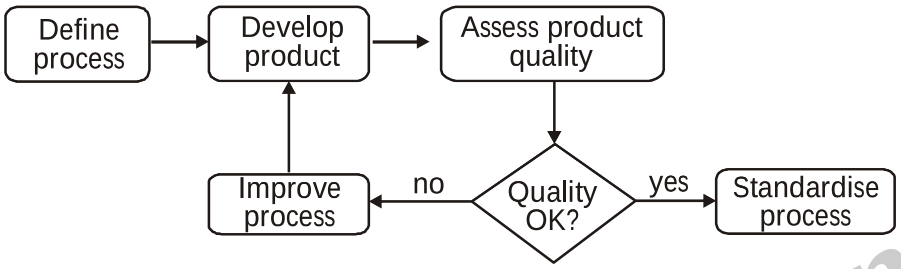
The cost/benefit analysis presented in the equations can be performed for all high priority applications identified during inventory analysis . Those application that show the highest cost/benefit can be targeted for reengineering, while work on others can be postponed until resources are available.

5.9 SPI Process

Q11. Discuss in detail about Process and Product Quality In Software Engineering.

Ans : **(Imp.)**

It is general, that the quality of the development process directly affects the quality of delivered products. The quality of the product can be measured and the process is improved until the proper quality level is achieved. The process of quality assessment based on this approach.



In manufacturing systems there is a clear relationship between production process and product quality. However, quality of software is highly influenced by the experience of software engineers. In addition, it is difficult to measure software quality attributes, such as maintainability, reliability, usability, etc., and to tell how process characteristics influence these attributes. However, experience has shown that process quality has a significant influence on the quality of the software.

Process quality management includes the following activities :

1. Defining process standards.
2. Monitoring the development process.
3. Reporting the software.

1. Quality Assurance and Standards

Quality assurance is the process of defining how software quality can be achieved and how the development organization knows that the software has the required level of quality. The main activity of the quality assurance process is the selection and definition of standards that are applied to the software development process or software product. There are two main types of standards. The product standards are applied to the software product, i.e. output of the software process. The process standards define the processes that should be followed during software development. The software standards are based on best practices and they provide a framework for implementing the quality assurance process.

The development of software engineering project standards is a difficult and time consuming process. National and international bodies such as ANSI and the IEEE develop standards that can be applied to software development projects. Organizational standards, developed by quality assurance teams, should be based on these national and international standards.

Product standards	Process standards
Requirements document structure	Project plan approval process
Method header format	Version release process
Java programming style	Change control process
Change request form	Test recording process

2. ISO

ISO 9000 is an international set of standards that can be used in the development of a quality management system in all industries. ISO 9000 standards can be applied to a range of organizations from manufacturing to service industries. ISO 9001 is the most general of these standards. It can be applied to organizations that design, develop and maintain products and develop their own quality processes. A supporting document (ISO 9000-3) interprets ISO 9001 for software development.

The ISO 9001 standard isn't specific to software development but includes general principles that can be applied to software development projects. The ISO 9001 standard describes various aspects of the quality process and defines the organizational standards and procedures that a company should define and follow during product development. These standards and procedures are documented in an organizational quality manual.

The ISO 9001 standard does not define the quality processes that should be used in the development process. Organizations can develop own quality processes and they can still be ISO 9000 compliant companies. The ISO 9000 standard only requires the definition of processes to be used in a company and it is not concerned with ensuring that these processes provide best practices and high quality of products. Therefore, the ISO 9000 certification doesn't means exactly that the quality of the software produced by an ISO 9000 certified companies will be better than that software from an uncertified company.

3. Documentation Standards

Documentation standards in a software project are important because documents can represent the software and the software process. Standardized documents have a consistent appearance, structure and quality, and should therefore be easier to read and understand. There are three types of documentation standards:

1. Documentation process standards. These standards define the process that should be followed for document production.
2. Document standards. These standards describe the structure and presentation of documents.
3. Documents interchange standards. These standards ensure that all electronic copies of documents are compatible.

4. Quality planning

Quality planning is the process of developing a quality plan for a project. The quality plan defines the quality requirements of software and describes how these are to be assessed. The quality plan selects those organizational standards that are appropriate to a particular product and development process. Quality plan has the following parts:

1. Introduction of product.
2. Product plans.
3. Process descriptions.
4. Quality goals.
5. Risks and risk management.

The quality plan defines the most important quality attributes for the software and includes a definition of the quality assessment process. Table shows generally used software quality attributes that can be considered during the quality planning process.

Safety	Understandability	Portability
Security	Testability	Usability
Reliability	Adaptability	Reusability
Resilience	Modularity	Efficiency
Robustness	Complexity	Learn ability
Maintainability		

Quality control provides monitoring the software development process to ensure that quality assurance procedures and standards are being followed. The deliverables from the software development process are checked against the defined project standards in the quality control process. The quality of software project deliverables can be checked by regular quality reviews and/or automated software assessment. Quality reviews are performed by a group of people. They review the software and software process in order to check that the project standards have been followed and that software and documents conform to these standards. Automated software assessment processes the software by a program that compares it to the standards applied to the development project.

Q12. What do you mean by Process Improvement and Maturity? Explain the CMM Model.

Ans :

It is believed that only by improving the process can be quality & productivity be improved. Process improvement required understanding the current process & its deficiencies & then taking actions to remove the deficiencies. This is possible only if the current process is under statistical control.

Capability Maturity Model (CMM)

To improve its software process, an organization needs to first understand the states & then develop a plan to improve the process. It is generally agreed that changes to a process must be introduced in small increments. The changes that take place depend on the current state of the process. This concept of introducing changes in small increments based on the current state of the process has been captured in the capability Maturity Model (CMM) framework. The CMM framework provides a general roadmap for process improvement.

Software process capability describes the range of expected results that can be achieved by following the process. The process capability of an organization determines what can be expected from the organization in terms of quality & productivity. The CMM framework provides process improvement in small increments as processes go from their current levels to the next higher level when they are improved. Hence, during the course of process improvement, a process moves from level to level until it reaches Level 5.

The CMM provides characteristics of each level, which can be used to assess the current level of the process of an organization. As the movement from one level is to the next level, the characteristics of the levels also suggest the areas in which the process should be improved so that it can move to the next higher level. For each level, it specifies the areas in which improvement can be absorbed & will bring the maximum benefits.

- **Level 1:** The initial process is an ad-hoc process that has no formalized method for any activity. Basic project controls for: ensuring that activities are being done properly but the project plan is missing. Success in such organizations depends solely on the quality & capability of individuals. The process capability is unpredictable as the process constantly changes. Organizations at this level can benefit most by improving project management, quality assurance & change control.
- **Level 2:** In a repeatable process, policies for managing a software project & procedures to implement those policies exist. That is, project management is well developed in a process at this level. Project commitments are realistic & based on past experience with similar projects. Cost & schedule are tracked & problems resolved when they arise. Formal configuration control mechanism exist results obtains by this process can be repeated as the project planning & tracking is formal.
- **Level 3:** At the defined level, the organization has standardized on a software process, which is properly documented. A software process group exists in the organization that owns & manages the process. In the process, each step is carefully defined with verifiable entry & exit criteria, methods for performing the step, & verification mechanism for the output of the step.
- **Level 4:** At the managed level, quantitative goals exist for process & products Data is collected from software processes, which is used to build models to characterize the process. Due to the models, the organization has a good insight in the process. The results

of using such a process can be predicted in quantitative terms.

- **Level 5:** At the optimizing level, the focus is on continuous improvement. Data is collected & routinely analyzed to identify areas that can be strengthened to improve quality or productivity. New technologies & tools are introduced & their effects are measured to improve the performance of the process. Best software engineering & management practices are used throughout the organization.

Q13. What are measurements used for the Process improvement in software?

Ans :

Software measurement provides a numeric value for some quality attribute of a software product or a software process. Comparison of these numerical values to each other or to standards draws conclusions about the quality of software or software processes. Software product measurements can be used to make general predictions about a software system and identify anomalous software components.

Software metric is a measurement that relates to any quality attributes of the software system or process. It is often impossible to measure the external software quality attributes, such as maintainability, understandability, etc., directly. In such cases, the external attribute is related to some internal attribute assuming a relationship between them and the internal attribute is measured to predict the external software characteristic. Three conditions must be hold in this case:

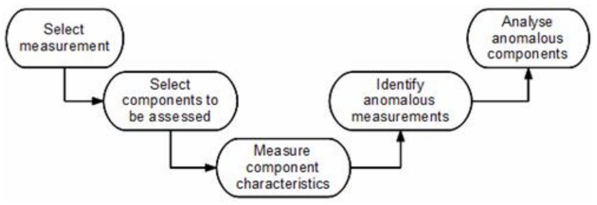
1. The internal attribute must be measured accurately.
2. A relationship must exist between what we can measure and the external behavioural attribute.
3. This relationship has to be well understood, has been validated and can be expressed in terms of a mathematical formula.

The Measurement Process

A software measurement process as a part of the quality control process is shown in Figure 12.2. The steps of measurement process are the followings:

1. Select measurements to be made. Selection of measurements that are relevant to answer the questions to quality assessment.
2. Select components to be assessed. Selection of software components to be measured.
3. Measure component characteristics. The selected components are measured and the associated software metric values computed.
4. Identify anomalous measurements. If any metric exhibit high or low values it means that component has problems.
5. Analyze anomalous components. If anomalous values for particular metrics have been identified these components have to be examined to decide whether the anomalous metric values mean that the quality of the component is compromised.

Generally each of the components of the system is analyzed separately. Anomalous measurements identify components that may have quality problems.


Select measurement
Select components to be assessed
Measure component characteristics
Identify anomalous measurements
Analyse anomalous components

Product Metrics

The software characteristics that can be easily measured such as size do not have a clear and consistent relationship with quality attributes such as understandability and maintainability. Product metrics has two classes:

1. Dynamic metrics. These metrics (for example execution time) are measured during the execution of a program.

2. Static metrics. Static metrics are based on measurements made of representations of the system such as the design, program or documentation.

Dynamic metrics can be related to the efficiency and the reliability of a program. Static metrics such as code size are related to software quality attributes such as complexity, understandability, maintainability, etc.

5.10 Process Change Management (PCM)

Q14. What is Process Change Management? Explain in Detail.

Ans :

Changes in the process are unavoidable and should be made to improve the productivity and quality of the process. These changes can be applied by using Process Change Management (PCM), which is a technique to improve the software processes in the organization. Thus, this technique helps in increasing productivity and quality by improving software processes for the developing software.

Process change management determines the process improvement goals, identifies, evaluates and implements improvements to the standard software process in the organization and defines software processes for the new projects to be developed. The organization follows a written policy for implementing software process improvements. The policy follows the steps listed below.

- The organization has the quantitative and measurable goals to improve the software processes and tracks performance against these goals.
- The organization's process improvement is directed towards improving the quality, increasing productivity and decreasing the time for the development of the product.
- The organization's staff members participate in the process improvement.
- Process improvement does not take place in a single stage; it is a continuous process which goes through different stages as listed in Table. The software process improvement activities are managed according to a documented procedure, which is maintained in the central repository. Review of the processes is conducted in a phased and structured manner to ensure that continuous process improvement activities are conducted and monitored.

Stages	Description
Process analysis	Model and analyze existing processes.
Improvement identification	Identify quality, cost or schedule bottlenecks.
Process change introduction	Modify the process to remove identified bottlenecks.
Measure improvement	Compare outputs between previous and changed processes.
Process change training	Train staff involved in new process proposals.
Change tuning	Evolve and improve process improvements.

Training programs are established to enable and encourage individuals in the organization to participate in the process improvement activities. Improvement opportunities are identified and evaluated, which benefit the organization. When software process improvements are approved for general practice, the organization standard software process and the project-defined software processes are revised appropriately.

Software process change management follows a procedure for improving the software processes in the organization. This procedure specifies the following steps.

1. The improvement of the software process requires a proposal to be submitted, which includes the organization's software process improvement goals and recommendations for software process assessment. In addition, it contains analysis of data on the performance of the project as compared to the quality of the software and the productivity goals.
2. The proposal is evaluated to ensure whether to implement it. The decision for implementing the proposal is then documented.
3. The benefits expected from the software process are determined. These benefits include quality of the product, end user satisfaction, and so on.
4. The priority of software process improvement proposals is determined, according to which selection for the implementation is made.
5. Implementation of the software process improvement actions resulting from the proposal is assigned and planned.
6. The actions for software process improvement, which require significant effort, are assigned to the team responsible for implementing the actions.
7. The status of each process improvement proposal is monitored.
8. Software process improvement proposals whose responses are unusually long are identified and acted upon.
9. The user reviews the changes before they are implemented, which are estimated to have an impact on the quality of the product.
10. When the actions of the software process improvement are completed, they are reviewed and verified.
11. The submitter of the software process improvement proposal receives acknowledgement of proposals and notification of the disposition. of the proposals.

When a decision is made to transfer software process improvement into general practice, the improvement is again implemented according to a documented procedure, which specifies the following steps.

1. The resources required to manage the changes in the software process are established.
2. The strategy to collect data for evaluating and monitoring changes in the performance of the software process is reviewed, agreed and documented. The individuals, who are implementing the software processes affected by the change, agree to this strategy. The support tools are instrumented accordingly to record the required data automatically.
3. Training is provided before installing the process change for general use. Training courses are updated to reflect the current software processes.
4. Appropriate process changes are incorporated into the organization's standard software process and the project-defined software processes.

5.11 CMM

Q15. What is the Capability Maturity Model?

Ans :

Capability Maturity Model (CMM) broadly refers to a process improvement approach that is based on a process model. CMM also refers specifically to the first such model, developed by the Software Engineering Institute (SEI) in the mid-1980s, as well as the family of process models that followed. A process model is a structured collection of practices that describe the characteristics of effective processes; the practices included are those proven by experience to be effective.

CMM can be used to assess an organization against a scale of five process maturity levels. Each level ranks the organization according to its standardization of processes in the subject area being assessed. The subject areas can be as diverse as software engineering, systems engineering, project management, risk management, system acquisition, information technology (IT) services and personnel management.

CMM was developed by the SEI at Carnegie Mellon University in Pittsburgh. It has been used extensively for avionics software and government projects, in North America, Europe, Asia, Australia, South America, and Africa.Currently, some government departments require software development contract organization to achieve and operate at a level 3 standard.

History

The Capability Maturity Model was initially funded by military research. The United States Air Force funded a study at the Carnegie-Mellon Software Engineering Institute to create a model (abstract) for the military to use as an objective evaluation of software subcontractors. The result was the Capability Maturity Model, published as Managing the Software Process in 1989. The CMM is no longer supported by the SEI and has been superseded by the more comprehensive Capability Maturity Model Integration (CMMI).

Maturity Model

The Capability Maturity Model (CMM) is a way to develop and refine an organization's processes. The first CMM was for the purpose of developing and refining software development processes. A maturity model is a structured collection of elements that describe characteristics of effective processes. A maturity model provides:

- A place to start the benefit of a community's prior experiences
- A common language and a shared vision
- A framework for prioritizing actions
- A way to define what improvement means for your organization

A maturity model can be used as a benchmark for assessing different organizations for equivalent comparison. It describes the maturity of the company based upon the project the company is dealing with and the clients.

Context

In the 1970s, technological improvements made computers more widespread, flexible, and inexpensive. Organizations began to adopt more and more computerized information systems and the field of software development grew significantly. This led to an increased demand for developers—and managers which was satisfied with less experienced professionals.

Unfortunately, the influx of growth caused growing pains; project failure became more commonplace not only because the field of computer science was still in its infancy, but also because projects became more ambitious in scale and complexity. In response, individuals such as Edward Yourdon, Larry Constantine, Gerald Weinberg, Tom DeMarco, and David Parnas published articles and books with research results in an attempt to professionalize the software development process.

Watts Humphrey's Capability Maturity Model (CMM) was described in the book Managing the Software Process (1989). The CMM as conceived by Watts Humphrey was based on the earlier work of Phil Crosby. Active development of the model by the SEI began in 1986.

The CMM was originally intended as a tool to evaluate the ability of government contractors to perform a contracted software project. Though it comes from the area of software development, it can be, has been, and continues to be widely applied as a general model of the maturity of processes in IS/IT (and other) organizations.

The model identifies five levels of process maturity for an organisation. Within each of these maturity levels are KPAs (Key Process Areas) which characterise that level, and for each KPA there are five definitions identified:

1. Goals
2. Commitment
3. Ability
4. Measurement
5. Verification

The KPAs are not necessarily unique to CMM, representing - as they do - the stages that organizations must go through on the way to becoming mature.

The assessment is supposed to be led by an authorised lead assessor. One way in which companies are supposed to use the model is first to assess their maturity level and then form a specific plan to get to the next level. Skipping levels is not allowed.

Time line

- 1987 SEI-87-TR-24 (SW-CMM questionnaire), released.
- 1989 Managing the Software Process, published.
- 1991 SW-CMM v1.0, released.
- 1993 SW-CMM v1.1, released.
- 1997 SW-CMM revisions halted in support for CMMI.
- 2000 CMMI v1.02, released.
- 2002 CMMI v1.1, released.
- 2006 CMMI v1.2, released.

Current State

Although these models have proved useful to many organizations, the use of multiple models has been problematic. Further, applying multiple models that are not integrated within and across an organization is costly in terms of training, appraisals, and improvement activities. The CMM Integration project was formed to sort out the problem of using multiple CMMs. The CMMI Product Team's mission was to combine three source models:

1. The Capability Maturity Model for Software (SW-CMM) v2.0 draft C
2. The Systems Engineering Capability Model (SECM)
3. The Integrated Product Development Capability Maturity Model (IPD-CMM) v0.98
4. Supplier sourcing

CMMI is the designated successor of the three source models. The SEI has released a policy to sunset the Software CMM and previous versions of the CMMI. The same can be said for the SECM and the IPD-CMM; these models were superseded by CMMI.

Levels of the CMM

There are five levels of the CMM:

Level 1 - Initial

1. Processes are usually ad hoc and the organization usually does not provide a stable environment. Success in these organizations depends on the competence and heroics of the people in the organization and not on the use of proven processes. In spite of this ad hoc, chaotic environment, maturity level 1 organizations often produce products and services that work; however, they frequently exceed the budget and schedule of their projects.
2. Organizations are characterized by a tendency to over commit, abandon processes in the time of crisis, and not be able to repeat their past successes again.

3 Software project success depends on having quality people.

Level 2 - Repeatable

1. Software development successes are repeatable. The processes may not repeat for all the projects in the organization. The organization may use some basic project management to track cost and schedule.
2. Process discipline helps ensure that existing practices are retained during times of stress. When these practices are in place, projects are performed and managed according to their documented plans.
3. Project status and the delivery of services are visible to management at defined points (for example, at major milestones and at the completion of major tasks).
4. Basic project management processes are established to track cost, schedule, and functionality. The minimum process discipline is in place to repeat earlier successes on projects with similar applications and scope. There is still a significant risk of exceeding cost and time estimate.

Level 3 - Defined

1. The organization's set of standard processes, which is the basis for level 3, is established and improved over time. These standard processes are used to establish consistency across the organization. Projects establish their defined processes by the organization's set of standard processes according to tailoring guidelines.
2. The organization's management establishes process objectives based on the organization's set of standard processes and ensures that these objectives are appropriately addressed.
3. A critical distinction between level 2 and level3 is the scope of standards, process descriptions, and procedures. At level 2, the standards, process descriptions, and procedures may be quite different in each specific instance of the process (for example, on a particular project). At level 3, the standards, process descriptions, and procedures for a project are tailored from the organization's set of standard processes to suit a particular project or organizational unit.

Level 4 - Managed

1. Using precise measurements, management can effectively control the software development effort. In particular, management can identify ways to adjust and adapt the process to particular projects without measurable losses of quality or deviations from specifications. At this level organization set a quantitative quality goal for both software process and software maintenance.
2. Subprocesses are selected that significantly contribute to overall process performance. These selected subprocesses are controlled using statistical and other quantitative techniques.
3. A critical distinction between maturity level 3 and maturity level 4 is the predictability of process performance. At maturity level 4, the performance of processes is controlled using statistical and other quantitative techniques, and is quantitatively predictable. At maturity level 3, processes are only qualitatively predictable.

Level 5 - Optimizing

1. Focusing on continually improving process performance through both incremental and innovative technological improvements. Quantitative process-improvement objectives for the organization are established, continually revised to reflect changing business objectives, and used as criteria in managing process improvement. The effects of deployed process improvements are measured and evaluated against the quantitative process-improvement objectives. Both the defined processes and the organization's set of standard processes are targets of measurable improvement activities.
2. Process improvements to address common causes of process variation and measurably improve the organization's processes are identified, evaluated, and deployed.
3. Optimizing processes that are nimble, adaptable and innovative depends on the participation of an empowered workforce

aligned with the business values and objectives of the organization. The organization's ability to rapidly respond to changes and opportunities is enhanced by finding ways to accelerate and share learning.

A critical distinction between maturity level 4 and maturity level 5 is the type of process variation addressed. At maturity level 4, processes are concerned with addressing special causes of process variation and providing statistical predictability of the results. Though processes may produce predictable results, the results may be insufficient to achieve the established objectives. At maturity level 5, processes are concerned with addressing common causes of process variation and changing the process (that is, shifting the mean of the process performance) to improve process performance (while maintaining statistical probability) to achieve the established quantitative process-improvement objectives.The most beneficial elements of CMM Level 2 and 3:

1. Creation of Software Specifications, stating what is going to be developed, combined with formal sign off, an executive sponsor and approval mechanism. This is NOT a living document, but additions are placed in a deferred or out of scope section for later incorporation into the next cycle of software development.
2. A Technical Specification, stating how precisely the thing specified in the Software Specifications is to be developed will be used. This is a living document.
3. Peer Review of Code (Code Review) with metrics that allow developers to walk through an implementation, and to suggest improvements or changes. Note - This is problematic because the code has already been developed and a bad design can not be fixed by "tweaking", the Code Review gives complete code a formal approval mechanism.
4. Version Control - a very large number of organizations have no formal revision control mechanism or release mechanism in place.
5. The idea that there is a "right way" to build software, that it is a scientific process involving engineering design and that groups of developers are not there to simply work on the problem du jour.

5.12 The People Capability Maturity Model

Q16. Describe in detail the frame work of PCCM.

Ans : **(Imp.)**

The People Capability Maturity Model adapts the maturity framework of the Capability Maturity Model for Software (CMMSM), to managing and developing an organization's workforce. The motivation for the P-CMM is to radically improve the ability of organizations to attract, develop, motivate, organize, and retain the talent needed to continuously improve software development capability.

Based on the best current practices in the fields such as human resources and organizational development, the P-CMM provides organizations with guidance on how to gain control of their processes for managing and developing their workforce. The P-CMM helps organizations to characterize the maturity of their workforce practices, guide a program of continuous workforce development, set priorities for immediate actions, integrate workforce development with process improvement, and establish a culture of excellence. It describes an evolutionary improvement path from adhoc, inconsistently performed practices, to a mature, disciplined development of the knowledge, skills, and motivation of the workforce.

The P-CMM consists of five maturity levels that lay successive foundations for continuously improving talent, developing effective teams, and successfully managing the people assets of the organization. Each maturity level is a well-defined evolutionary plateau that institutionalizes a level of capability for developing the talent within the organization.

The five stages of the People CMM framework are:

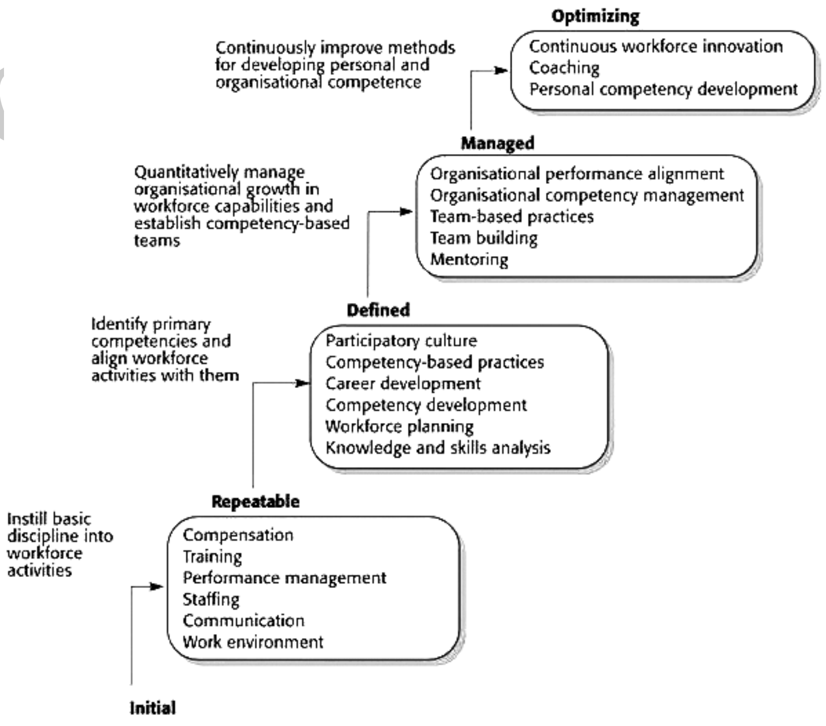
1. **P-CMM - Initial Level** (Typical characteristics: Inconsistency in performing practices, Displacement of responsibility, Ritualistic practices, and Emotionally detached workforce).

2. **P-CMM - Managed Level** (Typical characteristics: Work overload, Environmental distractions, Unclear performance objectives or feedback, Lack of relevant knowledge, or skill, Poor communication, Low morale)

3. **P-CMM - Defined Level** (Although there are performing basic workforce practices, there is inconsistency in how these practices are performed across units and little synergy across the organization. The organization misses opportunities to standardize workforce practices because the common knowledge and skills needed for conducting its business activities have not been identified)

4. **P-CMM - Predictable Level** (The organization manages and exploits the capability created by its framework of workforce competencies. The organization is now able to manage its capability and performance quantitatively. The organization is able to predict its capability for performing work because it can quantify the capability of its workforce and of the competency-based processes they use in performing their assignments)

5. **P-CMM - Optimizing Level** (The entire organization is focused on continual improvement. These improvements are made to the capability of individuals and workgroups, to the performance of competency-based processes, and to workforce practices and activities. The organization uses the results of the quantitative management activities established at Maturity Level 4 to guide improvements at Maturity Level 5. Maturity Level 5 organizations treat change management as an ordinary business process to be performed in an orderly way on a regular basis)

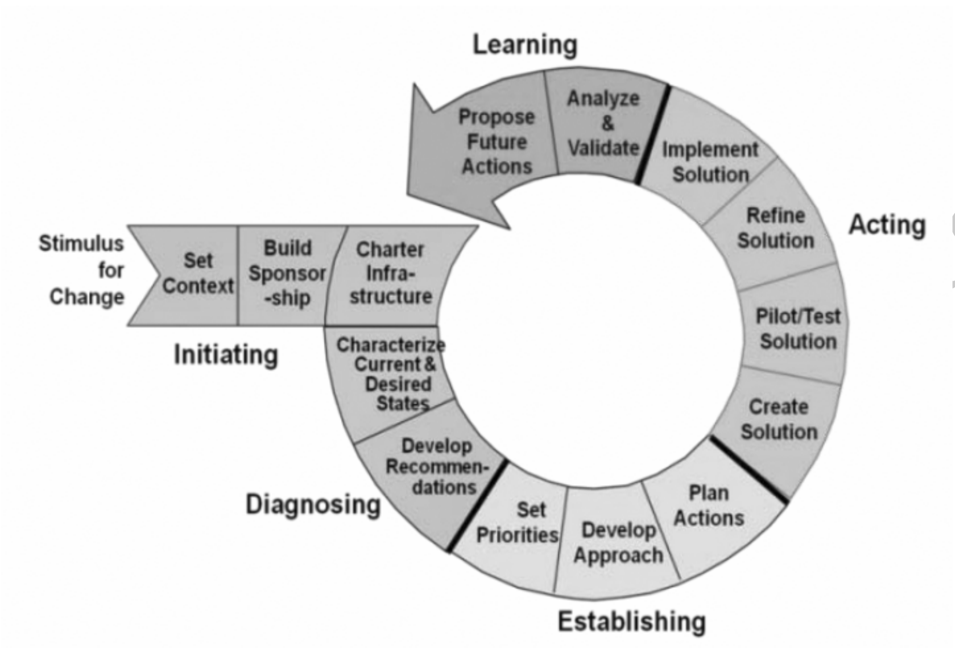

Optimizing
Continuously improve methods for developing personal and organisational competence
Continuous workforce innovation
Coaching
Personal competency development
Managed
Quantitatively manage organisational growth in workforce capabilities and establish competency-based teams
Organisational performance alignment
Organisational competency management
Team-based practices
Team building
Mentoring
Defined
Identify primary competencies and align workforce activities with them
Participatory culture
Competency-based practices
Career development
Competency development
Workforce planning
Knowledge and skills analysis
Repeatable
Instill basic discipline into workforce activities
Compensation
Training
Performance management
Staffing
Communication
Work environment
Initial

5.13 IDEAL Model

Q17. Explain in detail about Ideal Model and Spice in SPI.

Ans : **(Imp.)**

The IDEAL model is developed by Software Engineering Institute (SEI) of Carnegie Mellon University in 1996 for software process improvement. It is an approach to continuous improvement which includes some phases to build a successful improvement strategy. [4 5] This model consists of five phases i.e. initiating, diagnosing, establishing, acting and leveraging phase.



All these phases are described as follows:

1. **Initiating Phase**

   It is the starting point. In this phase, establishment of initial improvement infrastructure is done, the roles and responsibilities of infrastructure are specified, and the initial resources are appointed. The general objectives of SPI model are also specified in this initiating phase.

2. **Diagnosing Phase**

   This phase starts the continuous software process improvement path and lays groundwork for the later phases. In this phase, lessons are learned from the past improvements efforts, and according to the organization's view, strategic business plans and SPI action plan is initiated.

3. **Establishing Phase**

   During the establishing phase, issues are prioritized which are decide by organization to include in their improvement activities and also the solutions regarding those issues are developed. The general goals which are defined in initiating phase are used to develop measurable goals and these measurable goals are included in the final version of the SPI action plan.

4. **Acting Phase**

   In the acting phase, the solutions to improvements are created, controlled and arranged to see whether it is going in a right direction or not, refined and then implement these solutions. This phase is all about doing the work required to achieve the desired goals.

5. **Leveraging Phase**

   The main goal of the leveraging phase is to make the next pass through the IDEAL model more adequate. Solutions are made, and lessons are learned from the past and by adopting these ways we can improve in the future.

Advantages

IDEAL model provides a disciplined strategy and for the improvement process and establishes a long-term improvement strategy. It is the effective approach to software process improvement methods.

Disadvantages

Since, it is a continuous improvement model. No recovery phase exits in this model and hence either it will be successful or fail all at once.

Q18. Describe briefly about SPICE Model.

Ans :

ISO/IEC validated a project under the working title SPICE (Software Process Improvement and capability determination) in January 1993. It includes the participation of dominant world experts in the field of software engineering. The main aim of this project was to determine a standard for determining the ability of its continual improvement and software production process. Standard ISO/IEC 15504 or SPICE provides a framework for the determination of software processes. This framework can be used by organizations involved in devising, executing, monitoring, controlling and improving the attainment, supply, development, operation, measurement and support of software.

SPICE has a process model architecture which is two dimensional structures: the process dimension and the capability dimension. The process dimension includes the main processes and the base practices under the process categories. These process categories are as follows:

1. **Customers-Suppliers (CUS)**

   It consists of 8 CUS processes. These are the group of processes which directly affect the customers, transition of the software to the customer, and support development.

2. **Engineering (ENG)**

   It consists of 7 ENG processes. It defines straightaway, executes or maintains a system and software product, also maintains its user documentation.

3. **Project (PRO)**

   It consists of 8 PRO processes. It consist a group of processes that helps in building the project, combine, and manages its resources to produce a product or provide a service that satisfies the requirement of customer.

4. **Supports (SUP)**

   It consists of 5 SUP processes. It supports and prepares the performance of the processes in a project.

1. ATM APPLICATION

1. Phases in Software Development Project, Overview, Need, Coverage of Topics

Ans :

The software requirements for an automated teller machine network. ATM It is intended for the designer developer and maintainer of the ATM.

Scope

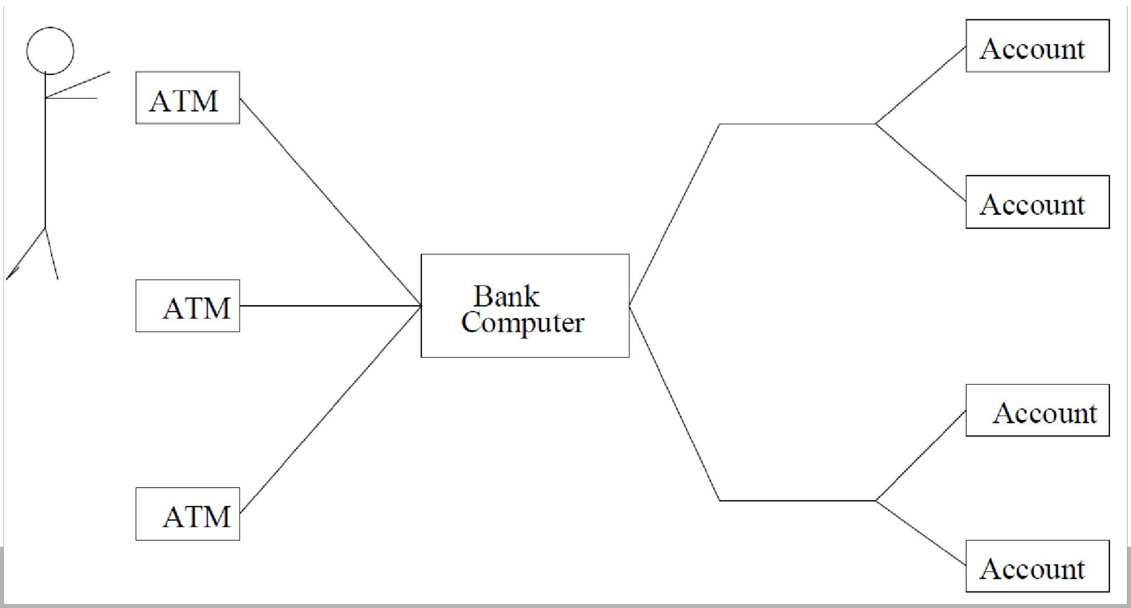
The function of the ATM is to support a computerized banking network.

Overview

There will be some defnitions of important terms Section contains a general description of the ATM Section identifies the specific functional requirements the external interfaces and performance requirements of the ATM.

Definitions

- **Account:** A single account in a bank against which transactions can be applied Accounts may be of various types with at least checking and savings A customer can hold more than one account.
- **ATM:** A station that allows customers to enter their own transactions using cash cards as identification The ATM interacts with the customer to gather transaction information sends the transaction information to the central computer for validation and processing and dispenses cash to the customer We assume that an ATM need not operate independently of the network.
- **Bank:** A financial institution that holds accounts for customers and that issues cash cards authorizing access to accounts over the ATM network.
- **Bank computer:** The computer owned by a bank that interfaces with the ATM network and the banks own cashier stations A bank may actually have its own internal network of computers to process accounts but we are only concerned with the one that interacts with the network.
- **Cash card:** A card assigned to a bank customer that authorizes access to accounts using an ATM machine Each card contains a bank code and a card number coded in accordance with national standards on credit cards and cash cards The bank code uniquely identifies the bank within the consortium The card number determines the accounts that the card can access A card does not necessarily access all of a customer s accounts Each cash card is owned by a single customer but multiple copies of it may exist so the possibility of simultaneous use of the same card from different machines must be considered.
- **Customer:** The holder of one or more accounts in a bank A customer can consist of one or more persons or corporations the correspondence is not relevant to this problem The same person holding an account at a different bank is considered a different customer.
- **Transaction:** A single integral request for operations on the accounts of a single customer. We only specified that ATMs must dispense cash but we should not preclude the possibility of printing checks or accepting cash or checks We may also want to provide the exibility to operate on accounts of different customers although it is not required yet. The different operations must balance properly.



User Characteristics

There are several users of the ATM network

- **Customer:** The customer interacts with the ATM network via the ATM It must be very easy for them to use the ATM They should be supported by the system in every possible way.
- **Maintainer:** It should be easy to maintain the whole system The maintainer should be the only person that is allowed to connect a new ATM to the network

Abbreviations

Throughout this document the following abbreviations are used

- k is the maximum withdrawal per day and account
- m is the maximum withdrawal per transaction
- n is the minimum cash in the ATM to permit a transaction t is the total fund in the ATM at start of day.

2. To assign the requirement engineering tasks.

Ans :

Specific Requirements

Functional Requirements

The functional requirements are organized in two sections First requirements of the ATM and second requirements of the bank.

Requirements of the automated teller machine

The requirements for the automated teller machine are organized in the following way General requirements for authorization requirements for a transaction.

General Functional requirement

Description-Initialize parameters t, k, m, n.

- **Input-**ATM is initialized with t dollars, k, m, n, are entered

- **Processing-**Storing the parameters
- **Output-**Parameters are set

Performance Requirement

Description -Each bank may be processing transactions from several ATMs at the same time.

Attributes

- **Availability-** The ATM network has to be available hours a day.
- **Security -** The ATM network should provide maximal security. In order to make that much more transparent there are the following requirements. It must be impossible to plug into the network.
- **Maintainability-** Only maintainers are allowed to connect new ATM s to the network.
- Transferability Conversions Not Applicable.

Other Requirements

- **Data Base -**The ATM must be able to use several data formats according to the data formats that are provided by the data bases of different banks A transaction should have all the properties of a data base transaction Atomicity Consistency Isolation Durability.

3. To perform the system analysis: Requirement analysis, SRS.

Ans :

Hardware Interface Requirements

There are various hardware components with which the machine is required to interact. Various hardware interface requirements that need to be fulfilled for successful functioning of the software are as follows:

- The ATM power supply shall have a 10/220 V AC manual switch.
- The ATM card should have the following physical dimensions:-
  - **Width** - 85.47mm - 85.72mm
  - **Height** - 53.92mm - 54.03mm
  - **Thickness** - 0.76mm - 0.08mm
- The card reader shall be a magnetic stripe reader
- The card reader shall have Smart card option.
- The slot for a card in the card reader may include an extra indentation for the embossed area of the card. In effect it acts as a polarization key and may be used to aid the correct insertion orientation of the card. This is an additional characteristic to the magnetic field sensor which operates off the magnetic stripe and is used to open a mechanical gate on devices such as ATMs.
- There shall be a 40 column dot matrix receipt printer.
- There shall be a 40 column dot matrix statement printer.
- The receipt dispenser shall be a maximum of 4" width and 0.5" thickness.
- The statement dispenser shall be a maximum of 5" width and 0.5" thickness.
- The envelope depository shall be a maximum of 4.5" width, 10" length and 0.5" thickness.
- Screen resolution of at least 800 × 600-required for proper and complete viewing of screens. Higher resolution would not be a problem.

Software Interface Requirements

In order to perform various different functions, this software needs to interact with various other software's. So there are certain software interface requirements that need to be fulfilled which are listed as follows:-

- The transaction management software used to manage the transaction and keep track of resources shall be BMS version 2.0.
- The card management software used to verify pin no and login shall be CMS version 3.0.
- Yamaha codecs 367/98 for active speakers.
- The database used to keep record of user accounts shall be Oracle version 7.0.

Communication Interface Requirements

The machine needs to communicate with the main branch for each session for various functions such as login verification, account access etc. so the following are the various communication interface requirements that are needed to be fulfilled in order to run the software successfully:

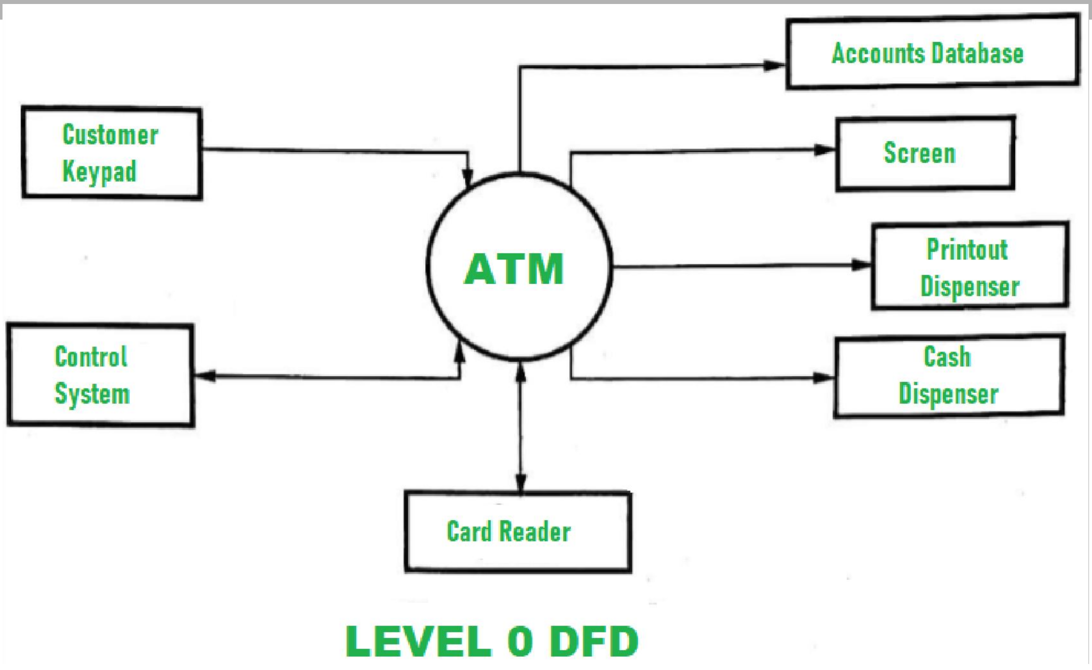
- The system will employ dial-up POS with the central server for low cost communication.
- The communication protocol used shall be TCP/IP.
- Protocol used for data transfer shall be File Transfer Protocol (FTP).

4. To perform the function-oriented diagram: DFD and Structured chart.

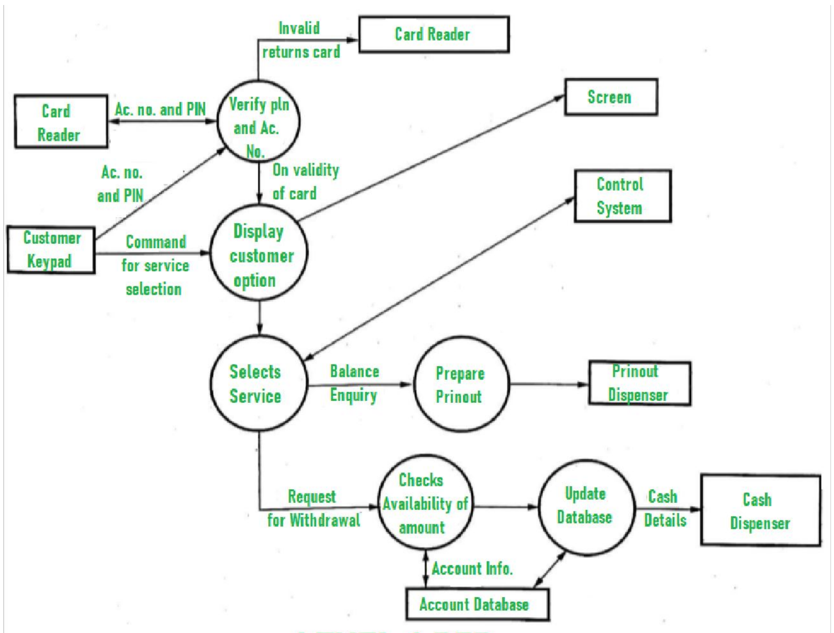
Ans :

DFD (Data Flow Diagram) of an ATM System consist of two levels of DFD. These levels are Level 0 DFD and Level 1 DFD. Both these levels are used for making the DFD of an ATM system.

- **Level 0 DFD:** This level is also known as Context Level DFD. At this level, only the interacting inputs and outputs with a system are described. The DFD of this level is shown below:



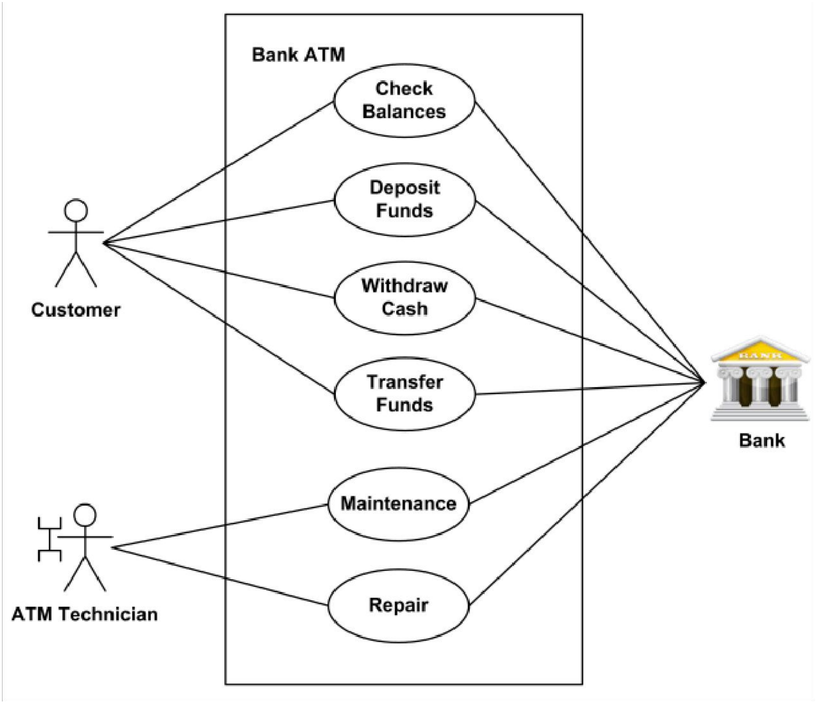
➢ **Level 1 DFD:** At this level, more detailed information is given about the processing of the ATM system. The DFD of this level is shown below:



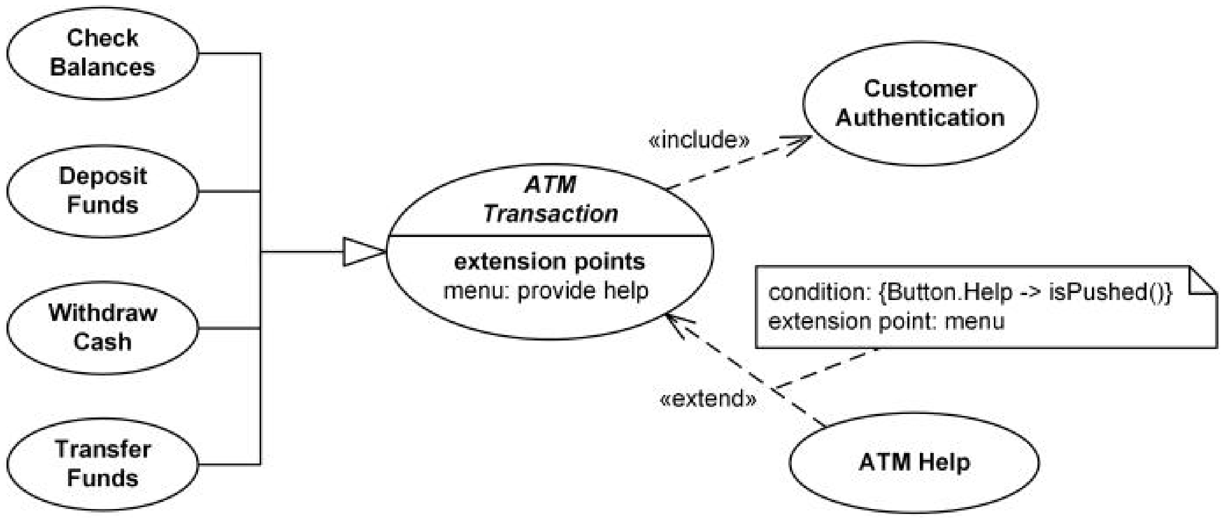
LEVEL 1 DFD

5. To perform the user's view analysis: Use case diagram

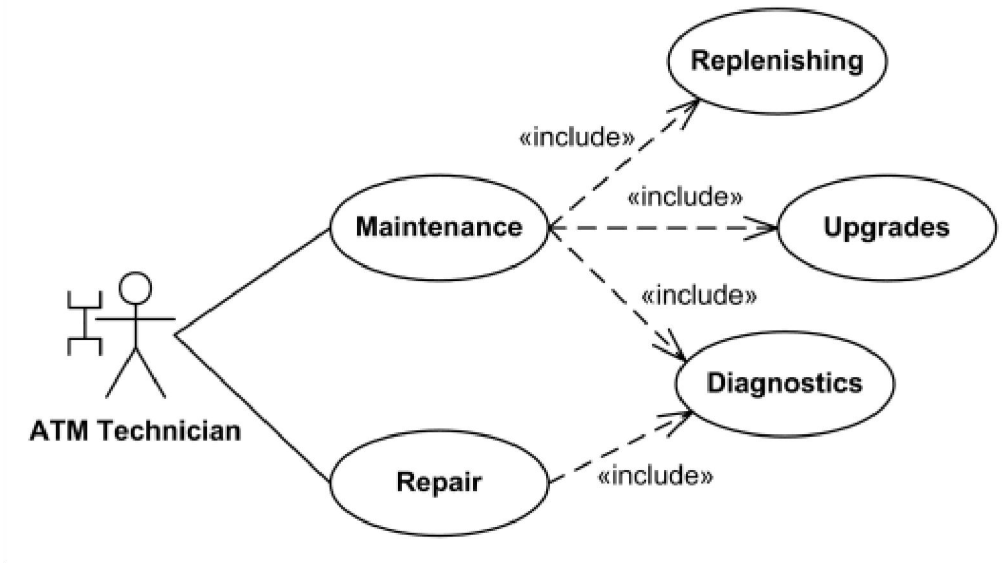
Customer (actor) uses bank ATM to Check Balances of his/her bank accounts, Deposit Funds, Withdraw Cash and/or Transfer Funds (use cases). ATM Technician provides. Maintenance and Repairs. All these use cases also involve Bank actor whether it is related to customer transactions or to the ATM servicing.



On most bank ATMs, the customer is authenticated by inserting a plastic ATM card and entering a personal identification number (PIN). Customer Authentication use case is required for every ATM transaction so we show it as include relationship. Including this use case as well as transaction generalizations make the ATM Transaction an abstract use case.



Customer may need some help from the ATM. ATM Transaction use case is extended via extension point called menu by the ATM Help use case whenever ATM Transaction is at the location specified by the menu and the bank customer requests help, e.g. by selecting Help menu item.



ATM Technician maintains or repairs Bank ATM. Maintenance use case includes Replenishing ATM with cash, ink or printer paper, Upgrades of hardware, firmware or software, and remote or on-site Diagnostics. Diagnostics is also included in (shared with) Repair use case.

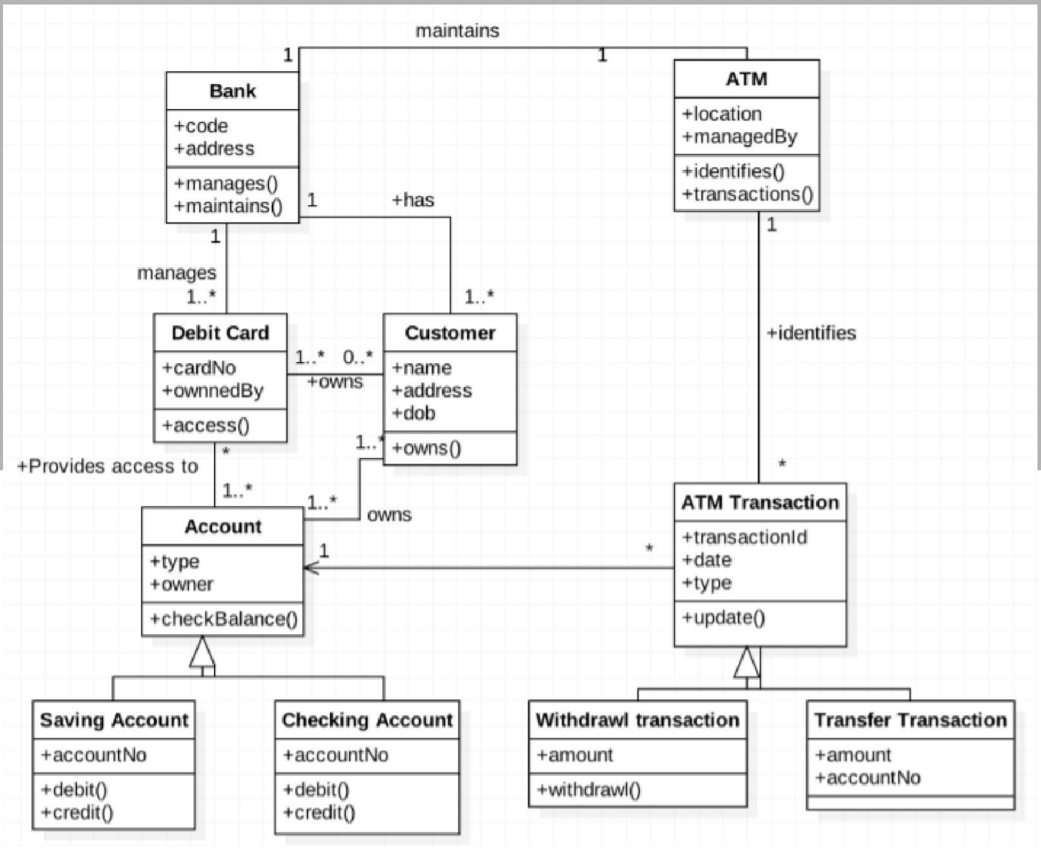
6. To draw the structural view diagram: Class diagram, object diagram.

Ans :

Class diagram

Class diagrams describe the static structure of a system, or how it is structured rather than how it behaves. These diagrams contain the following elements:

- **Classes:** which represent entities with common characteristics or features. These features include attributes, operations, and associations.
- **Associations:** which represent relationships that relate two or more other classes where the relationships have common characteristics or features. These features include attributes and operations.



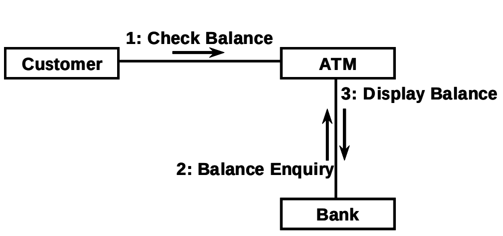
7. To draw the behavioural view diagram: Sequence diagram, Collaboration diagram.

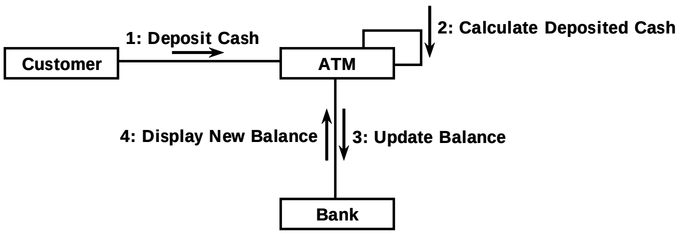
Ans :

Collaboration Diagrams

A Communication or Collaboration diagram, as shown is a directed graph that uses objects and actors as graph nodes. The focus of the collaboration diagram is on the roles of the objects as they interact to realize a system function. Directional links are used to indicate communication between objects.

These links are labelled using appropriate messages. Each message is prefixed with a sequence number indicating the time ordering needed to realize the system function.

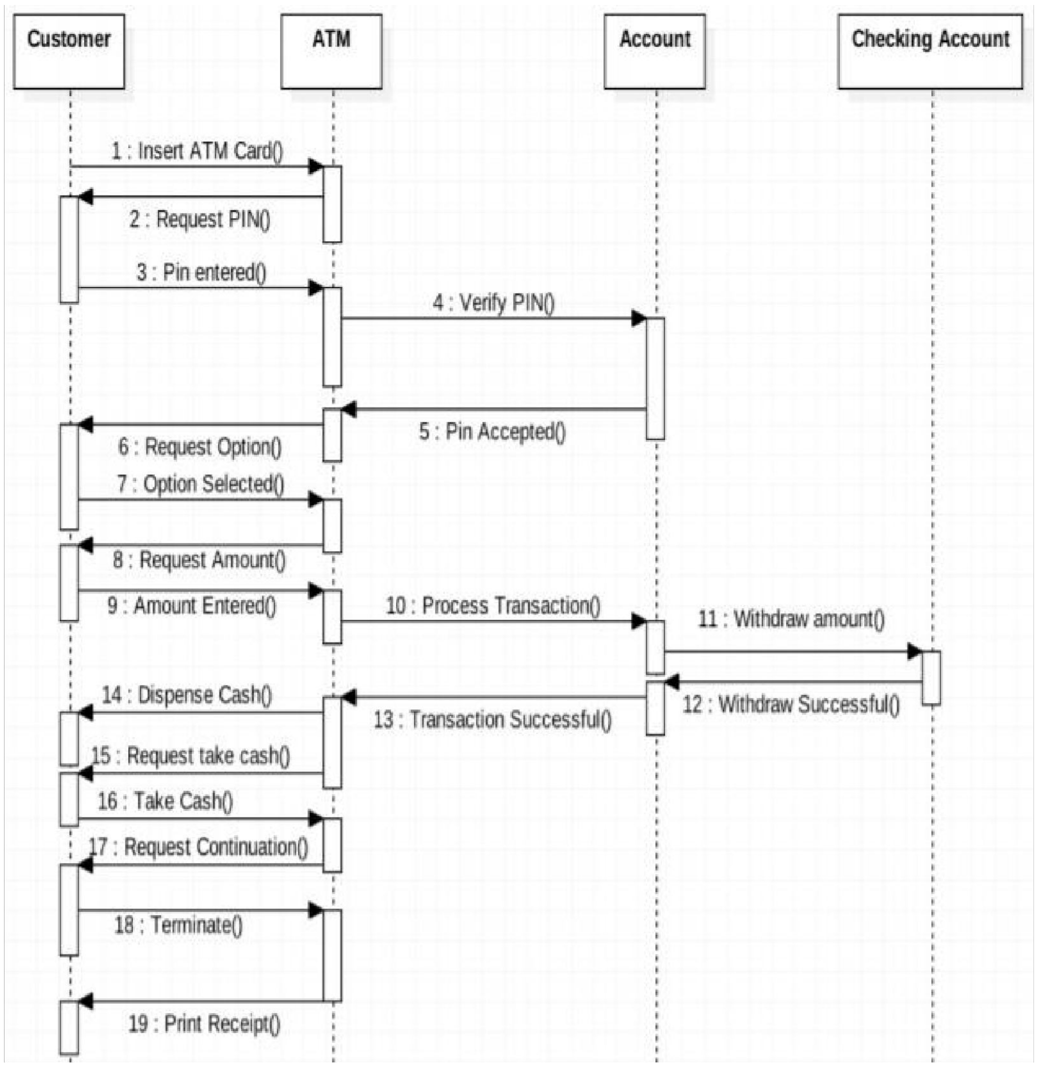




Sequence Diagram

Sequence diagrams typically show the flow of functionality through a use case, and consist of the following components:

- **Actors:** Involved in the functionality.
- **Objects:** That a system needs to provide the functionality.
- **Messages:** Which represent communication between objects.



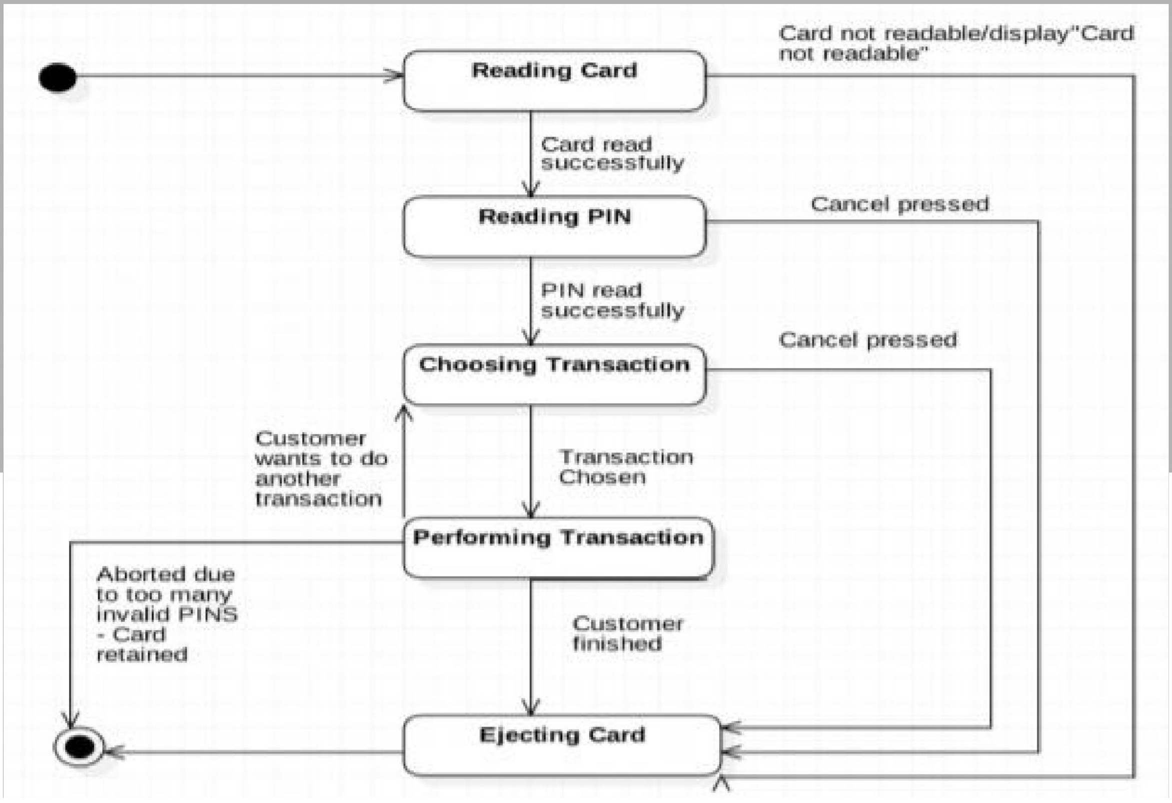
8. To draw the behavioural view diagram: State-chart diagram, Activity diagram.

Ans :

State Diagram

State transition diagrams provide a way to model the various states in which an object can exist. While the class diagram show a static picture of the classes and their relationships, state transition diagrams model the dynamic behaviour of a system in response to external events (stimuli). State transition diagrams consist of the following:

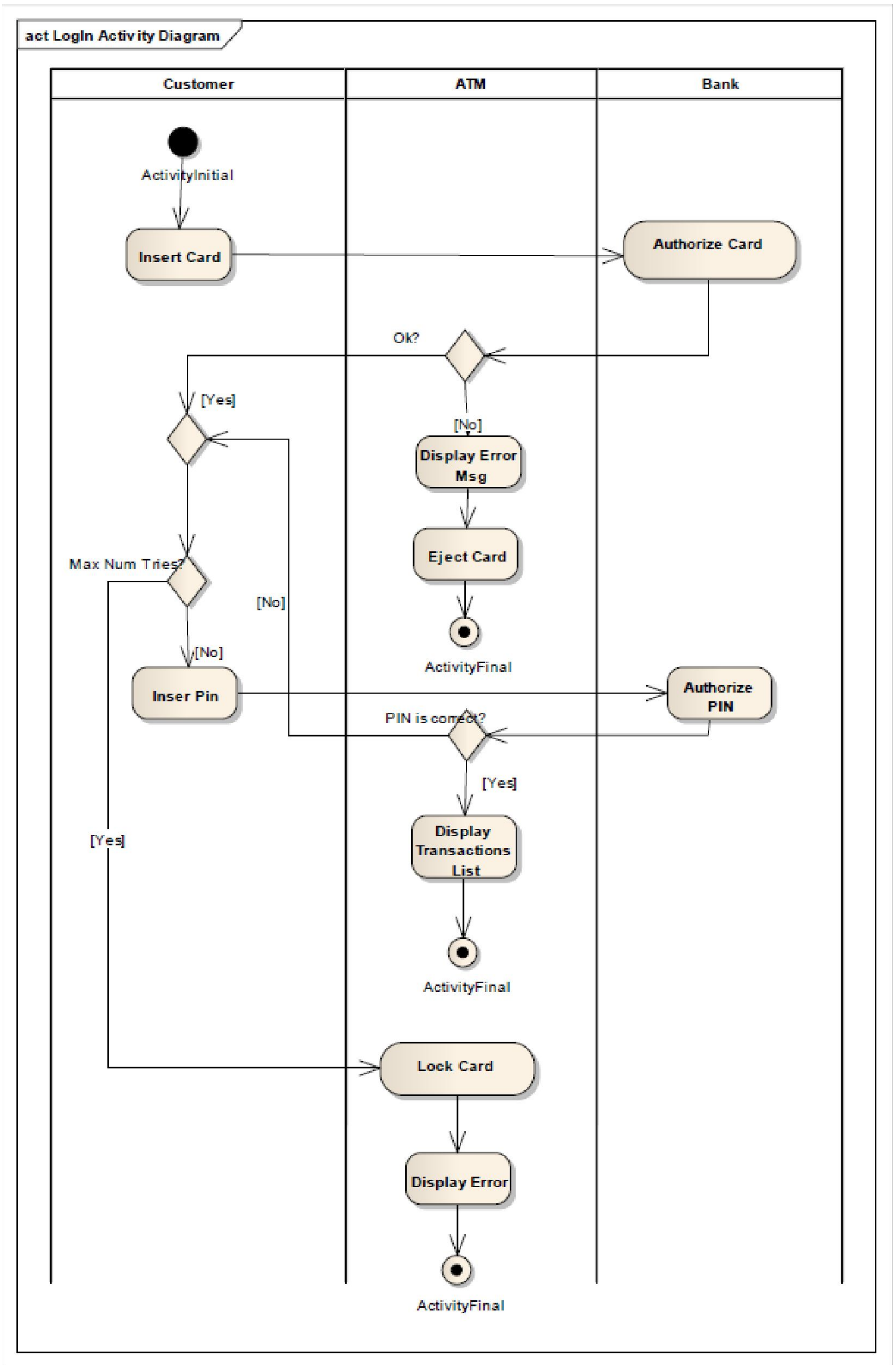
- **States:** Which show the possible situations in which an object can find itself
- **Transitions:** Which show the different events which cause a change in the state of an object?

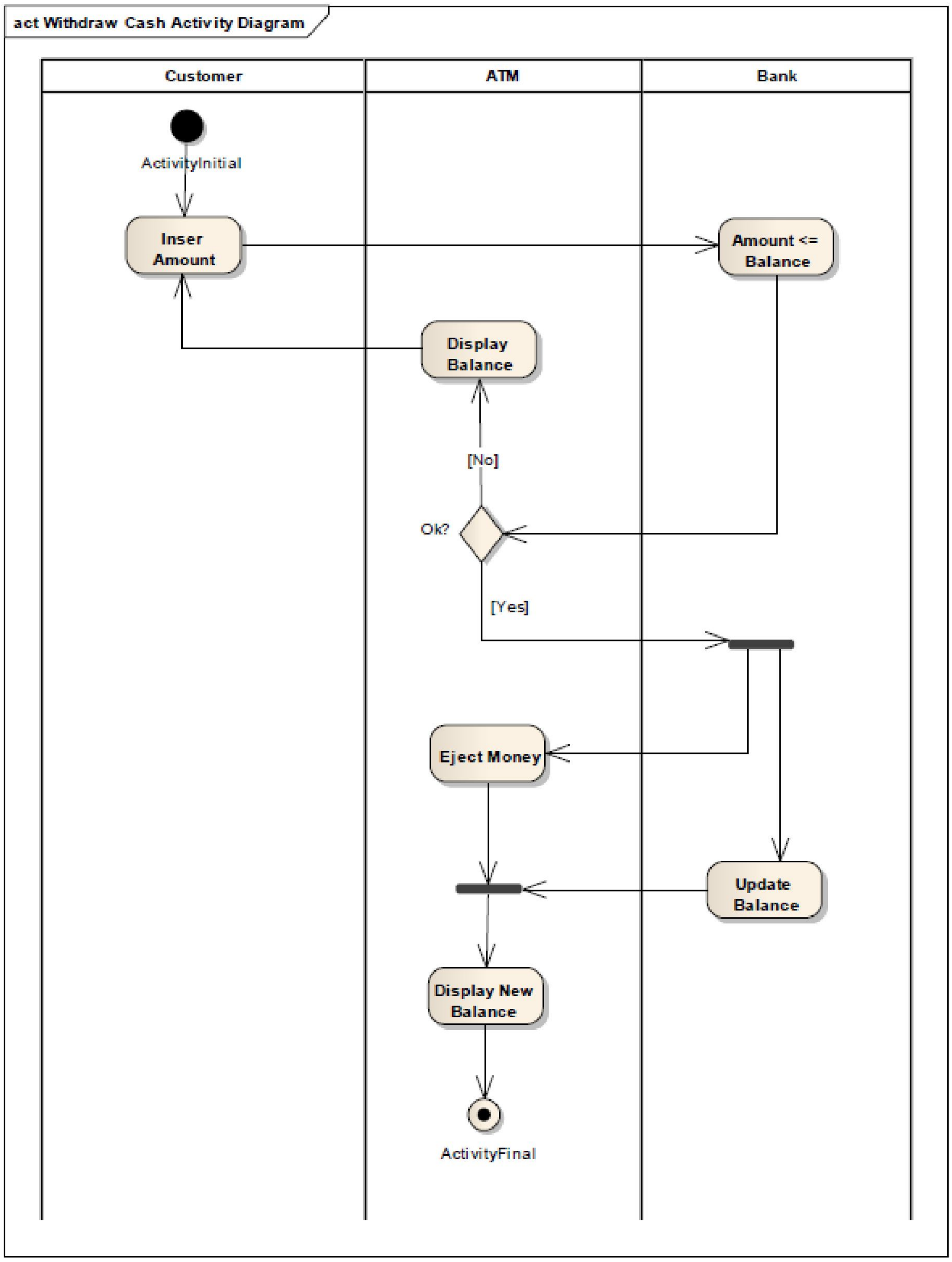


Activity Diagram

Activity diagrams describe the activities of a class. They are similar to state transition diagrams and use similar conventions, but activity diagrams describe the behavior/states of a class in response to internal processing rather than external events. They contain the following elements:

- **Swimlanes:** Which delegate specific actions to objects within an overall activity.
- **Action States:** Which represent uninterruptible actions of entities, or steps in the execution of an algorithm.
- **Action Flows:** Which represent relationships between the different action states on an entity.
- **Object Flows:** Which represent utilization of objects by action states, or influence of action states on objects.


act LogIn Activity Diagram
Customer
ATM
Bank
ActivityInitial
Insert Card
Authorize Card
Ok?
[Yes]
[No]
Display Error Msg
Eject Card
ActivityFinal
Max Num Tries?
[No]
[No]
Inser Pin
Authorize PIN
PIN is correct?
[Yes]
[Yes]
Display Transactions List
ActivityFinal
Lock Card
Display Error
ActivityFinal

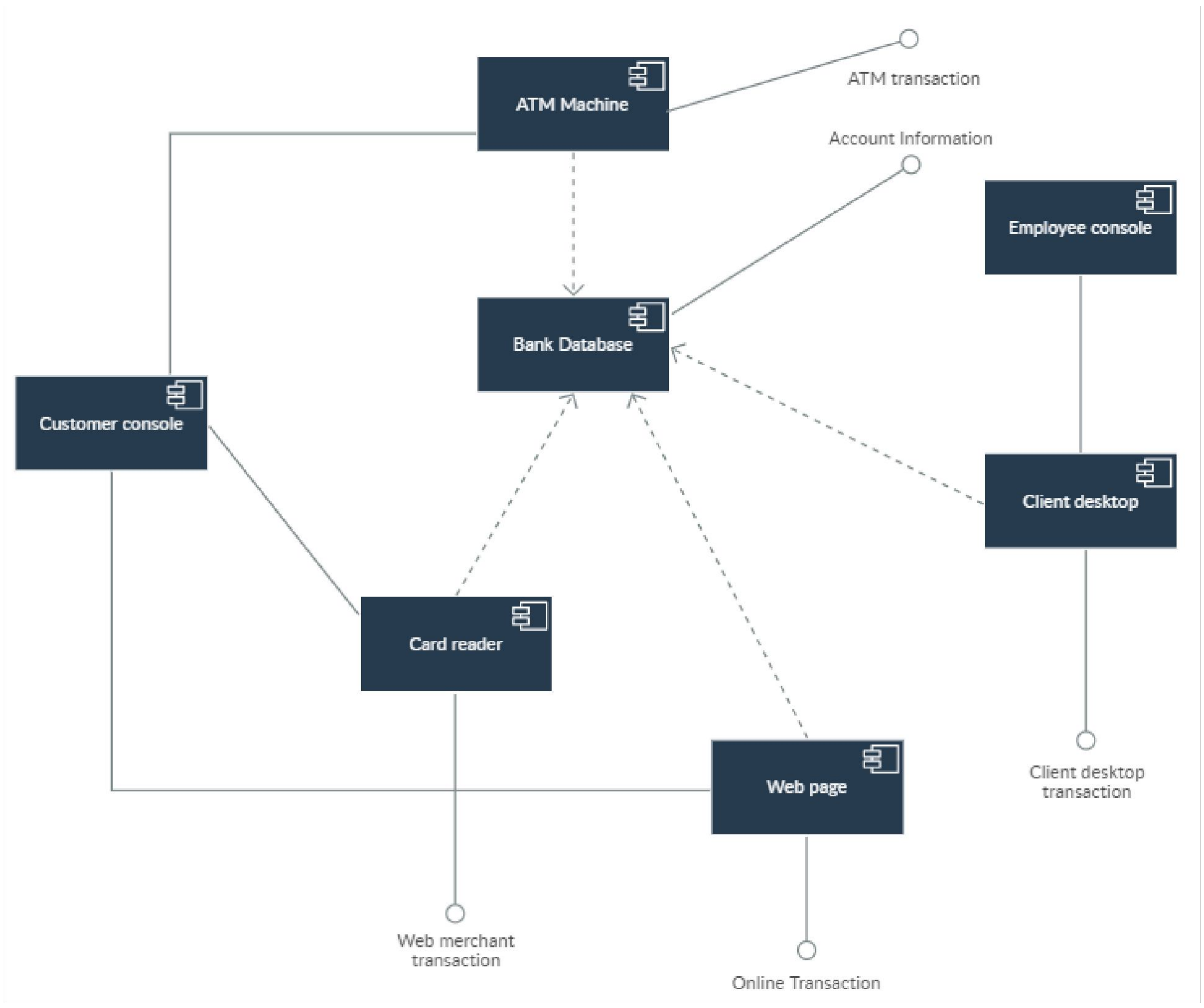

act Withdraw Cash Activity Diagram
Customer
ATM
Bank
ActivityInitial
Inser Amount
Amount <= Balance
Display Balance
[No]
Ok?
[Yes]
Eject Money
Update Balance
Display New Balance
ActivityFinal

9. To draw the implementation view diagram: Component diagram, Deployment diagram.

Ans :

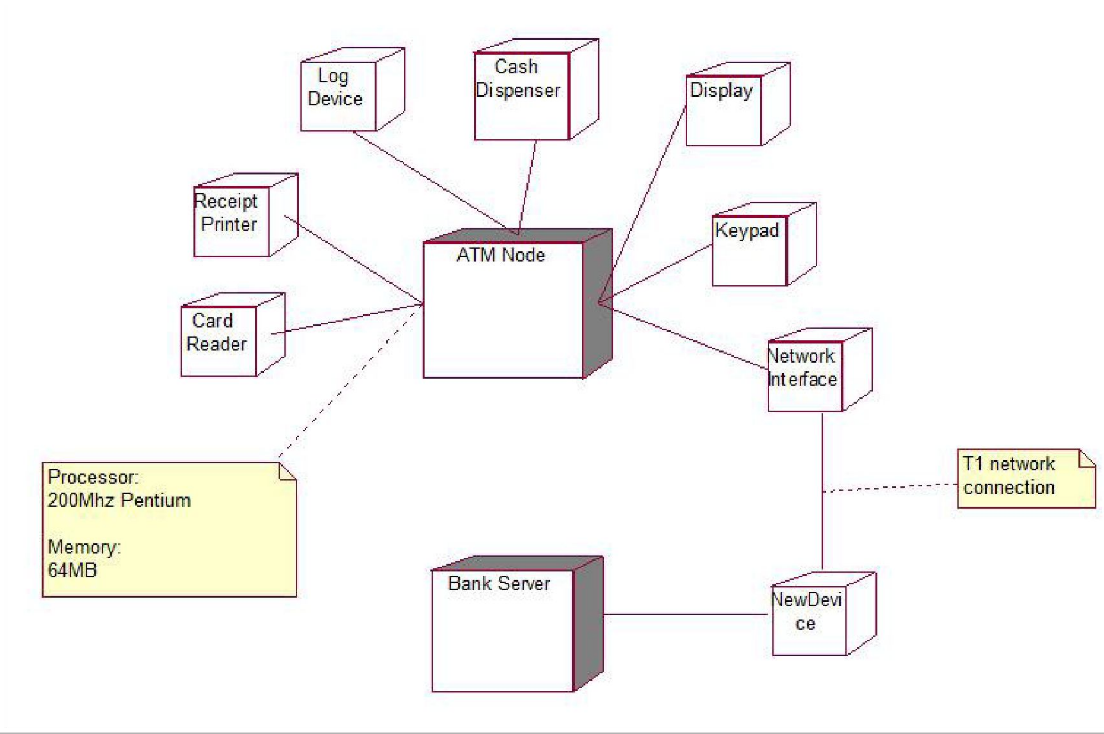
Component Diagram

A component diagram depicts how components are wired together to form larger components or software systems. This diagram illustrates the complex system in ATM.



Deployment Diagram

The UML Deployment Diagram is used for visualization of elements and components of a program, that exist at the stage of its execution. It contains graphical representations of processors, devices, processes, and relationships between them. The UML Deployment Diagram allows to determine the distribution of system components on its physical nodes, to show the physical connections between all system nodes at the stage of realization, to identify the system bottlenecks and reconfigure its topology to achieve the required performance. The UML Deployment diagram is typically developed jointly by systems analysts, network engineers and system engineers.



10. To perform various testing using the testing tool unit testing, integration testing

Ans :

Prerequisite – Types of Software Testing

Unit Testing

Unit Testing is a software testing technique by means of which individual units of software i.e. group of computer program modules, usage procedures and operating procedures are tested to determine whether they are suitable for use or not. It is a testing method using which every independent modules are tested to determine if there are any issue by the developer himself. It is correlated with functional correctness of the independent modules.

Unit Testing is defined as a type of software testing where individual components of a software are tested.

Unit Testing of software product is carried out during the development of an application. An individual component may be either an individual function or a procedure. Unit Testing is typically performed by the developer.

In SDLC or V Model, Unit testing is first level of testing done before integration testing. Unit testing is such type of testing technique that is usually performed by the developers. Although due to reluctance of developers to tests, quality assurance engineers also do unit testing.

Objective of Unit Testing:

The objective of Unit Testing is:

1. To isolate a section of code.
2. To verify the correctness of code.
3. To test every function and procedure.

4. To fix bug early in development cycle and to save costs.
5. To help the developers to understand the code base and enable them to make changes quickly.
6. To help for code reuse.

Types of Unit Testing

There are 2 type of Unit Testing: Manual, and Automated.

Test Cases for ATM

Given below are the various test cases for ATM.

1. Verify if the card reader is working correctly. A screen should ask you to insert the pin after inserting the valid card.
2. Verify if the cash dispenser is working as expected.
3. Verify if the receipt printer is working correctly. Which means it can print the data on the paper and the paper comes out properly.
4. Verify if the Screen buttons are working correctly. For touch screen: Verify if it is operational and working as per the expectations.
5. Verify if the text on the screen button is visible clearly.
6. Verify the font of the text on the screen buttons.
7. Verify each number button on the Keypad.
8. Verify the functionality of the Cancel button on the Keypad.
9. Verify the text color of the keypad buttons. The numbers should be visible clearly.
10. Verify the text color and font of the data on the screen. The user should be able to read it clearly.
11. Verify the language selection option. If the messages or data are displayed in the selected language.
12. Insert the card, the correct pin, and print the receipt for available balance.
13. Verify the receipt printing functionality after a valid transaction. Whether the printed data is correct or not.
14. Verify how much time the system takes to log out.
15. Verify the timeout session functionality.
16. Verify the deposit slot functionality depending on its capability (Cash or cheque or both) by inserting a valid cheque.
17. Verify using different cards (Cards of different banks).

Verifying the Message

18. Insert the card and an incorrect PIN to verify the message.
19. Verify the message when there is no cash in the ATM.
20. Verify the messages after a transaction.
21. Verify if a user will get a correct message if a card is inserted incorrectly.

Messages for each and every scenario should be verified.

Cash Withdrawal

22. Verify the cash withdrawal functionality by inserting some valid amount.
23. Verify if a user can perform only one cash withdrawal transaction per PIN insert.
24. Verify the different combinations of operation and check if there will be a power loss in the middle of the operation.

Negative Test cases

25. Verify the functionality by entering a wrong pin number for 3 or more times.
26. Verify the card reader functionality by inserting an expired card.
27. Verify the deposit slot functionality by inserting an invalid cheque.
28. Verify the cash withdrawal functionality by inserting invalid numbers like 10, 20, 50 etc.
29. Verify the cash withdrawal functionality by entering an amount greater than the per day limit,
30. Verify the cash withdrawal functionality by entering an amount greater than per transaction limit.
31. Verify the cash withdrawal functionality by entering an amount greater than the available balance in the account.

Integration Testing

Integration Testing is defined as a type of testing where software modules are integrated logically and tested as a group. A typical software project consists of multiple software modules, coded by different programmers. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated.

Integration Testing focuses on checking data communication amongst these modules. Hence it is also termed as 'I & T' (Integration and Testing), 'String Testing' and sometimes 'Thread Testing'.

Approaches, Strategies, Methodologies of Integration Testing

Software Engineering defines variety of strategies to execute Integration testing, viz.

- ➢ Big Bang Approach
- ➢ Incremental Approach: which is further divided into the following
- ➢ Top Down Approach
- ➢ Bottom Up Approach
- ➢ Sandwich Approach - Combination of Top Down and Bottom Up

ATM test cases may vary from one bank to another. Each back has its process. You can add test cases based on your companies requirement document.

Test Cases for ATM

1. Verify the 'ATM Card Insertion Slot' is as per the specification.
2. Verify the ATM machine accepts card and PIN details.
3. Verify the error message by inserting a card incorrectly.
4. Verify the error message by inserting an invalid card (Expired Card)
5. Verify the error message by entering an incorrect PIN

6. Verify that the user is asked to enter the PIN after inserting a valid ATM Card
7. Verify that PIN is encrypted
8. Verify that there is an action like blocking of card occurs when the total no. of incorrect PIN attempts get surpassed
9. Verify the user is allowed to do only one cash withdrawal transaction per PIN request
10. Verify the machine logs out of the user session immediately after successful withdrawal
11. Verify the message when there is no money in the ATM
12. Verify the language selection functionality
13. Verify the cash withdrawal functionality by entering some valid amount
14. Verify the cash withdrawal functionality by entering an amount less than 100
15. Verify the cash withdrawal functionality by entering an amount greater than the total available balance in the account.
16. Verify the cash withdrawal functionality by entering an amount greater than per day limit
17. Verify the user is allowed to enter the amount again in case the amount entered is not valid. A proper message should be displayed.
18. Verify the ATM machine successfully takes out the money.
19. Verify the ATM machine takes out the balance printout after the withdrawal
20. Verify the font of the text displayed in ATM screen
21. Verify the text on the screen buttons visible clearly.
22. Verify the functionality of all the buttons on the keypad
23. Verify the text on the buttons visible clearly.
24. Verify that touch of the ATM screen is smooth and operational
25. Verify the user is allowed to choose different account types like Savings, Current etc.,
26. Verify the different combinations of operation and check if there will be an electricity loss in the middle of the operation. If there is an electricity loss in the middle of the transaction then the transaction should be marked as null and the amount shouldn't be disclosed to others.
27. Verify the functionality of the cash dispenser
28. Verify the functionality of the receipt printer
29. Verify whether the printed data is correct or not in the receipt
30. Verify how much time the system takes to log out.

2. Library Management System

1. Phases in software development project, overview, need, coverage of topics

Ans :

SystemPlanning

The developing process of the Library Management System will be carried.

Project Aims andObjectives

The project aims and objectives that will be achieved after completion of the system were carried out. The succession of the system also will be evaluated.

The project objectives are :

- To eliminate the paper-work in library
- To record every transaction in computerized system so that problem such as record file missing won't happen again
- To implement BarCode, SMS technologies into thesystem
- To design a user friendly graphical user interface which suit theusers
- To complete the system according to project schedule
- To produce technical report that documents the phases, tasks and deliverables in the project

Background of Project

Library Management System is an application refer to other library system and it is suitable to use by small and medium size library. It is use by librarian and library admin to manage the library using a computerized system. The system was developed and designed to help librarian record every book transaction so that the problem such as file missing or record missing will not happened again.

Barcode reader is equipped in this system so that users can enjoy the convenience without need to key in the barcode of the book themselves. It is convenience and time saving as the users can direct scan in the book's barcode id when the members borrows few books in one time.

Book and member maintenance module also included in Library Management System. Users can register or edit the member or book in the system. With this computerized maintenance, library will not lost the book record or member record which always happen when no computerized system bring used.

Project Scope

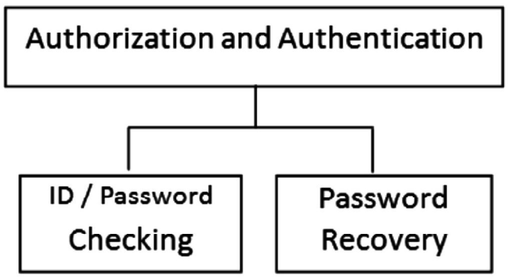
Project scope will carried out what modules were contains inside the Library Management System.

2. To assign the requirement engineering tasks

Ans :

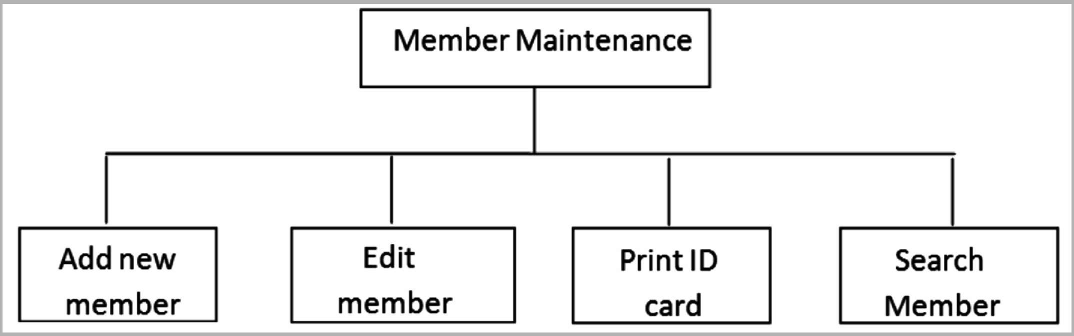
Library system

Authorization and AuthenticationModule



This module is used by user which means librarian in the library. They need to login to the system using their id and password. In order to distinguish the user's level, user can access to different module when successfully login. For example, only admin level users are able to access the report module.

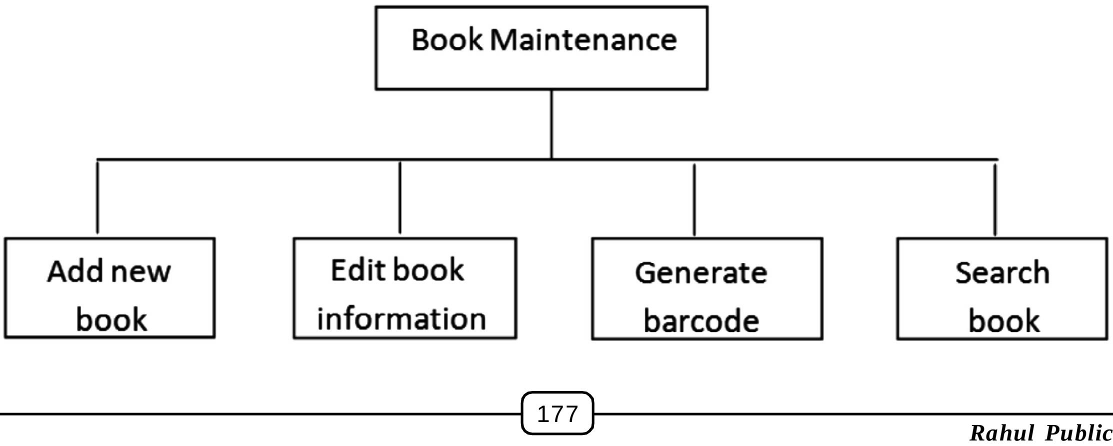
Member Maintenance Module



This module can be accessed by either librarian or library admin to maintain member's profile or record such as search, add, edit and print ID card.

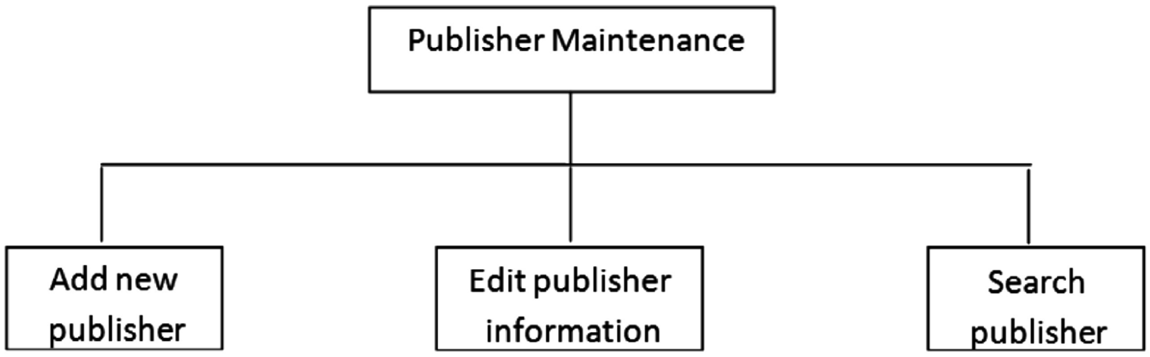
Book Maintenance Module

Book Module can access by any user from all levels. This module can used to maintain the book inventory record such as search, add and edit. In addition, we can generate the barcode for particular book and print it out so that librarian can stick the barcode on the book cover.

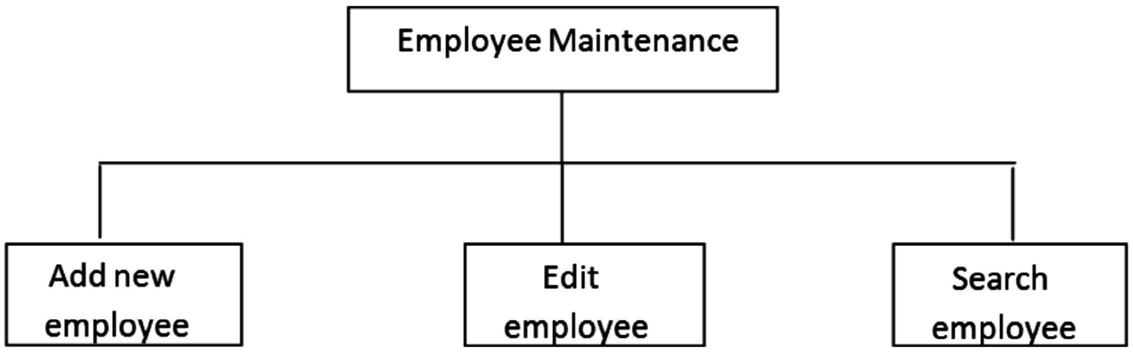


Publisher MaintenanceModule

This module allows user to add and edit the book's publisher. Publisher is used when register a new book.

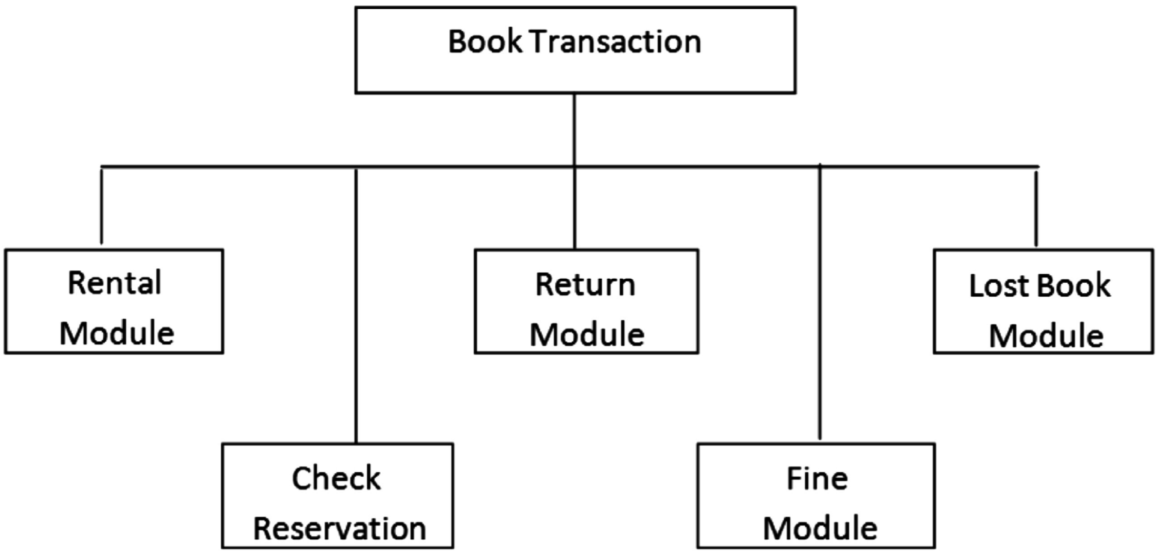


Employee Maintenance Module



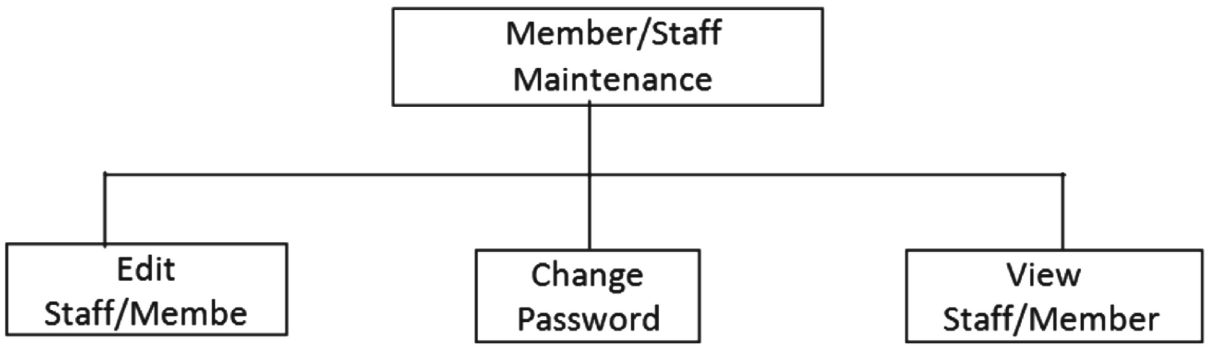
Employee Maintenance is only can carried out by admin level user. It can use to add the new librarian to the library which means add the new user.

Book Transaction Module

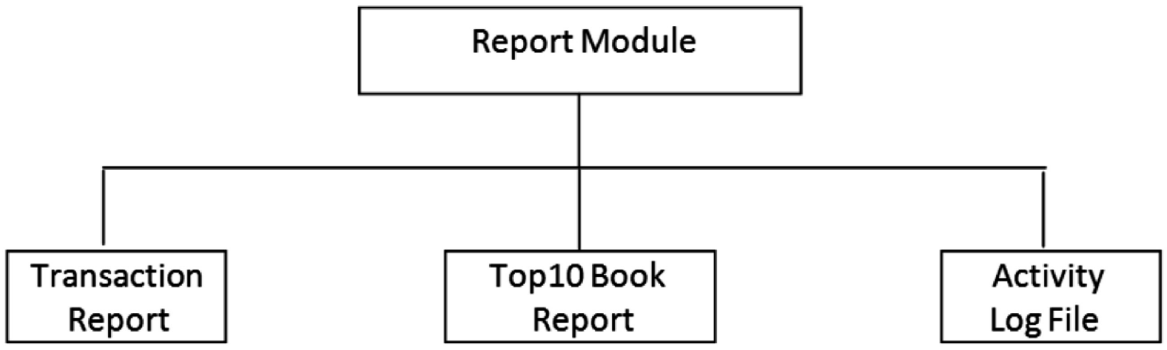


Book Transaction module is a main module in Library Management System. When member wants to borrow books, return books or they want to register lost book, it is all under book Transaction module. This module can be accessed by normal user or admin user. When member wants to borrow a book, librarian needs to scan in their member id. After that, librarian will scan their book's barcode id. If the book is under reservation, the book is not available to rent.

For return module, librarian just needs to scan the book's barcode id, and confirm the rental detail with user. If the rental detail is correct, return module can be complete if no any fine issued.



Report Module



Report module is the main module for admin user. It is because normal user is not allowed to view the report. The report divided into 3 types. First one is transaction report which can let admin views the book transaction happen on particular date such as rental report and return report.

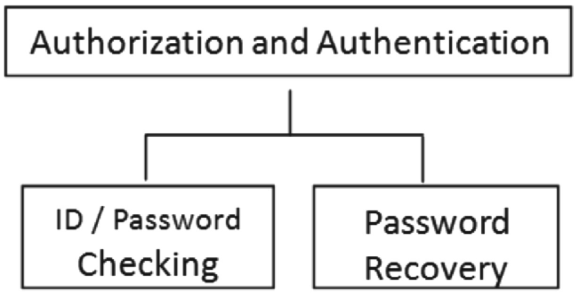
Top10 Report is the top rental rate's book. Admin can filter the information based on book's category and also filter by date in type of daily, monthly and yearly.

Activity Log File is a log which records every process in the Library Management System such as login / logout activity, register new book, new member or edit information or a member. All the activity done by every user will be record so that when system crash, admin or system admin are able to check the activity that may crash the system.

Library Web Site

At here, I will describe my friend's part which is library website which used by librarian and member.

Authorization and Authentication Module



This module is used by user and also admin user for the website. They need to login to the website using their id and password. In order to distinguish the user's level, user can access to different module when successfully login. For example, admin can implement News.

Member/Staff Maintenance Module

This module allows user and also admin to view their profile. Not only that, they allow to edit their profile and also change their password.

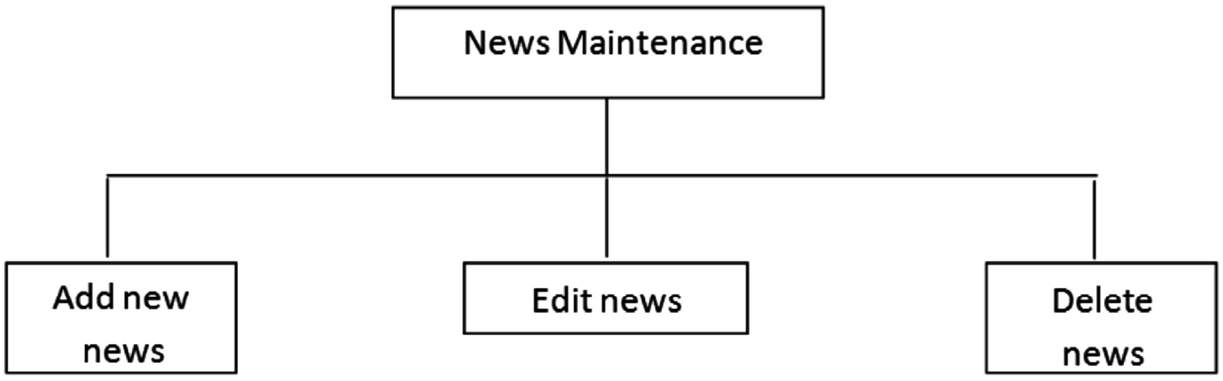
Search Module

Search module allow user or guess who visit website to search the book. Not only that the user allow viewing the detail of the book and also seeing the comment of the book. There are a few of type allow users to search. They can search via ISBN, book title, author, publisher, and category.

Top 10 new book and recommended book Module

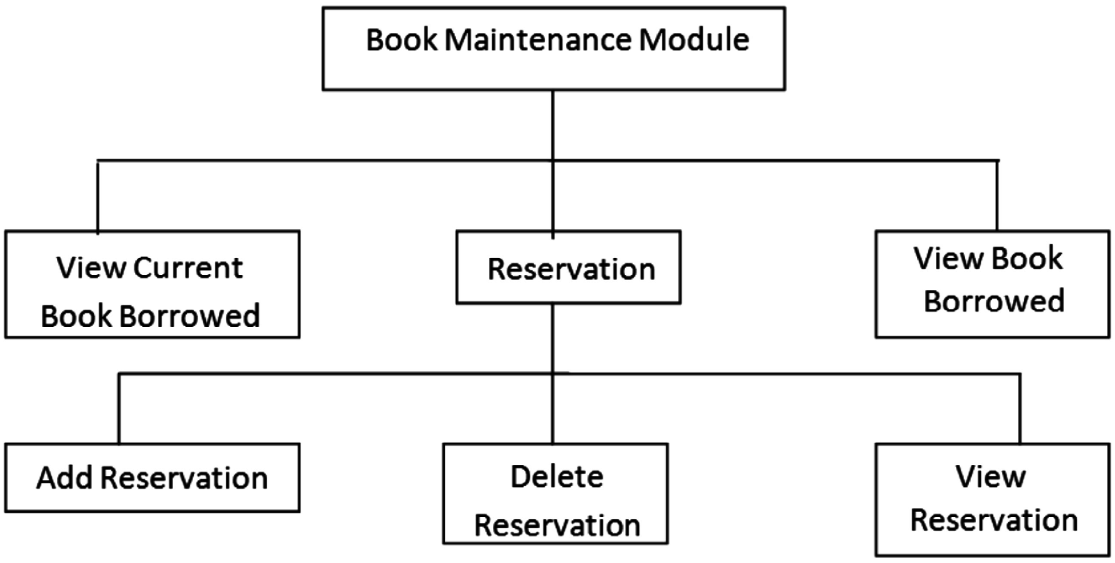
Top 10 new book modules allow guess and member to see the top 10 new book has brought by the library and the recommended book module only can see by member after log in to the system. It recommended based on their category example the user like to see action type book then it will recommend some action book for

News Maintenance Module



This module allows user to add, edit and delete the news. So when the users visit the website the website will have shown updated news.

Book Maintenance Module



Book maintenance module allows the member to view the book currently they have borrowed and also view the book they have borrowed. Not only that, there got reservation module which allow member to reserve book. Members are allowing reserving book and deleting the reservation and also view currently the book they reserve.

E-mailModule

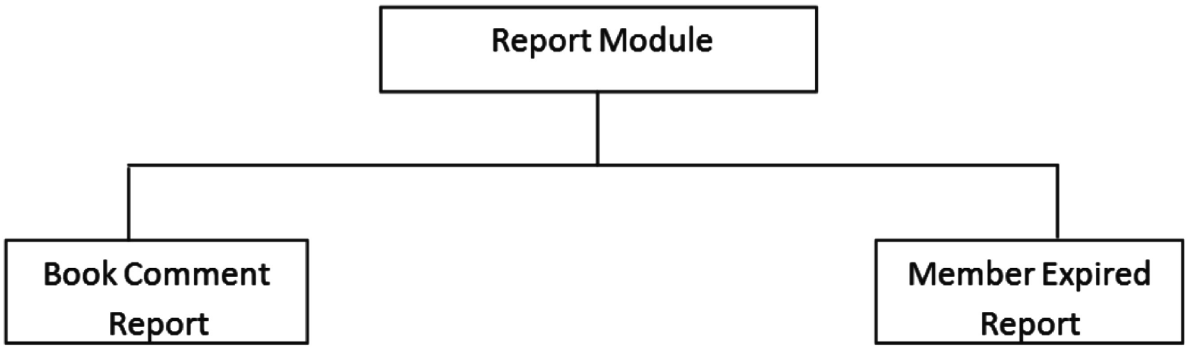
E-mail module has implemented in FAQ and also forget password. So when the member forgot their password they can get back their password via matching their ID and E-mail. Other than that, for FAQ if guess or members have more questions to ask they can ask via E-mail.

Report Module

Report module is the module for admin user. There is two report which is Book Comment and Member Expired.

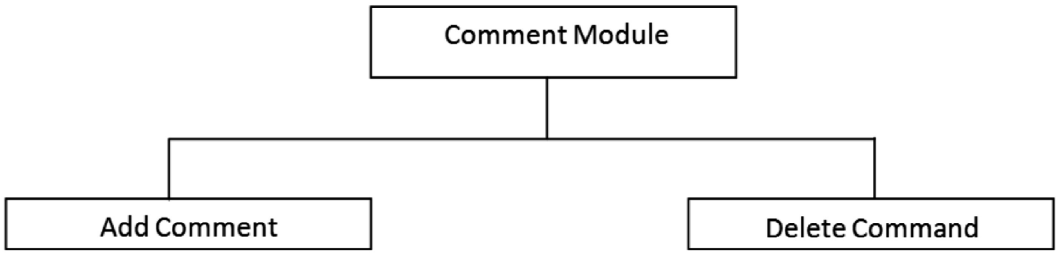
For Book Comment report is to view the top book that has been commented. It allows the admin to know the popularity of the book so that they can get more new books. It can view via day, month and year.

Member Expired report is to view the member is expired via which date, month or year. So they can remind their member to renew the membership and also can know the popularity of the library. So, it can be estimate the popularity of the library currently is increasing or decreasing.



CommentModule

Comment Module created is for member to comment a book. So, if the member have opinion about the book they can actually comment on the book. Not only that they can also delete their own comment. Admin users can delete any comment of the book that member has already comment.



3. To perform the system analysis: Requirement analysis, SRS

Ans :

Product Description

Library Management System is a computerized system which can helps user (librarian) to manage the library daily activity in electronic format. It reduces the risk of paper work such as file lost, file damaged and time-consuming. It can help user to manage the transaction or record more effectively and time-saving.

Problem Statement

The problem occurred before having computerized system includes:

File lost

When no computerizes system is implemented, the file always lost because of human and environment. Sometimes librarian didn't keep the record to its original place because of a lot member queue up to borrow books. After that the file was missing due to messyenvironment.

File damaged

In the other possibility, the file/record will be damaged due to accident. For example the librarian accidentally hit a glass of drink and pours onto the paper file. The record will be damaged. Besides this, natural disaster such as flood also will cause damage to the file record.

Difficult to search record

Without computerized system, when member wants to borrow a book, librarian hard to search for the member's record. It will cause time-consuming when a lot member are waiting to borrow the books.

Space-consuming

After long operation time of the library, the records are getting more and more. Finally, the physical record was space-consuming and no place to keep the file.

Difficult to viewreports

Report need to generate manually without computerizes system. Admin need to get the book transaction record and find the information based on the time period. It is time consuming to generate one report.

Cost consuming

Paper is needed to add every new record. After a long period of time, the cost to buy a paper can be high. On the other hand, library needs to employ more staff to solve the long queue problem. If the library only has one staff, it is not enough time to process the book transaction.

Systemobjectives

1. Improvement in control and performance

The system was developed to overcome the current problem occurred in library. The system must be able to validate the user, store the record and bug free.

2. Save cost

After implementing the computerized library system, library can only hire 1 or 2 staff to handle the book transaction process. With the aids of computerized system, library can save the cost of hire employee and also save the paper-cost.

3. Savetime

Librarian is able to search the record in short time by pressing only few keys. Compare to previous time before implementing the system, librarian can save a lottime.

I) SystemRequirements (Non-functionalRequirements)

Products Requirements

1. Efficiencyrequirements

With the library management system, librarian should be able to process faster when they process book transaction. In addition, with the use of bar code scanner, librarian can avoid to type the book id one by one, bar code scanner enable librarian to scan the book id instantly.

2. Reliability requirements

The system must perform accurately towards member request. For example, when the librarian saves the edited profile detail, after they review their detail, the details must be change according to the latest details that they have updated. When member return the book after the expired date, the fine should be calculate accurately. Besides that, in the registration form, it will have validity check to check the input to prevent wrong data type.

3. Usability requirements

This system must be designed with user-friendly and easy to use by the staff so that the user can perform their job nicely. It must have a clear instruction to guide user through the system. Besides that, the description of error message should be clear.

II) Organizational Requirements(Implementationrequirements)

In implementing the system, it uses the vb.net as the main programming language and tools. Besides that, the SQL language will be use to maintain the information in the database. On the other hand, SQL Server 2008 needs to be installed.

1. Delivery requirements

The whole system is estimate to be done around 6 months time and the documentation will be done in 2 months. The full system will be delivers in a softcopy method while the documentation will be delivers in hardcopy and softcopy.

III) External Requirements

1. Legislativerequirements

The information that use must be acknowledge by the authorized people so that it has no violating the law. This information is copyrighted and protected by the law. Besides that, when visitors wants to become the member of the library, he or she must agree with the rules stated in the system.

2. Security requirements

This system must be highly secure in the login part. It is because the report can only view by admin level. Staff can perform most of the process except viewing report module and log file module.

4. To perform the function-oriented diagram: DFD and Structured chart

Ans :

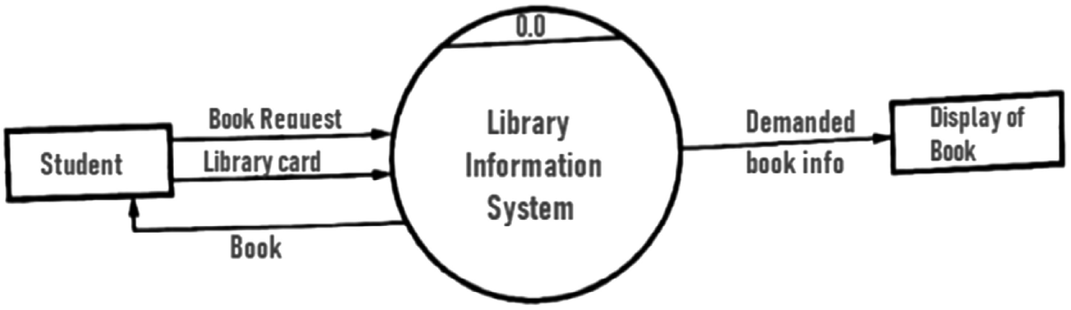
Data Flow Diagram (DFD) depicts the flow of information and the transformation applied when a data moves in and out from a system. The overall system is represented and described using input, processing and output in the DFD. The inputs can be:

- **Book request** when a student requests for a book.
- **Library card** when the student has to show or submit his/her identity as a proof.

The overall processing unit will contain the following output that a system will produce or generate:

- Book will be the output as the book demanded by the student will be given to them.
- Information of demanded book should be displayed by the library information system that can be used by the student while selecting the book which makes it easier for the student.

Level 0 DFD –



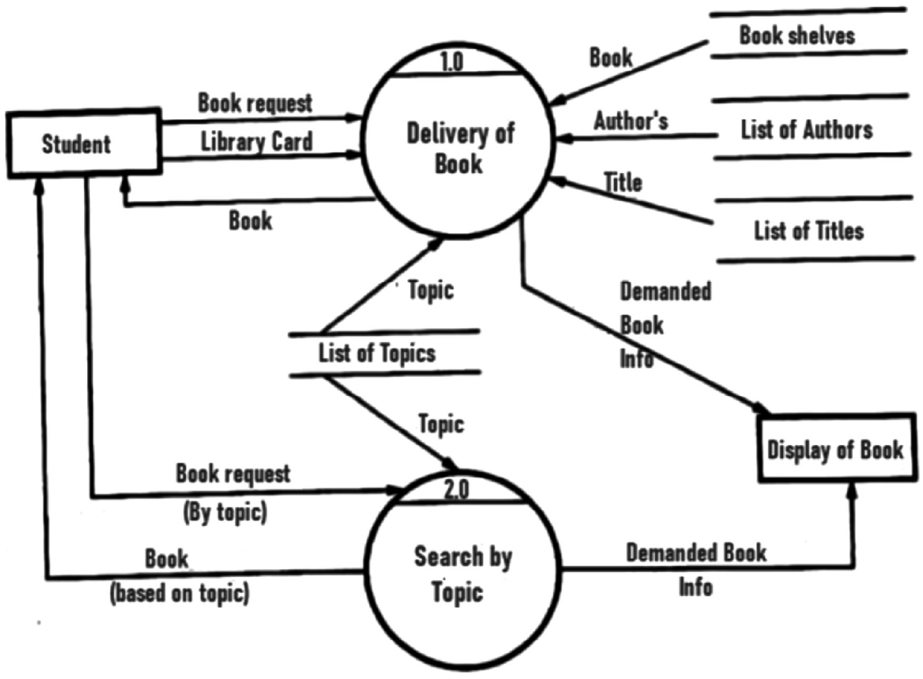
Level 1 DFD –

At this level, the system has to show or exposed with more details of processing.

The processes that are important to be carried out are:

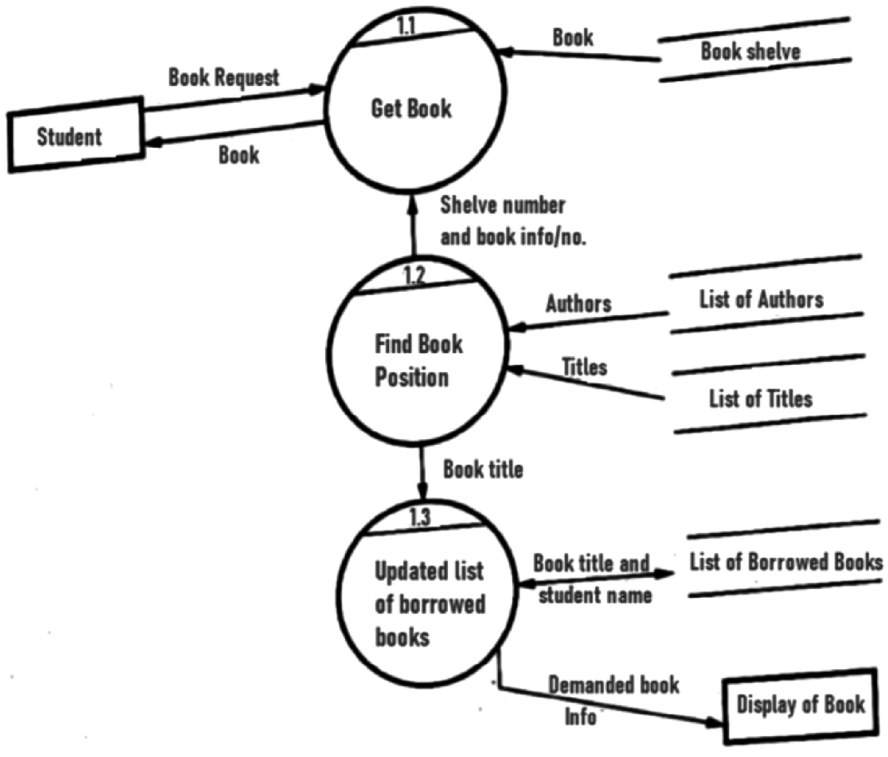
- Book delivery
- Search by topic

List of authors, List of Titles, List of Topics, the bookshelves from which books can be located are some information that is required for these processes. **Data store** is used to represent this type of information.



Level 1 DFD

Level 2 DFD –



Level 2 DFD

5. To perform the user's view analysis: Use case diagram

Ans :

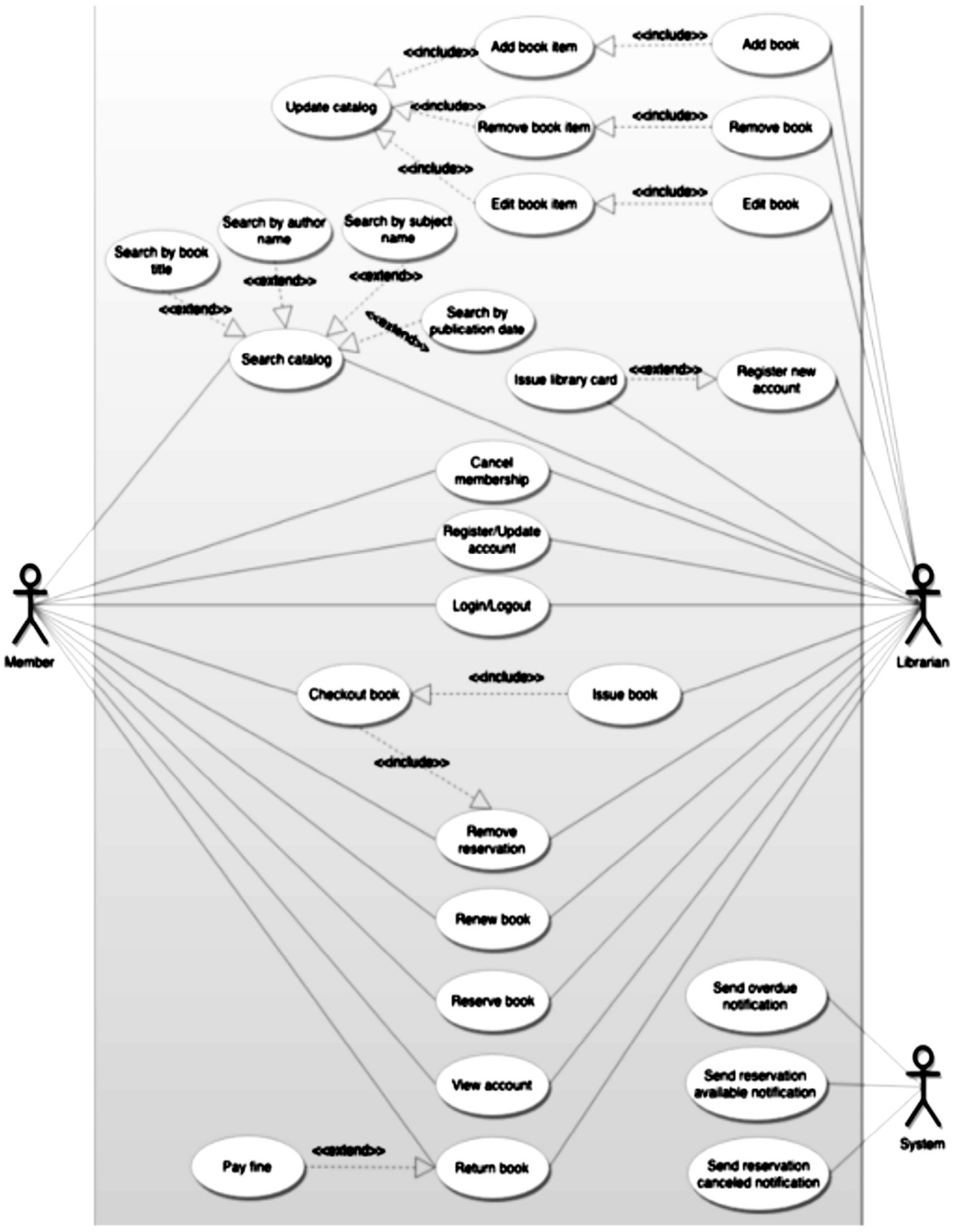
Use Case Diagram Library System Project

We have three main actors in our system:

- **Librarian:** Mainly responsible for adding and modifying books, book items, and users. The Librarian can also issue, reserve, and return book items.
- **Member:** All members can search the catalog, as well as check-out, reserve, renew, and return a book.
- **System:** Mainly responsible for sending notifications for overdue books, canceled reservations, etc.

Here are the top use cases of the Library Management System:

- **Add/Remove/Edit book:** To add, remove or modify a book or book item.
- **Search catalog:** To search books by title, author, subject or publication date.
- **Register new account/cancel membership:** To add a new member or cancel the membership of an existing member.
- **Check-out book:** To borrow a book from the library.
- **Reserve book:** To reserve a book which is not currently available.
- **Renew a book:** To reborrow an already checked-out book.
- **Return a book:** To return a book to the library which was issued to a member.



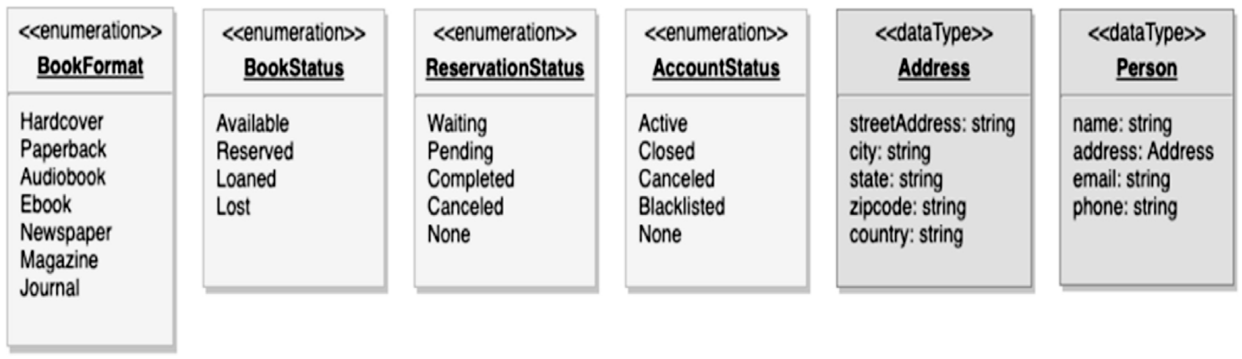
Add book item
<<include>>
Add book
<<include>>
Update catalog
<<include>>
Remove book item
<<include>>
Remove book
<<include>>
Edit book item
<<include>>
Edit book
Search by book title
Search by author name
Search by subject name
<<extend>>
<<extend>>
<<extend>>
Search by publication date
<<extend>>
Search catalog
Issue library card
<<extend>>
Register new account
Cancel membership
Register/Update account
Login/Logout
Member
Librarian
Checkout book
<<include>>
Issue book
<<include>>
Remove reservation
Renew book
Reserve book
Send overdue notification
View account
Send reservation available notification
System
Pay fine
<<extend>>
Return book
Send reservation canceled notification

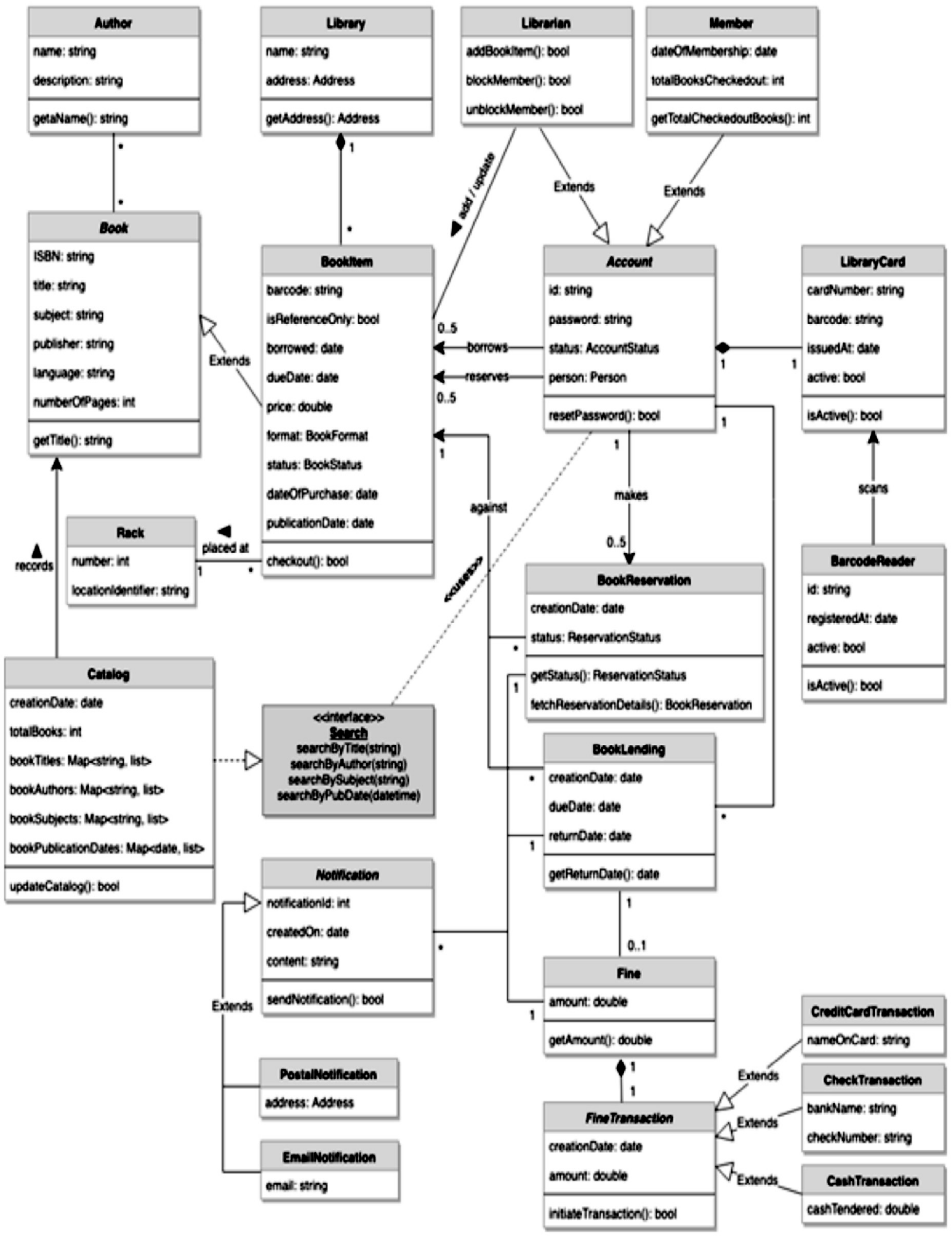
6. To draw the structural view diagram: Class diagram, object diagram

Ans :

Here are the main classes of our Library Management System:

- **Library:** The central part of the organization for which this software has been designed. It has attributes like 'Name' to distinguish it from any other libraries and 'Address' to describe its location.
- **Book:** The basic building block of the system. Every book will have ISBN, Title, Subject, Publishers, etc.
- **BookItem:** Any book can have multiple copies, each copy will be considered a book item in our system. Each book item will have a unique barcode.
- **Account:** We will have two types of accounts in the system, one will be a general member, and the other will be a librarian.
- **Library Card:** Each library user will be issued a library card, which will be used to identify users while issuing or returning books.
- **Book Reservation:** Responsible for managing reservations against book items.
- **BookLending:** Manage the checking-out of book items.
- **Catalog:** Catalogs contain list of books sorted on certain criteria. Our system will support searching through four catalogs: Title, Author, Subject, and Publish-date.
- **Fine:** This class will be responsible for calculating and collecting fines from library members.
- **Author:** This class will encapsulate a book author.
- **Rack:** Books will be placed on racks. Each rack will be identified by a rack number and will have a location identifier to describe the physical location of the rack in the library.
- **Notification:** This class will take care of sending notifications to library members.



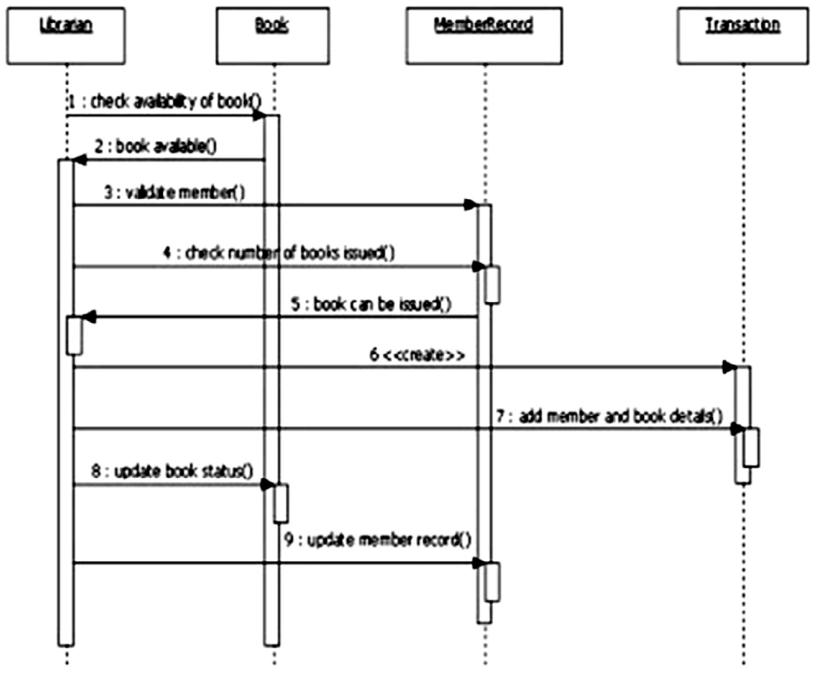

Author
name: string
description: string
getaName(): string
Library
name: string
address: Address
getAddress(): Address
Librarian
addBookItem(): bool
blockMember(): bool
unblockMember(): bool
Member
dateOfMembership: date
totalBooksCheckedout: int
getTotalCheckedoutBooks(): int
Book
ISBN: string
title: string
subject: string
publisher: string
language: string
numberOfPages: int
getTitle(): string
BookItem
barcode: string
isReferenceOnly: bool
borrowed: date
dueDate: date
price: double
format: BookFormat
status: BookStatus
dateOfPurchase: date
publicationDate: date
checkout(): bool
Account
id: string
password: string
status: AccountStatus
person: Person
resetPassword(): bool
LibraryCard
cardNumber: string
barcode: string
issuedAt: date
active: bool
isActive(): bool
BarcodeReader
id: string
registeredAt: date
active: bool
isActive(): bool
Rack
number: int
locationIdentifier: string
BookReservation
creationDate: date
status: ReservationStatus
getStatus(): ReservationStatus
fetchReservationDetails(): BookReservation
Catalog
creationDate: date
totalBooks: int
bookTitles: Map<string, list>
bookAuthors: Map<string, list>
bookSubjects: Map<string, list>
bookPublicationDates: Map<date, list>
updateCatalog(): bool
<<interface>>
Search
searchByTitle(string)
searchByAuthor(string)
searchBySubject(string)
searchByPubDate(datetime)
BookLending
creationDate: date
dueDate: date
returnDate: date
getReturnDate(): date
Notification
notificationId: int
createdOn: date
content: string
sendNotification(): bool
Fine
amount: double
getAmount(): double
PostalNotification
address: Address
EmailNotification
email: string
FineTransaction
creationDate: date
amount: double
initiateTransaction(): bool
CreditCardTransaction
nameOnCard: string
CheckTransaction
bankName: string
checkNumber: string
CashTransaction
cashTendered: double
add / update
Extends
borrows
reserves
makes
scans
against
placed at
records
<<uses>>
0..5
0..1
1
*

7. To draw the behavioural view diagram: Sequence diagram, Collaboration diagram

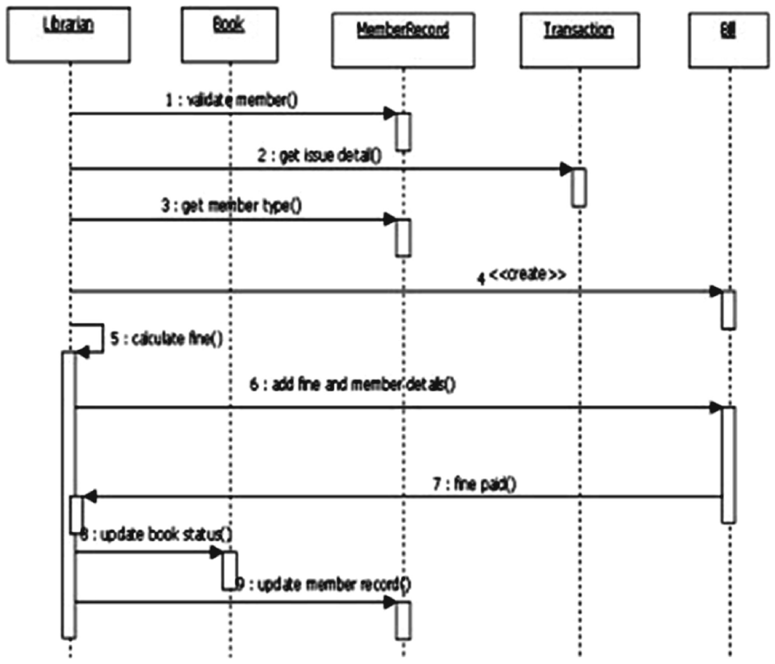
Ans :

Sequence diagram

Sequence diagram for issuing book:

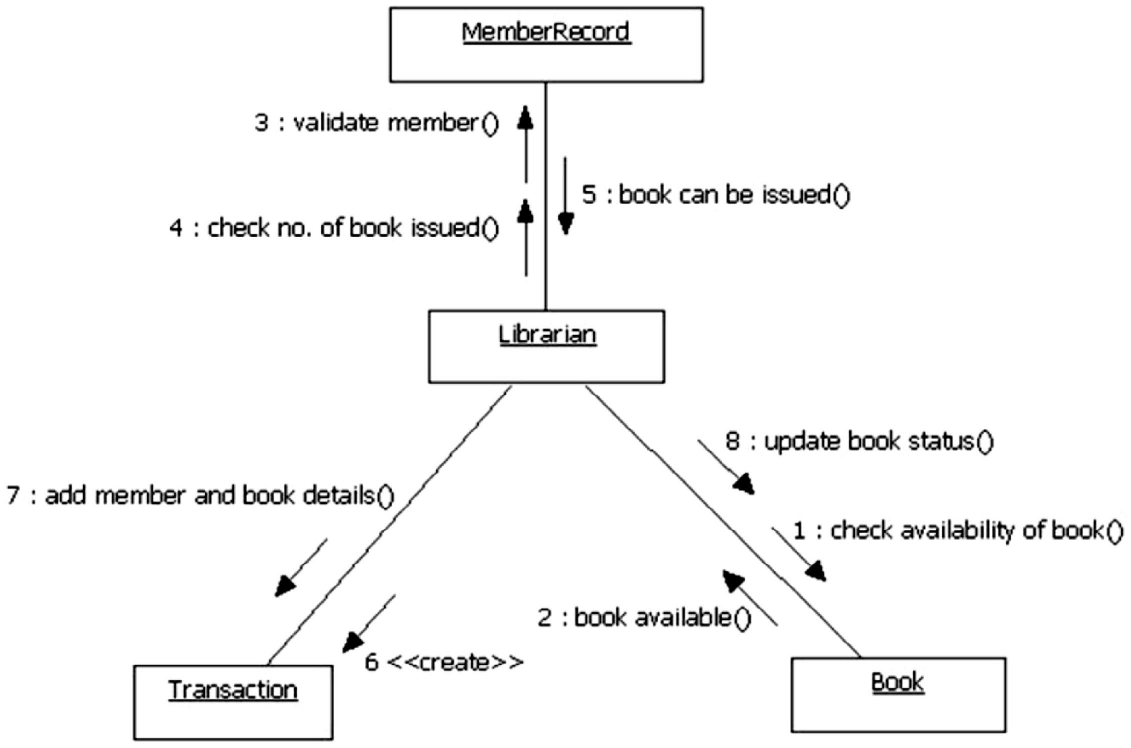


Sequence diagram for returning book:

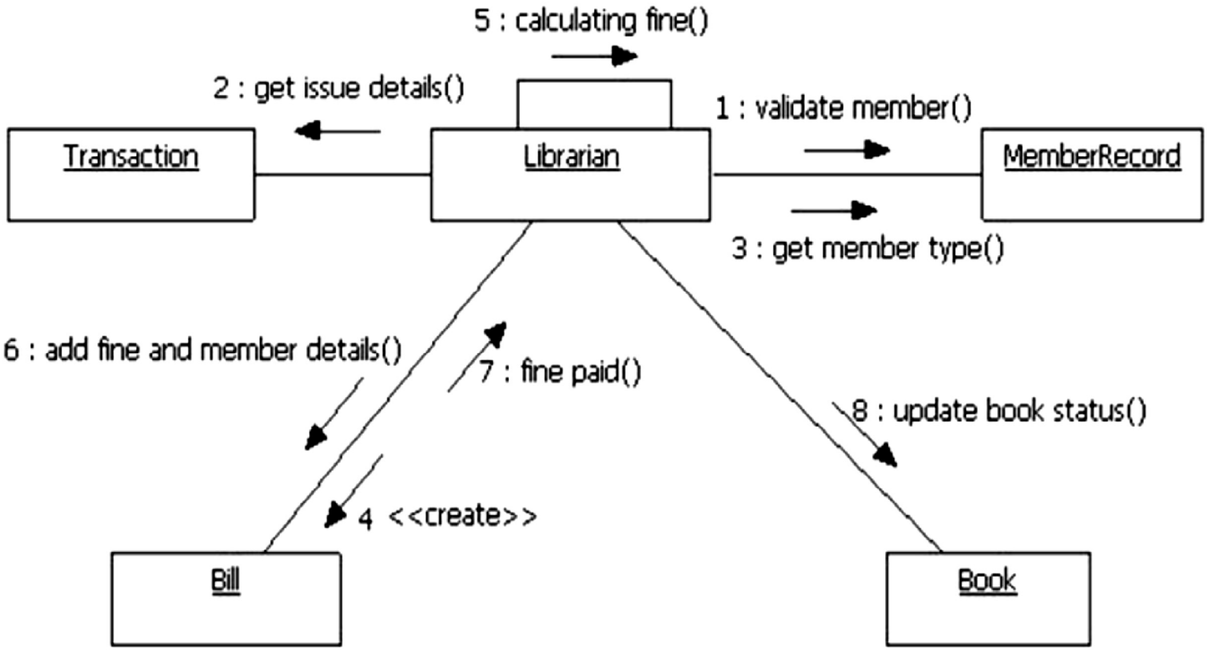


Collaboration Diagram

Collaboration diagram for issuing Book:



Collaboration diagram for returning Book:

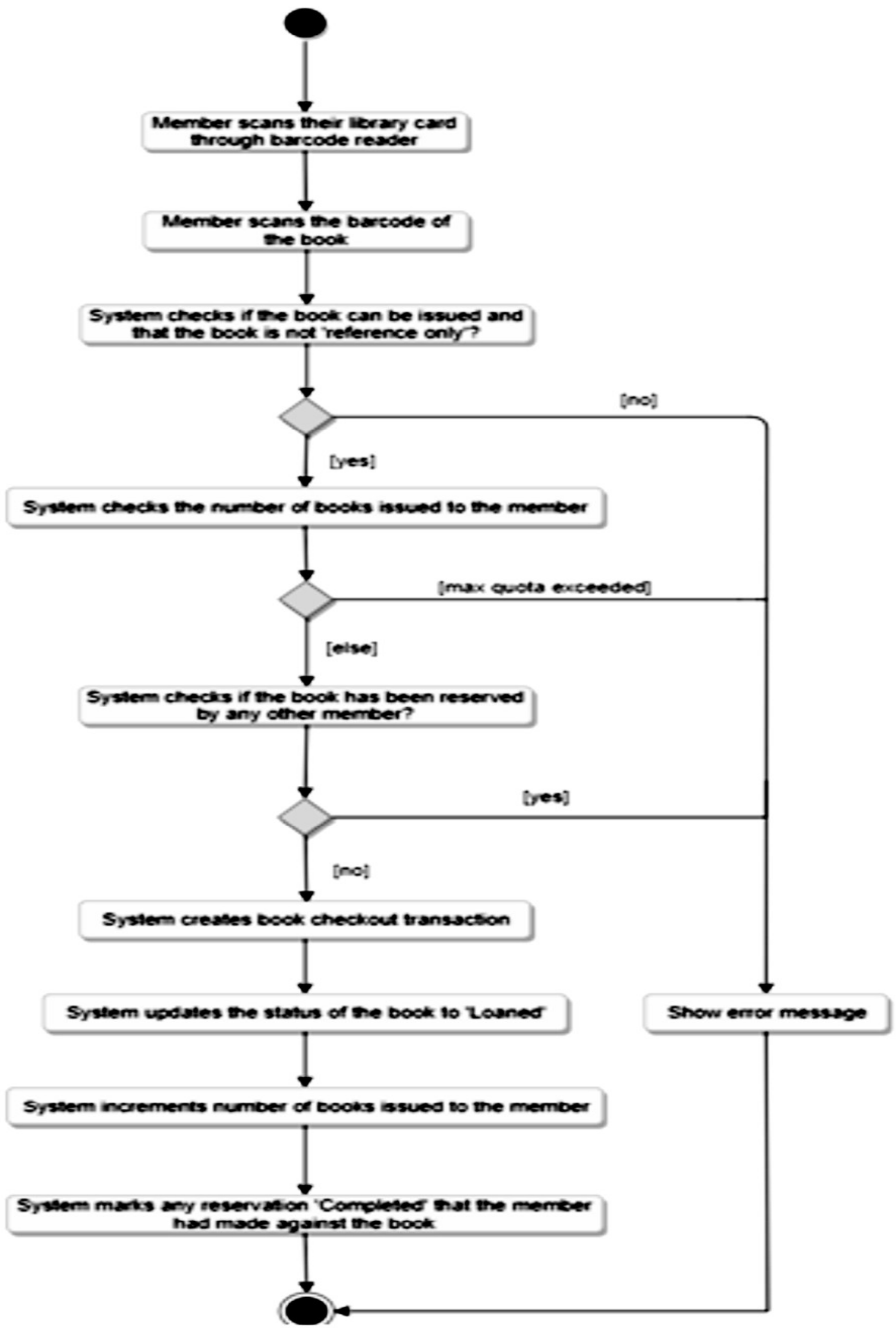


8. To draw the behavioural view diagram: State-chart diagram, Activity diagram

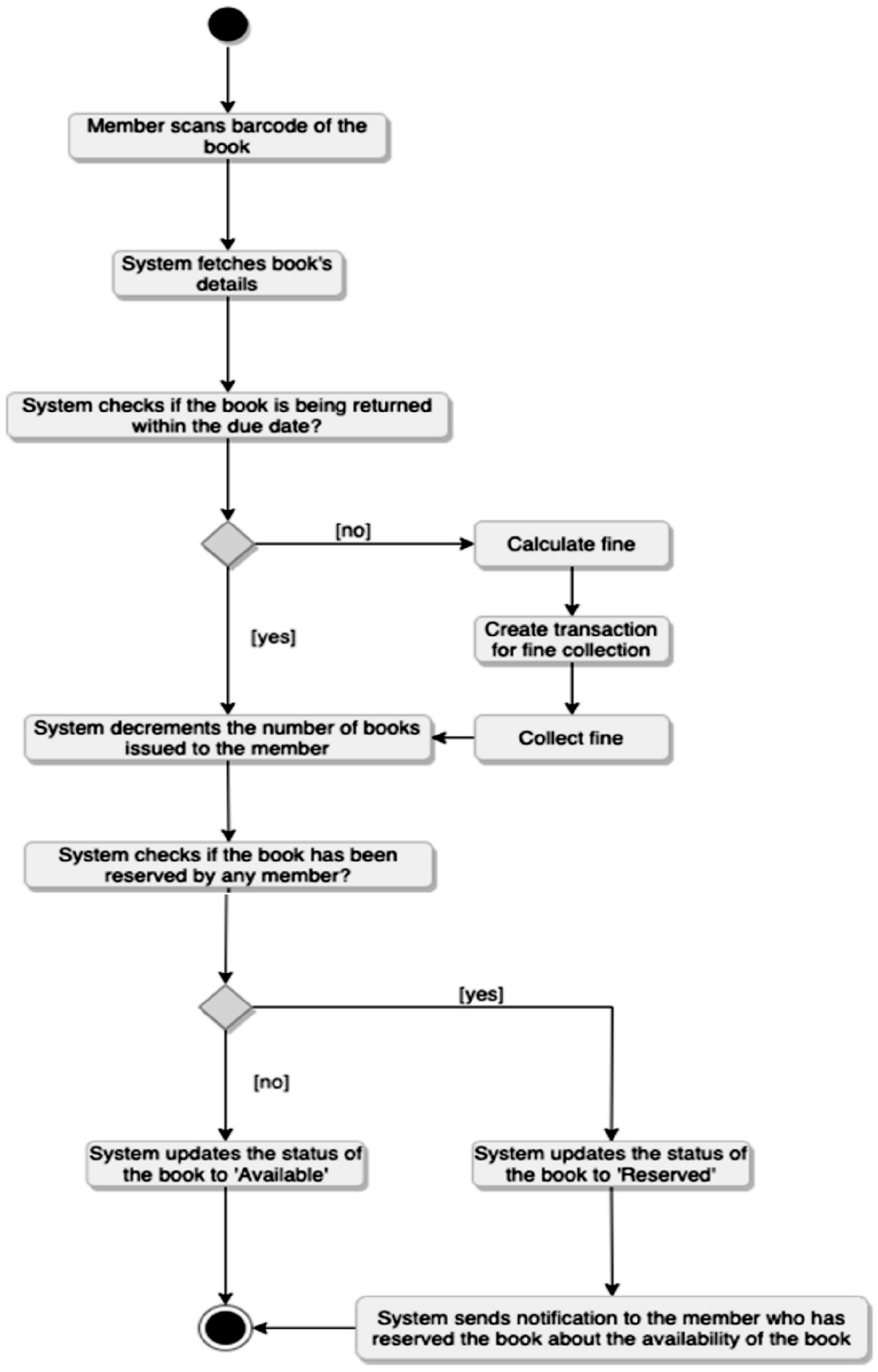
Ans :

Activity Diagram

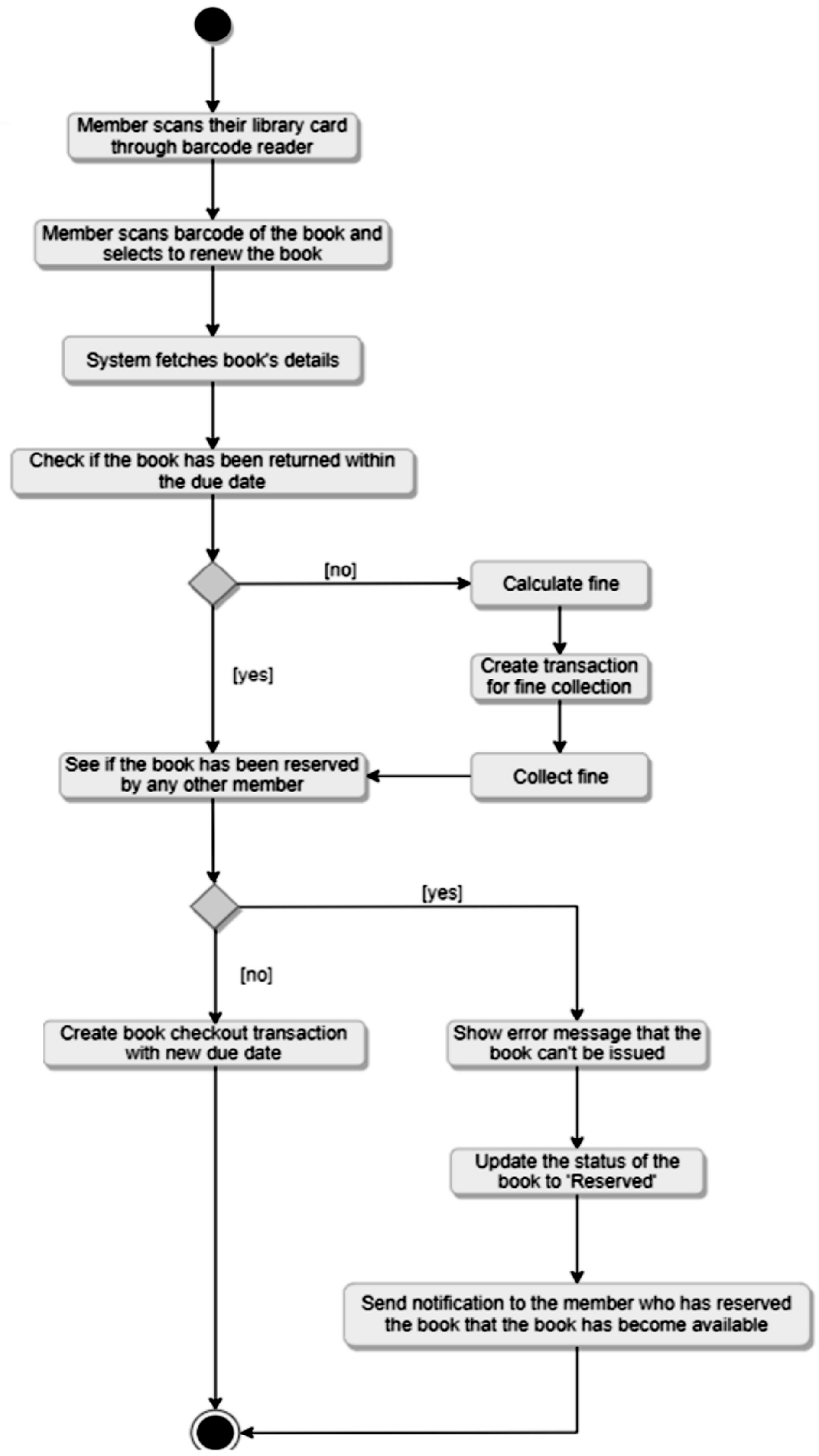
Check-out a book: Any library member or librarian can perform this activity. Here are the set of steps to check-out a book:


Member scans their library card through barcode reader
Member scans the barcode of the book
System checks if the book can be issued and that the book is not 'reference only'?
[no]
[yes]
System checks the number of books issued to the member
[max quota exceeded]
[else]
System checks if the book has been reserved by any other member?
[yes]
[no]
System creates book checkout transaction
System updates the status of the book to 'Loaned'
Show error message
System increments number of books issued to the member
System marks any reservation 'Completed' that the member had made against the book

Return a book: Any library member or librarian can perform this activity. The system will collect fines from members if they return books after the due date. Here are the steps for returning a book:


Member scans barcode of the book
System fetches book's details
System checks if the book is being returned within the due date?
[no]
Calculate fine
[yes]
Create transaction for fine collection
System decrements the number of books issued to the member
Collect fine
System checks if the book has been reserved by any member?
[yes]
[no]
System updates the status of the book to 'Available'
System updates the status of the book to 'Reserved'
System sends notification to the member who has reserved the book about the availability of the book

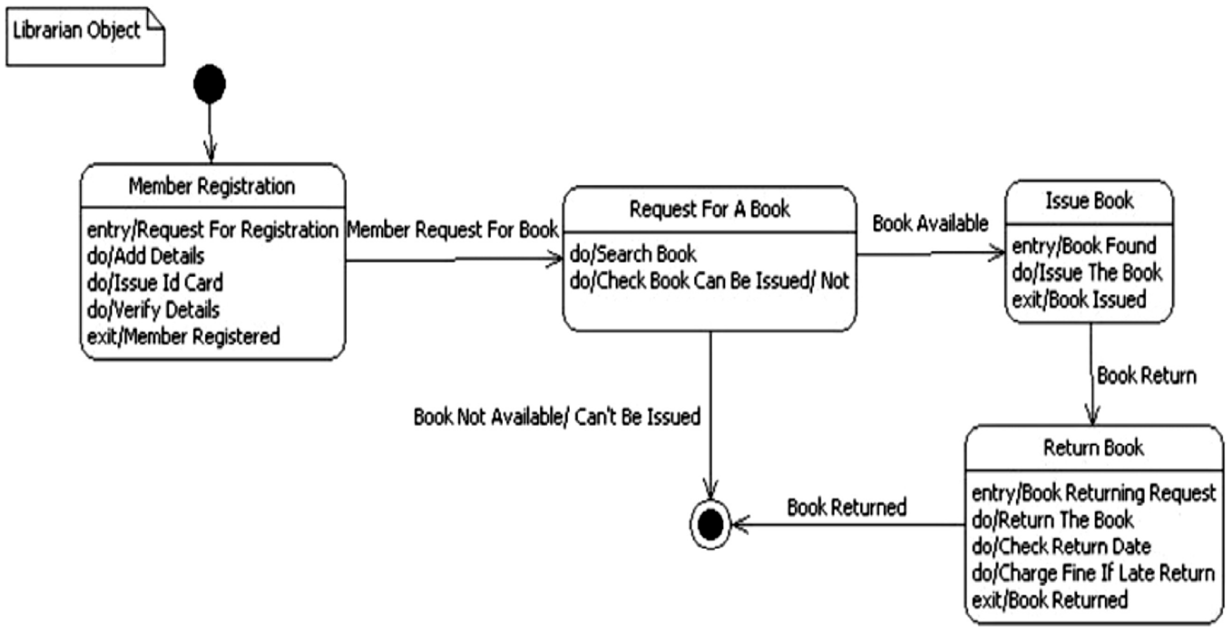
Renew a book: While renewing (re-issuing) a book, the system will check for fines and see if any other member has not reserved the same book, in that case the book item cannot be renewed. Here are the different steps for renewing a book:


Member scans their library card through barcode reader
Member scans barcode of the book and selects to renew the book
System fetches book's details
Check if the book has been returned within the due date
[no]
Calculate fine
[yes]
Create transaction for fine collection
See if the book has been reserved by any other member
Collect fine
[yes]
[no]
Create book checkout transaction with new due date
Show error message that the book can't be issued
Update the status of the book to 'Reserved'
Send notification to the member who has reserved the book that the book has become available

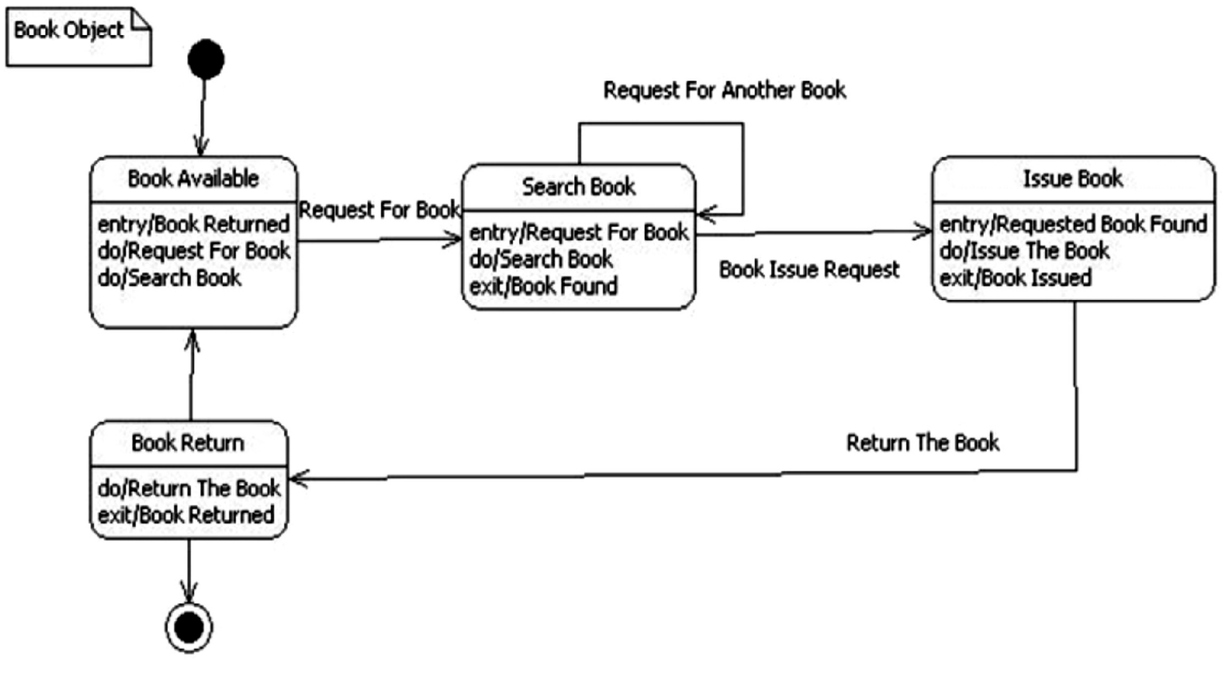
State-Chart Diagram

State diagram for Library Management System is shown below. Here we have described different states of following Objects: Librarian, Books, User

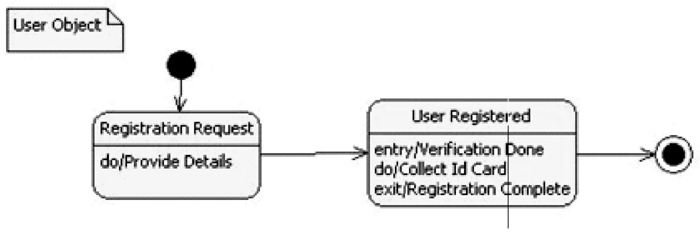
State Diagram For Librarian Object:



State Diagram For Books Object:



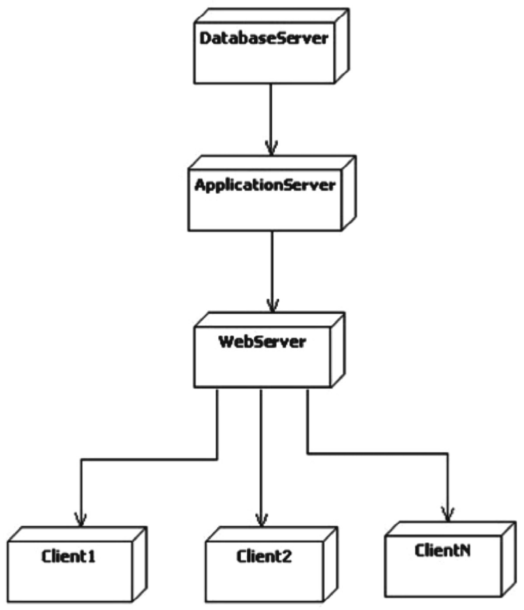
State Diagram For User Object:



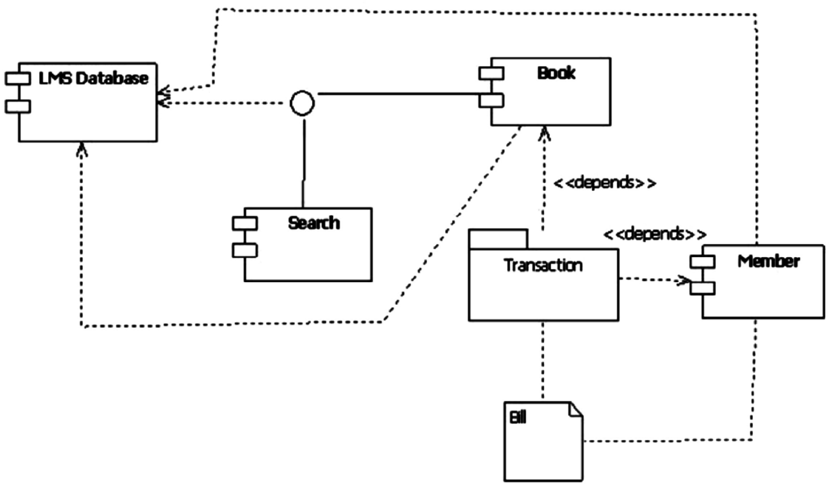
9. To draw the implementation view diagram: Component and Deployment diagram

Ans :

Deployment Diagram for Library management system



Component diagram for Library Management System



10. To perform various testing using the testing tool unit testing, integration testing

Ans :

Quality assurance is the review of the software product that checks for the correctness, reliability, completeness and maintainability. The different sections under quality assurance are unit testing, integrated testing, validation testing, output testing and user acceptance testing. Test cases gives an idea like on perform of some tasks what will be the predicted output or result. It will help in predicting the result on perform of certain tasks. The test cases below gives an idea of what result must be obtained on performing a particular task.

➢ **Login form**: The test cases involved are whether valid password and name are entered or invalid name and password entered.

➢ **Book entry form**: The test cases included are-on the click of add button, delete button, update button, search button, clear button, exit button and next button.

➢ **User account form**: The test cases included are-on the click of add button, delete button, update button, search button, clear button, exit button and next button.

➢ **Book return form**: The test cases included are-on the click of add button, delete button, update button, search button, clear button, exit button and next button.

Test Cases For Library Management System

LOGIN FORM:

SL.No	Test Case	Excepted Result	Test Result
1	Enter valid name and password & click on login button	Software should display main window	Successful
2	Enter invalid	Software should not display main window	successful

BOOK ENTRY FORM:

S.No.	Test Case	Excepted Result	Test Result
1	On the click of ADD button	At first user have to fill all fields with proper data , if any Error like entering text data instead of number or entering number instead of text..is found then it gives proper message otherwise Adds Record To the Database	successful
2.	On the Click of DELETE Button	This deletes the details of book by using Accession no.	Successful
3.	On the Click of UPDATE Button	Modified records are Updated in database by clicking UPDATE button.	Successful
4.	On the Click of SEARCH Button	Displays the Details of book for entered Accession no. Otherwise gives proper Error message.	Successful
5.	On the Click of CLEAR Button	Clears all fields	Successful
6.	On the Click of EXIT button	Exit the current book details form	successful
7.	On the Click of NEXT button	Display the next form	successful

User Account Form:

S.No	Test Case	Excepted Result	Test Result
1	On the click of ADD button	At first user have to fill all fields with proper data , if any Error like entering text data instead of number or entering number instead of text..is found then it gives proper message otherwise Adds Record To the Database	successful
2.	On the Click of DELETE Button	This deletes the details of student by using Register no.	Successful
3.	On the Click of UPDATE Button	Modified records are Updated in database by clicking UPDATE button.	Successful
4.	On the Click of SEARCH Button	Displays the Details of book for entered Register no. Otherwise gives proper Error message.	Successful
5.	On the Click of CLEAR Button	Clears all fields	Successful
6.	On the Click of EXIT button	Exit the current book details form	Successful
7.	On the Click of NEXT button	Display the next form	Successful

BOOK ISSUE FORM:

S.No	Test Case	Excepted Result	Test Result
1	On the click of ADD button	At first user have to fill all fields with proper data ,if the accession number book is already issued then it will giving proper msg.	Successful
2.	On the Click of DELETE Button	This deletes the details of book by using Register no.	Successful
3.	On the Click of UPDATE Button	Modified records are Updated in database by clicking UPDATE button.	Successful
4.	On the Click of SEARCH Button	Displays the Details of issued book..Otherwise gives proper Error message.	Successful
5.	On the Click of CLEAR Button	Clears all fields	Successful
6.	On the Click of EXIT button	Exit the current book details form	Successful
7.	On the Click of NEXT button	Display the next form	Successful

BOOK RETURN FORM:

S.No.	Test Case	Excepted Result	Test Result
1	On the click of ADD button	At first user have to fill all fields with proper data , if any Error like entering text data instead of number or entering number instead of text..is found then it gives proper message otherwise Adds Record To the Database	Successful
2.	On the Click of DELETE Button	Which deletes the details of book by using Register no.	Successful
3.	On the Click of UPDATE Button	Modified records are Updated in database by clicking UPDATE button.	Successful
4.	On the Click of SEARCH Button	Displays the Details of returned book … Otherwise gives proper Error message.	Successful
5.	On the Click of CLEAR Button	Clears all fields	Successful
6.	On the Click of EXIT button	Exit the current book details form	Successful
7.	On the Click of NEXT button		

3. Railway Reservation

1. Phases in software development project, overview, need, coverage of topics

Ans :

Our website has various kinds of information that helps regarding booking of tickets via railways.

Users will be able to search the train availability,the exact fare, the arrival and departure time of the train and they can also book the ticket by using the debit, credit or master card and after booking the ticket if the user want to cancel it then they can easily do it also.

Railway passengers frequently need to know about their ticket reservation status, ticket availability on a particular train or for a place, train arrival or departure details, special trains etc.. Customer information centers at the railway stations are unable to serve such queries at peak periods.

The number of the reservation counters available to the passengers and customers are very less.On most of the reservation systems there are long queues, so it takes a long time for any individual to book the ticket.As now there are no call centers facilities available to solve the queries of the passengers.

The online railway ticket reservation system aims to develop a web application which aims at providing trains details, trains availability, as well as the facility to book ticket in online for customers.

So, we thought of developing a web based application which would provide the users all these facilities from his terminal only as well as help them in booking their tickets. The Application was to be divided into two parts namely the user part , and the administrator part. And each of these has their corresponding features.

We decided to give the name of the website “ONLINE RAILWAY TICKET RESEVATION”.

The online railway ticket reservation system is developed using ASP.NET with C# as the backend in the .NET Framework.

Objectives

The objective of the online railway ticket reservation system Project is to design software to fully automate the process of issuing a railway ticket. That is:-

1. To create a database of thetrains
2. To search the trains it’s arrival and departure time, distance between source and destination.
3. To check the availability of theticket.
4. To calculate fare.
5. To book the ticket.
6. To cancel the ticket if necessary.

2. To assign the requirement engineering tasks

Ans :

Analysis

The track is one of the factors that strike to our mind when we speak about railways. The train may reach the intended destination through different tracks or through the single track which connects the source to the destination. Nowadays the passenger can book the railway tickets online easily than waiting in long queues to obtain the ticket. This system should include the name of the train, source, destination, time, date of arrival or departure etc. Nowadays the passenger can book the railway tickets online easily than waiting in long queues to obtain the ticket. The passenger ticket booking system should include the name of the train, source, destination, time, date of arrival or departure etc.

Online railway ticket reservation is a online ticket booking website, which is capable of booking ticket and search the train availability. This website is mainly created to fulfil the following requirements, it comprises of the following properties:-

- A central database that will store allinformation.
- An online website that will provide real- time information about the availability of tickets their prices.
- Every registered user is able to view his booking id that has been made in his/her name.
- Every registered user can change his password any time he wants tochange.
- Every guest user can search train availability,price of the ticket, arrival and departure time,distance between source and destinationetc.
- Every registered user has the facilities to print his ticket any time he wishes. Administrationlogin
  - In admin mode the administrator can make changes in traindetails.
  - He can also view all booking that has been made by differentusers.

The booking window contains all the facilities at one place, the user cansimply login to his account and can book histicket.

The features of this system are as follows:

- **Seats availability**: The users through the use of this system can check whether the seats are available in the particular train to travel.
- **Train tickets**: The users can access their train tickets through the online mode through the use of this application.
- **Stations**: This application can also contain the details of the various stations from the source to the destination of the passenger.
- **Trains**: The trains will be having the name and the id number.
- **Train schedules**: Even the schedules of different trains can be mentioned through this application.
- **Time in and time out**: For each train at every station there is a time in or time out time. It indicates that passengers should get in and get out of the train only within that particular duration of time.

3. To perform the system analysis: Requirement analysis, SRS

Ans :

System Concept Development Phase

The System Concept Development Phase begins after a business need or opportunity is validated by the Agency/Organization Program Leadership and the Agency/Organization CIO.

The purpose of the System Concept Development Phase is to:

- ➢ Determine the feasibilityand appropriateness of the alternatives.
- ➢ Identify system interfaces.
- ➢ Identify basic functional and data requirements to satisfy the business need.
- ➢ Establish system boundaries, identifygoals, objectives, critical success factors, and performance measures.
- ➢ Evaluate costs and benefits of alternative approaches to satisfy the basic functional requirements
- ➢ Assess project risks
- ➢ Identify and initiate risk mitigation actions, andDevelop high-level technical architecture,process models, data models, and a concept of operations.

This phase explores potential technical solutions within the context of the business need. It may include several trade-off decisions such as the decision to use COTS software products as opposed to developing custom software or reusing software components, or the decision to use an incremental delivery versus a complete, one- time deployment. Construction of executable prototypes is encouraged to evaluate technology to support the business process.

The System Boundary Document serves as an important reference document to support the Information Technology Project Request (ITPR) process. The ITPR must be approved by the State CIO before the project can move forward.

DevelopmentEnvironments

Hardware

Intel Core™ i3 7020U 2.30GHz 64 bit with 8GB RAM, 500 GB hard disk space and other Standard accessories.

Environment and Applications

- ➢ Microsoft Windows10.
- ➢ Microsoft Visual Studio2010.
- ➢ Microsoft SQL Server2005.
- ➢ Microsoft InternetExplorer.

Operating environment :

Hardware configuration

The minimum configuration for hardware is given below:

- ➢ Intel® Pentium® or higherprocessor.
- ➢ 4 GB RAM or higher

Software configuration

- Microsoft® Windows® 10 or laterversions
- A standard web browser.
- .Net framework.

Functions of_ Project

There are seven functionalities provided by the Railway Reservation System.

- **Create Reservations:** A passenger should be able to reserve seats in the train. A reservation form is filled by the passenger and given to the clerk, who then checks for the availability of seats for the specified date of journey. If seats are available them the entries are mode in the system regarding the train name, train number,date of journey, boarding station,destination,person name,sex and total fare. Passenger is asked to pay the required fare and the tickets are printed. It the seats are not available then the passenger is informed.
- **Cancel Reservation:** A passenger wishing to cancel a reservation is required to fill a form. The passenger then submits the form and the ticket to the clerk. The clerk then deletes the entries in the system and changes the reservation status of that train. The clerk crosses the ticket by hand to mark as cancelled.
- **Update Train Info:** Only the administrator or manager enters any changes related to the train information like change in the train name, train number,train route etc. in the system.
- **Generate Report:**Provision for generation of different reports should be given in the system. The system should be able to generate reservation chart, monthly train report etc.
- **Verify login:**For security reasons all the users of the system are given a user id and a password. Only if the id and password are correct is the user allowed entry to the system and select from the options available in the system.
- **View Reservation Status:**All the users should be able to see the reservation status of the train online. The users needs to enter the train number and the pin number printed on his ticket so that the system can display his current reservation status like confirmed,RAC or Wait - Listed.
- **View Train Schedule:** Provision should be given to see information related to the train schedules for the entire train network. The user should be able to see the train name, train number,boarding and destination stations, duration of journey etc.

4. To perform the function-oriented diagram: DFD and Structured chart

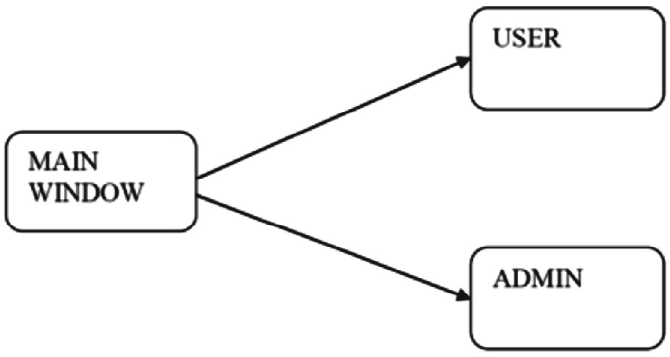
Ans :

Data Flow Diagram

This is the Zero Level DFD of Railway Reservation System, where we have elaborated the high level process of Railway Reservation. It's a basic overview of the whole Railway Reservation System or process being analyzed or modeled. It's designed to be an at-a-glance view of Train Route,Customer and Payment showing the system as a single high-level process, with its relationship to external entities of Trains,Booking and Ticket. It should be easily understood by a wide audience, including Trains,Ticket and Train Route In zero level DFD of Railway Reservation System, and we have described the high level flow of the Railway Reservation system.

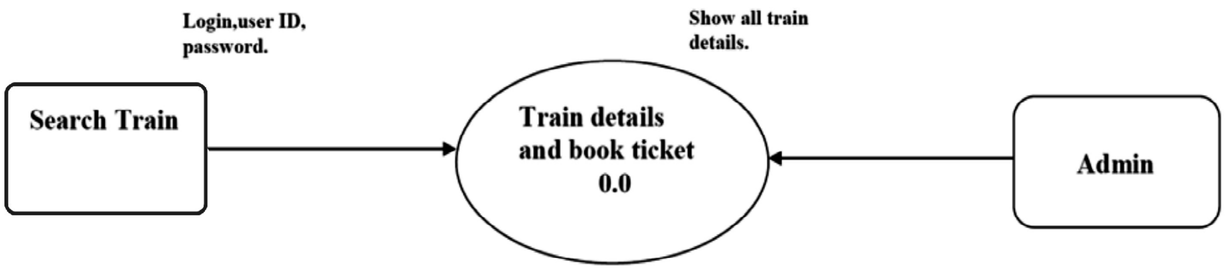
High Level Entities and process flow of Railway Reservation System:

- Managing all the Trains
- Managing all the Booking
- Managing all the Ticket
- Managing all the Train Shedule
- Managing all the Train Route
- Managing all the Customer
- Managing all the Payment

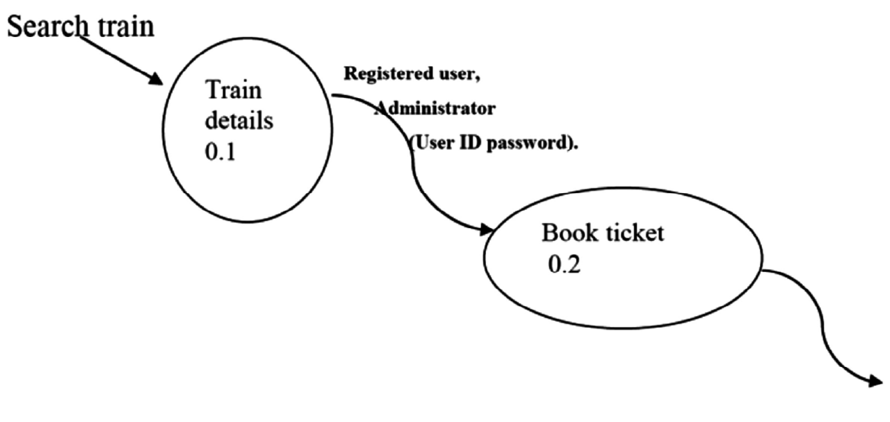


Level 0:

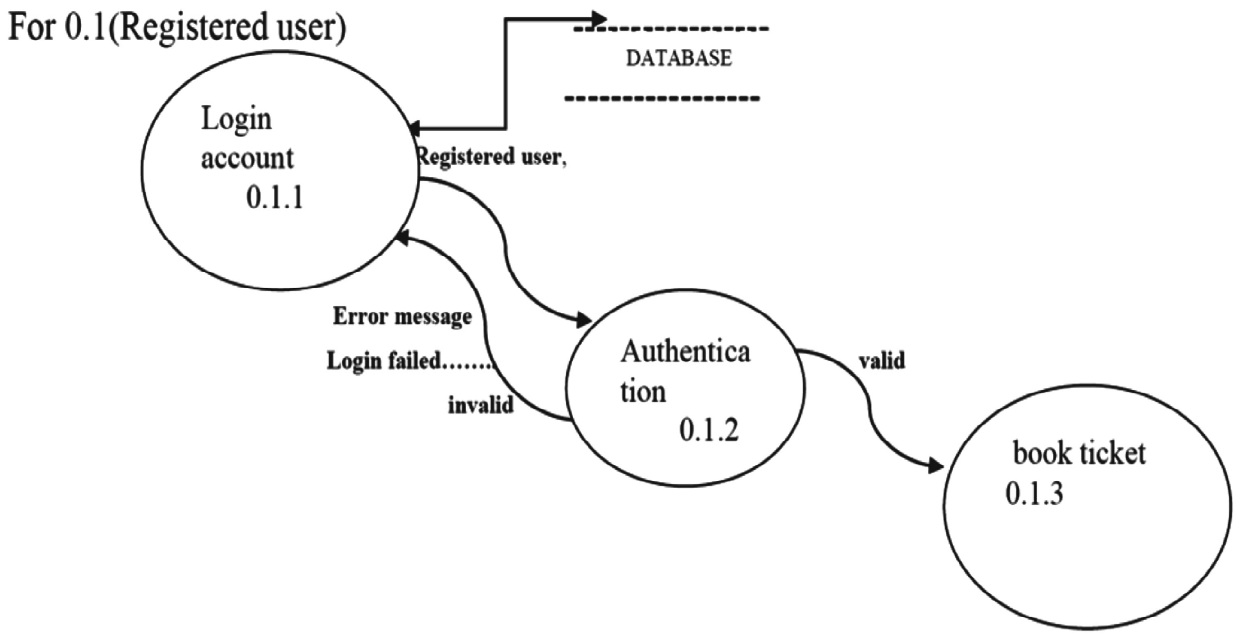
CONTEXT DIAGRAM



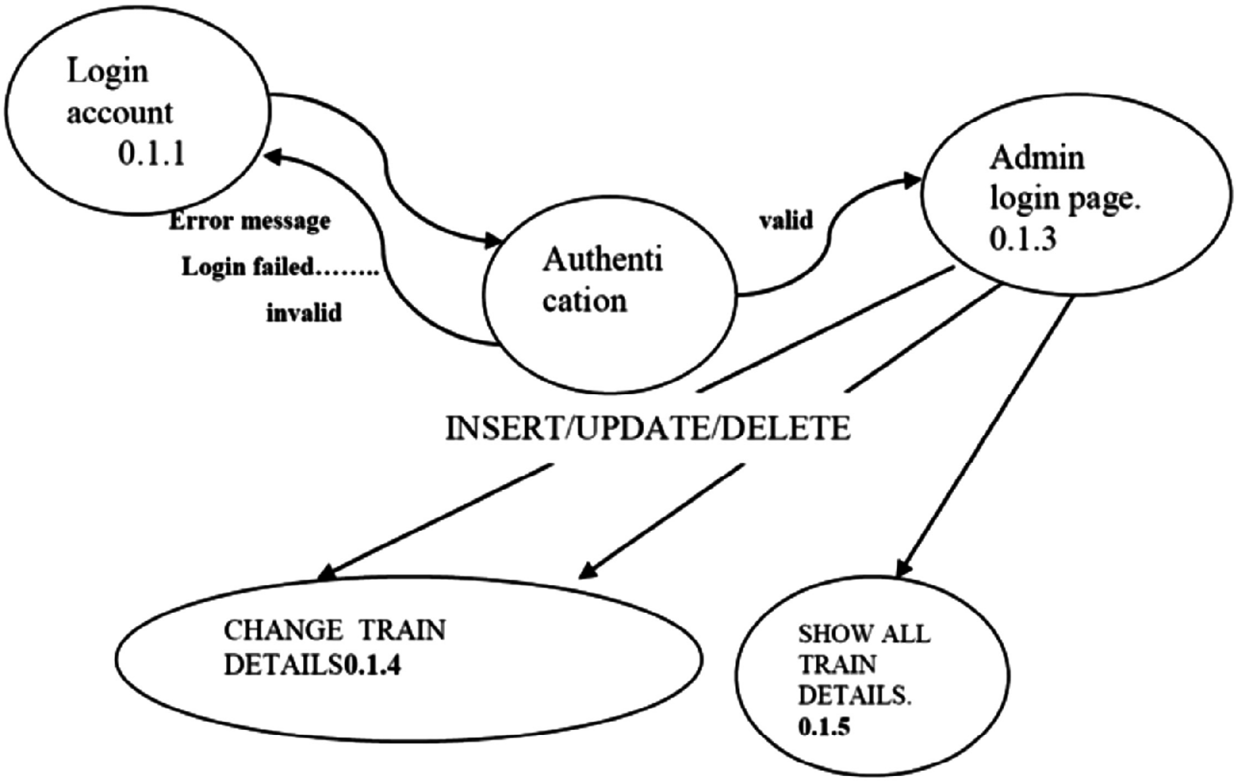
Level 1 DFD



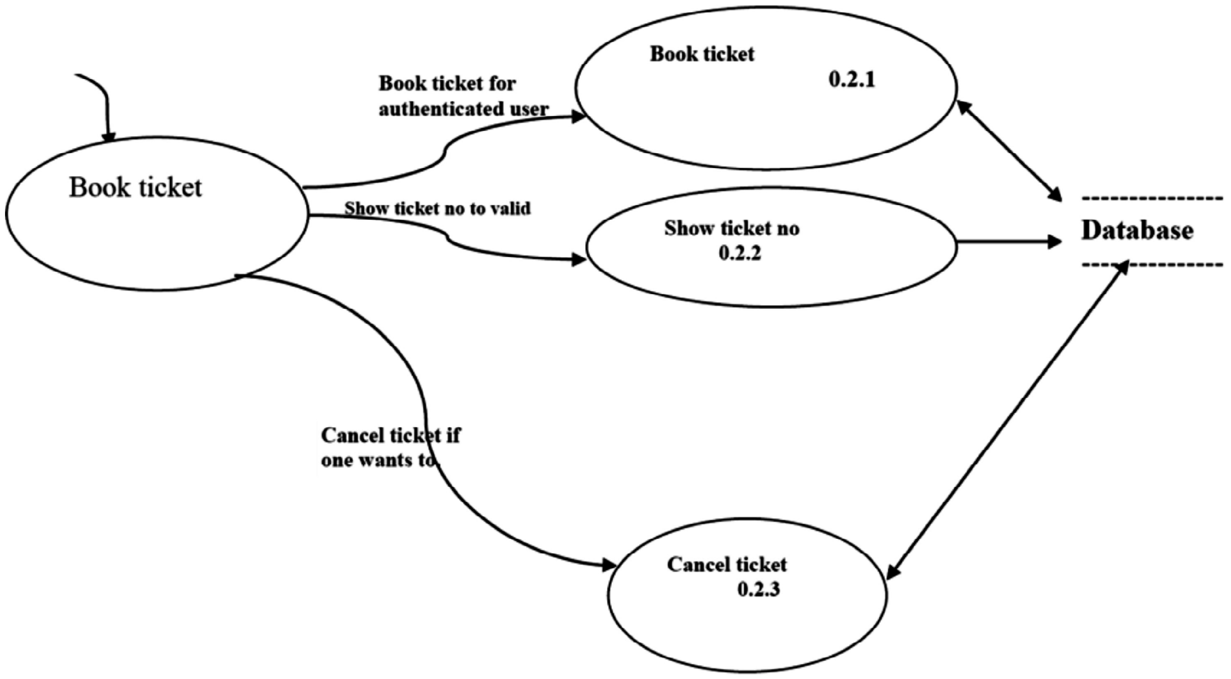
Level 2 DFD



Administrator



Level 3 DFD



5. To perform the user's view analysis: Use case diagram

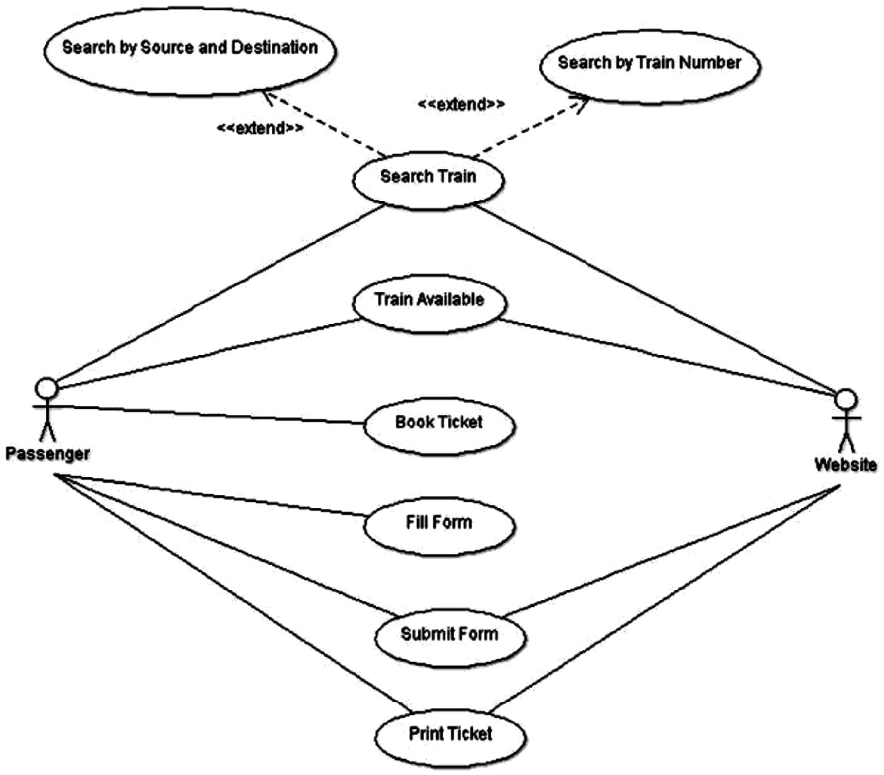
Ans :

This Use Case Diagram is a graphic depiction of the interactions among the elements of Railway Reservation System. It represents the methodology used in system analysis to identify, clarify, and organize system requirements of Railway Reservation System. The main actors of Railway Reservation System in this Use Case Diagram are: Super Admin, System User, Ticket Agent, Customers, who perform the different type of use cases such as ManageTrain, Manage Ticket, Manage Booking, Manage Customer, Manage Payment, Manage Train Route, Manage Train Schedule, Manage Users and Full Railway Reservation System Operations. Major elements of the UML use case diagram of Railway Reservation System are shown on the picture below.

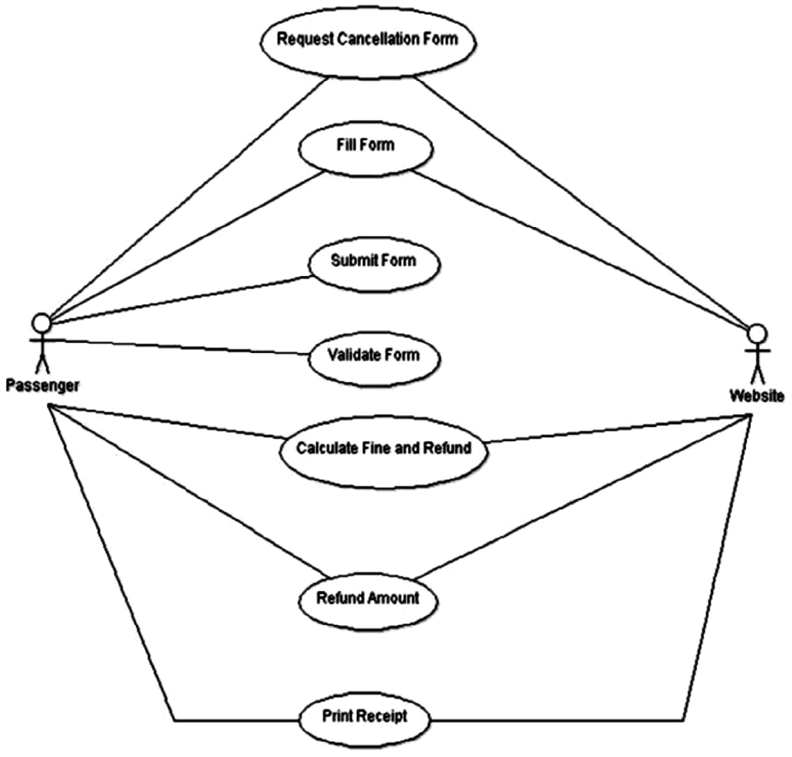
The relationships between and among the actors and the use cases of Railway Reservation System:

- **Super Admin Entity :** Use cases of Super Admin are ManageTrain, Manage Ticket, Manage Booking, Manage Customer, Manage Payment, Manage Train Route, Manage Train Schedule, Manage Users and Full Railway Reservation System Operations
- **System User Entity :** Use cases of System User are ManageTrain, Manage Ticket, Manage Booking, Manage Customer, Manage Payment, Manage Train Route, Manage Train Schedule
- **Ticket Agent Entity :** Use cases of Ticket Agent are Book Tickets, Search Vacant Seats, Collect Payment
- **Customers Entity :** Use cases of Customers are Search Trains, Book Tickets, Make Payments

Ticket Booking



Ticket Cancellation



6. To draw the structural view diagram: Class diagram, object diagram

Ans :

Railway Reservation System Class Diagram describes the structure of a Railway Reservation System classes, their attributes, operations (or methods)and the relationships among objects.The main classes of the Railway Reservation System are Train, Ticket, Booking, Customer, Payment, Train Route.

Classes of Railway Reservation System Class Diagram:

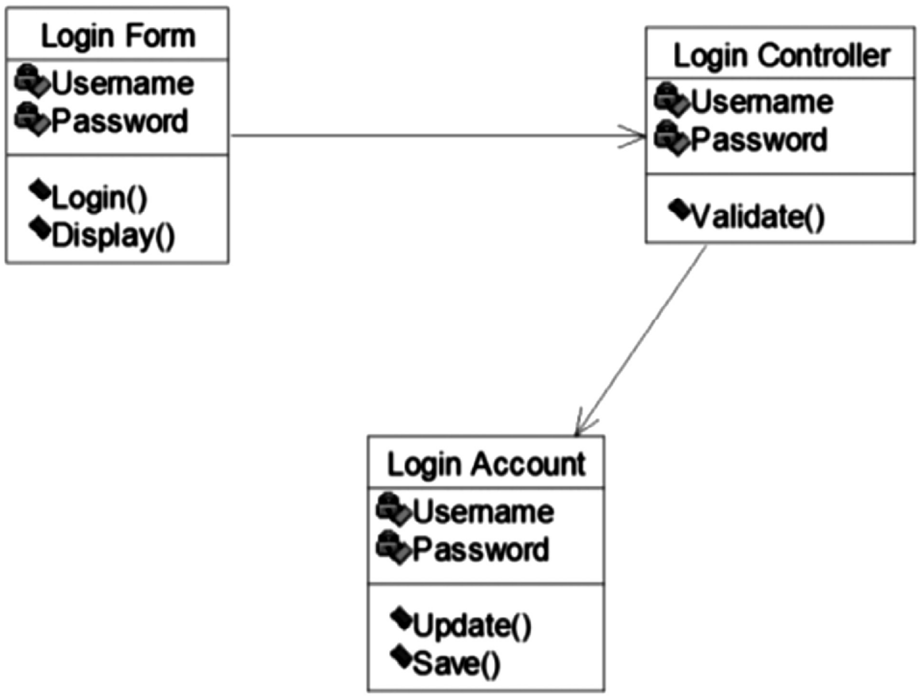
- **Train Class :** Manage all the operations of Train
- **Ticket Class :** Manage all the operations of Ticket
- **Booking Class :** Manage all the operations of Booking
- **Customer Class :** Manage all the operations of Customer
- **Payment Class :** Manage all the operations of Payment
- **Train Route Class :** Manage all the operations of Train Route

Classes and their attributes of Railway Reservation System Class Diagram:

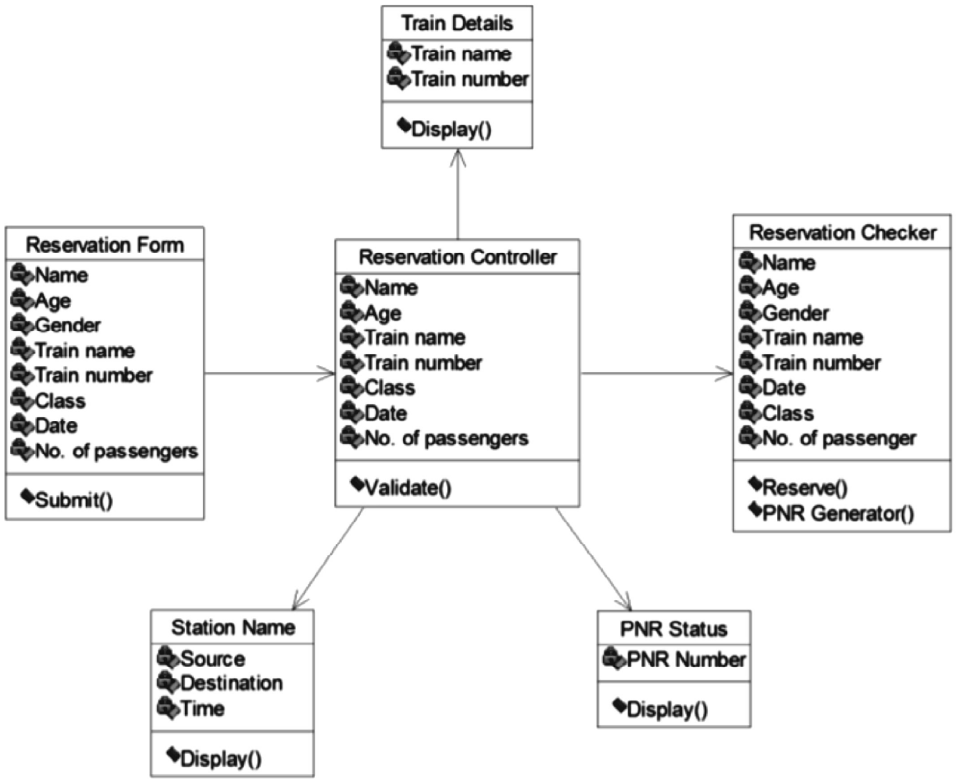
- **Train Attributes:** train_id, train_name, train_number, train_seat_number, train_ticket, train_type, train_description
- **Ticket Attributes:** ticket_id, ticket_customer_id, ticket_type, ticket_date, ticket_description
- **Booking Attributes:** booking_id, booking_title, booking_type, booking_ticket, booking_date, booking_description
- **Customer Attributes:** customer_id, customer_name, customer_mobile, customer_email, customer_username, customer_password, customer_address
- **Payment Attributes:** payment_id, payment_customer_id, payment_date, payment_amount,payment_description
- **Train Route Attributes:**train_route_id, train_route_name, train_route_type, train_route_description

Classes and their methods of Railway Reservation System Class Diagram:

- **Train Methods:** addTrain(), editTrain(), deleteTrain(), updateTrain(), saveTrainQ, searchTrain()
- **Ticket Methods:** addTicket(), editTicketQ, deleteTicket(), updateTicketQ, saveTicket(), searchTicket()
- **Booking Methods:** addBooking(), editBooking(), deleteBooking(), updateBookingQ, saveBooking(), searchBooking()
- **Customer Methods:** addCustomer(), editCustomerQ, deleteCustomerQ, updateCustomerQ, saveCustomer(), searchCustomer()
- **Payment Methods:** addPaymentQ, editPaymentQ, deletePayment(), updatePayment(), savePayment(), searchPayment()
- **Train Route Methods:** addTrain Route(), editTrain Route(), deleteTrain Route(), updateTrain Route(), saveTrain Route(), searchTrain Route()



Class Diagram for Login



Class Diagram for Reservation

7. To draw the behavioural view diagram: Sequence diagram, Collaboration diagram

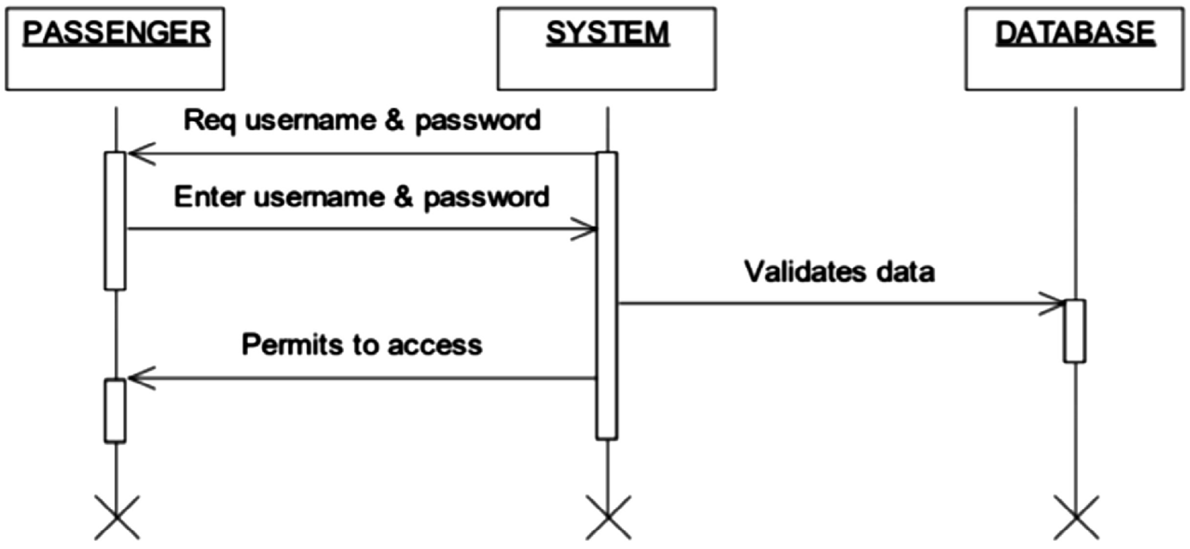
Ans :

This is the **UML sequence diagram of Railway Reservation System** which shows the interaction between the objects of Customer, Ticket, Train Route, Train Schedule, Booking. The instance of class objects involved in this UML Sequence Diagram of Railway Reservation System are as follows:

- Customer Object
- Ticket Object
- Train Route Object
- Train Schedule Object
- Booking Object

Login Sequence Diagram OF Railway Reservation System:

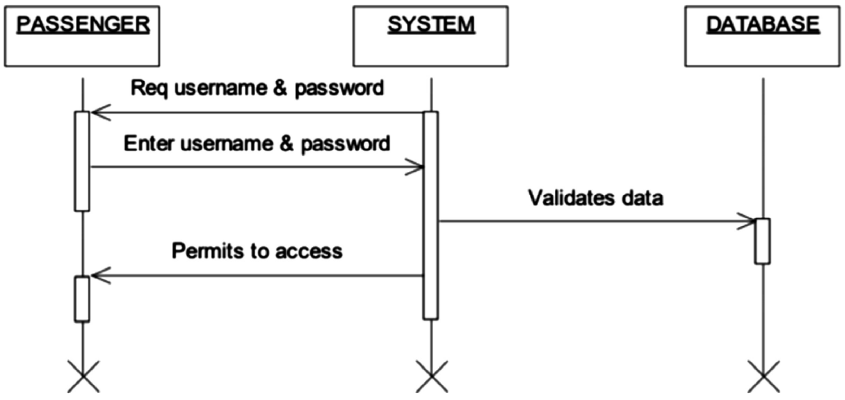
This is the **Login Sequence Diagram of Railway Reservation System,** where admin will be able to login in their account using their credentials. After login user can manage all the operations on Train Route, Customer, Ticket, Booking, Train Schedule. All the pages such as Ticket, Booking, Train Schedule are secure and user can access these page after login. The diagram below helps demonstrate how the login page works in a Railway Reservation System. The various objects in the Booking, Train Route, Customer, Ticket, and Train Schedule page—interact over the course of the sequence, and user will not be able to access this page without verifying their identity.



Sequence Diagram for Login

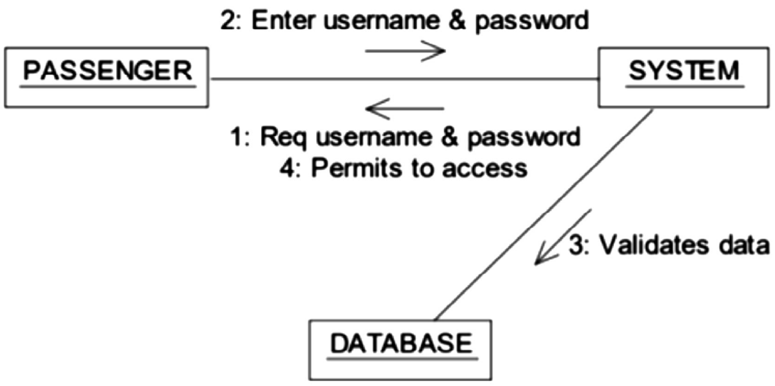
This is the **UML sequence diagram of Railway Reservation System** which shows the interaction between the objects of Customer, Ticket, Train Route, Train Schedule, Booking. The instance of class objects involved in this UML Sequence Diagram of Railway Reservation System are as follows:

- Customer Object
- System Object
- Database Object

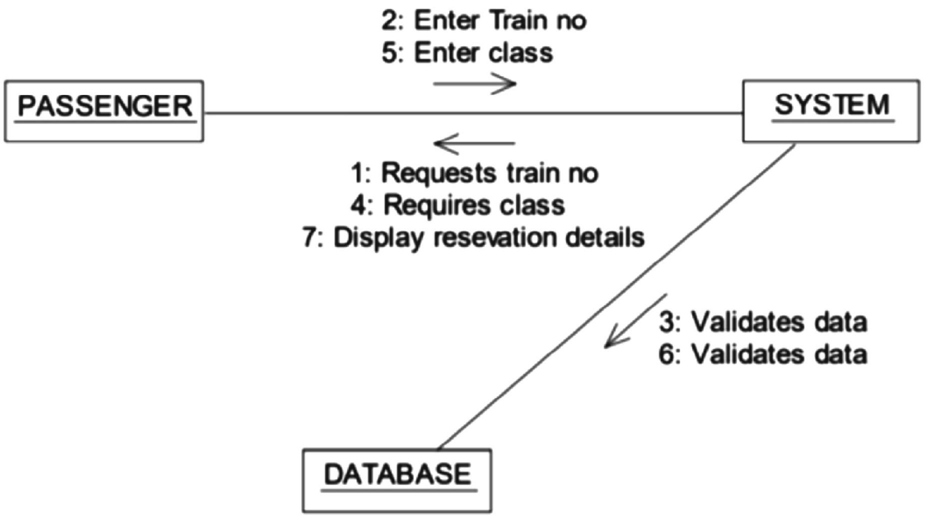


Sequence Diagram for Login

Collaboration Diagram



Collaboration Diagram for Login



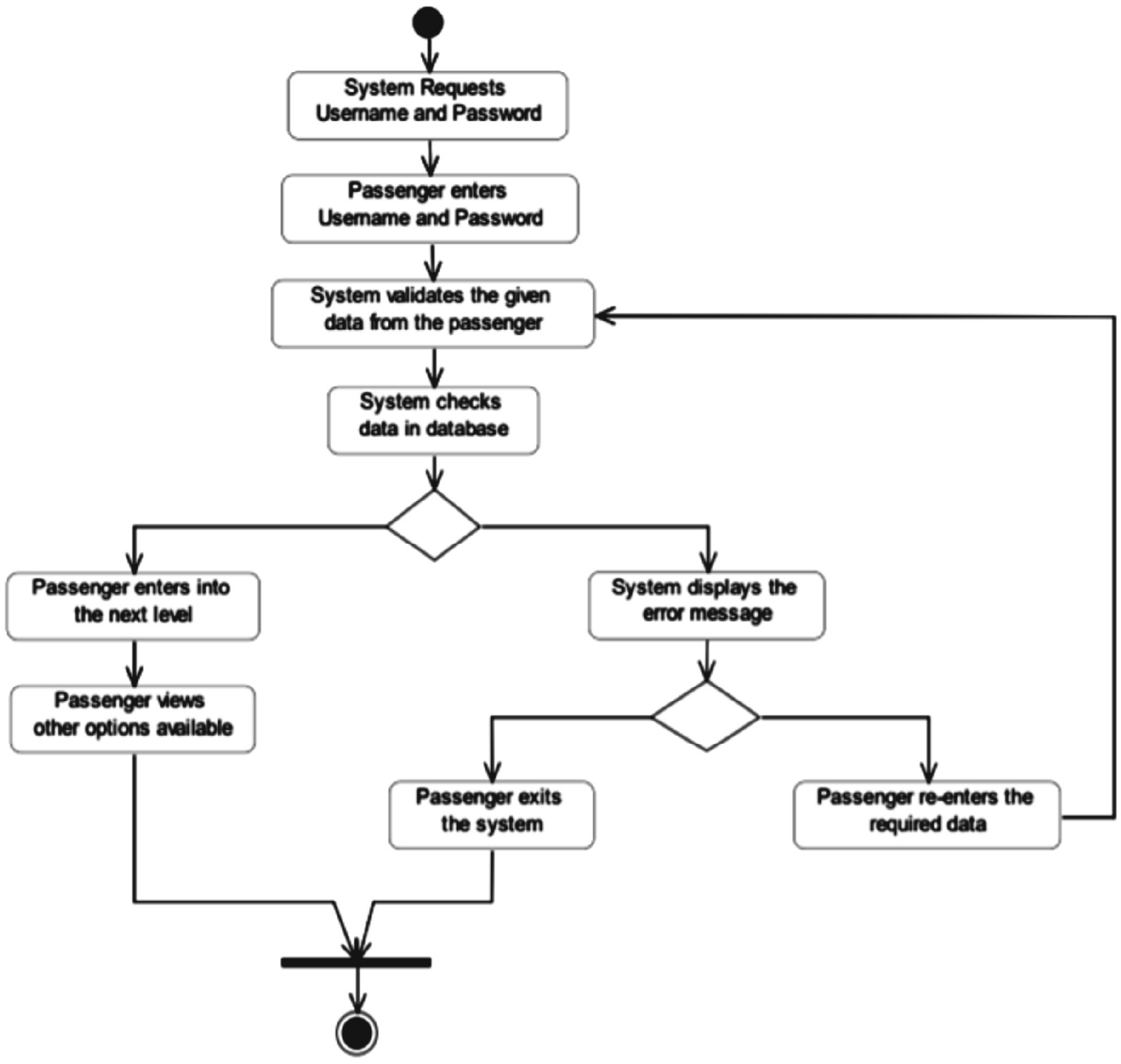
Collaboration Diagram for Reservation

8. To draw the behavioural view diagram: State-chart diagram, Activity diagram

Ans :

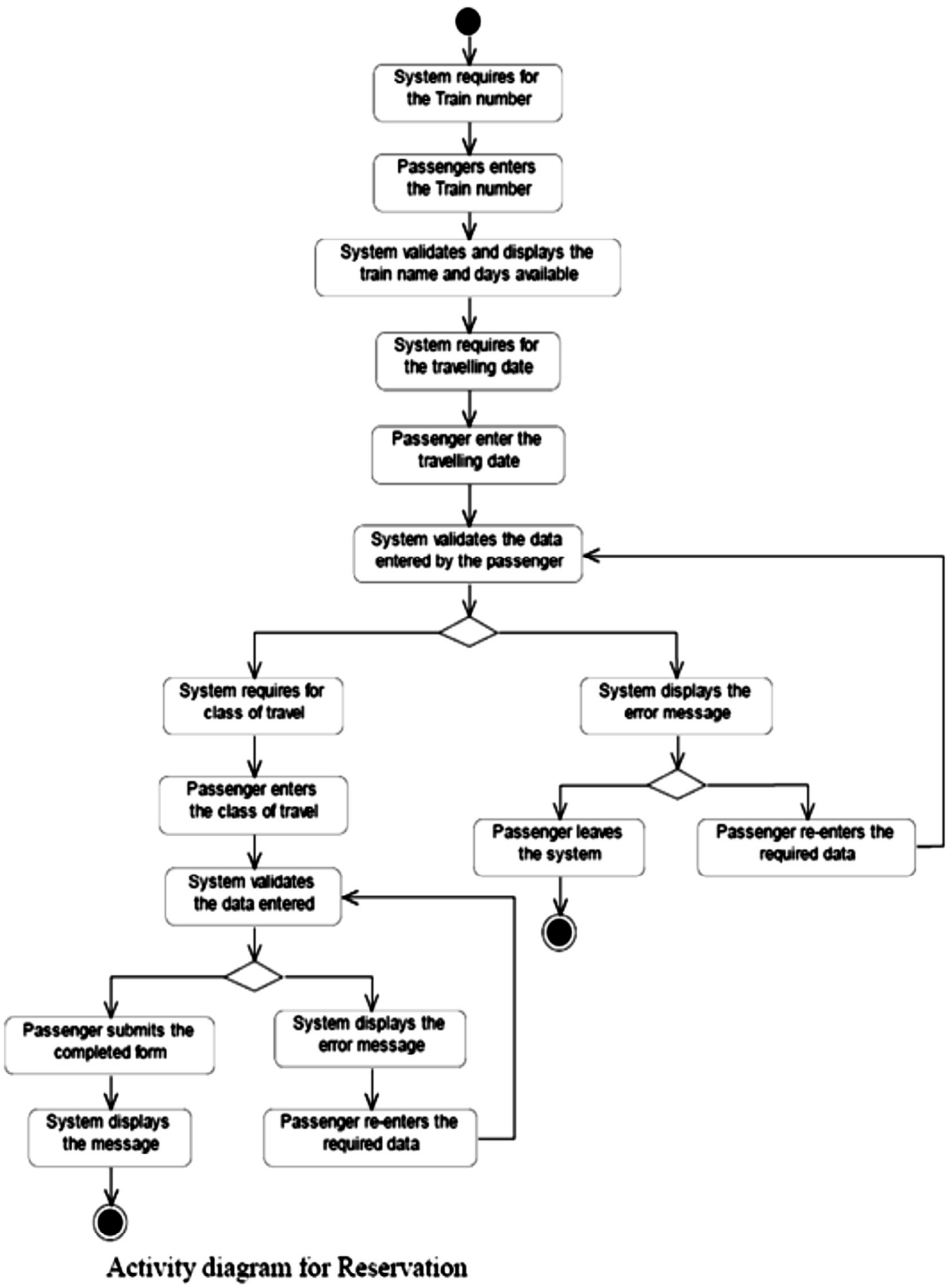
Login Activity Diagram

This is the **Login Activity Diagram of Railway Reservation System,** which shows the flows of Login Activity, where admin will be able to login using their username and password. After login user can manage all the operations on Payment, Ticket, Customer, Train Schedule, Booking. All the pages such as Customer, Train Schedule, Booking are secure and user can access these page after login. The diagram below helps demonstrate how the login page works in a Railway Reservation System. The various objects in the Train Schedule, Payment, Ticket, Customer, and Booking page—interact over the course of the Activity, and user will not be able to access this page without verifying their identity.



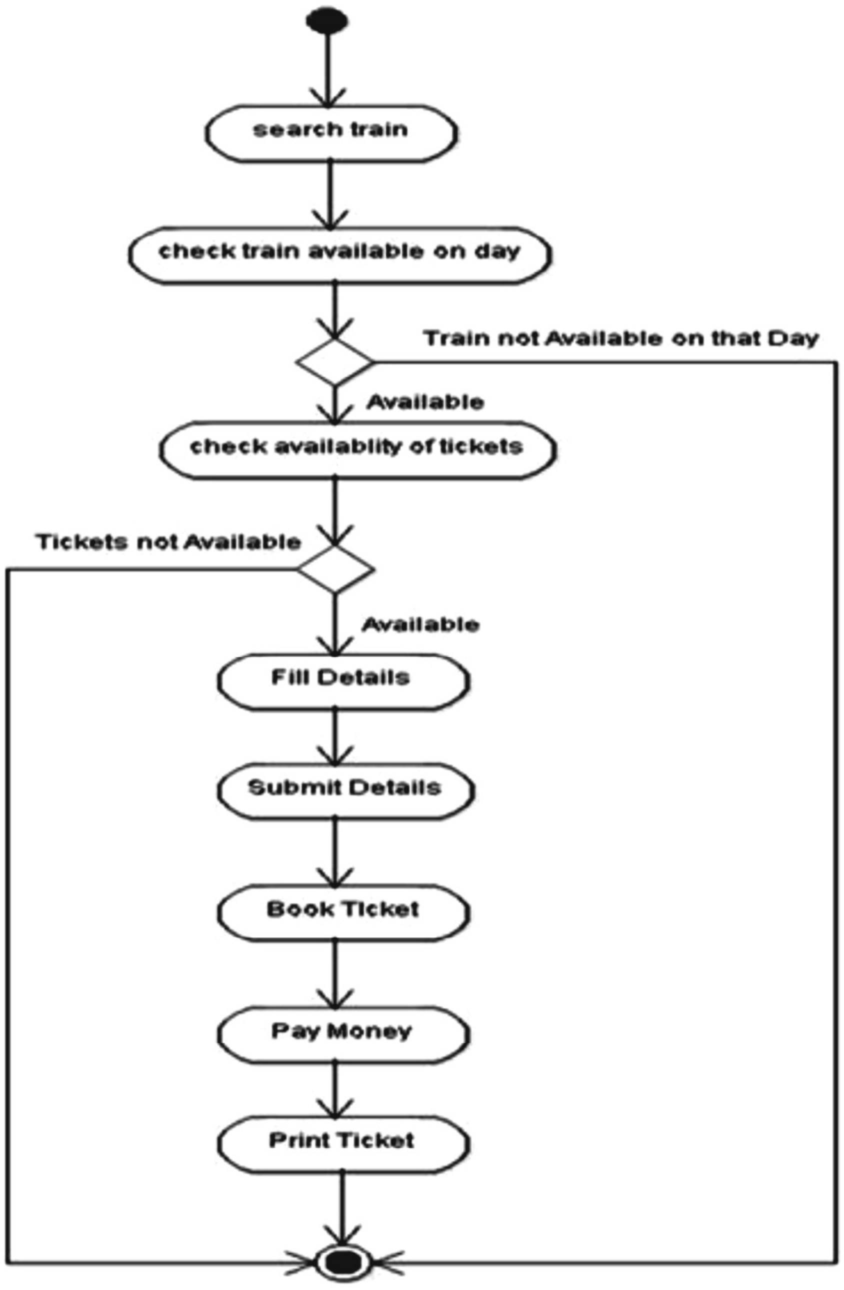
System Requests Username and Password
Passenger enters Username and Password
System validates the given data from the passenger
System checks data in database
Passenger enters into the next level
System displays the error message
Passenger views other options available
Passenger exits the system
Passenger re-enters the required data

Activity Diagram for Login

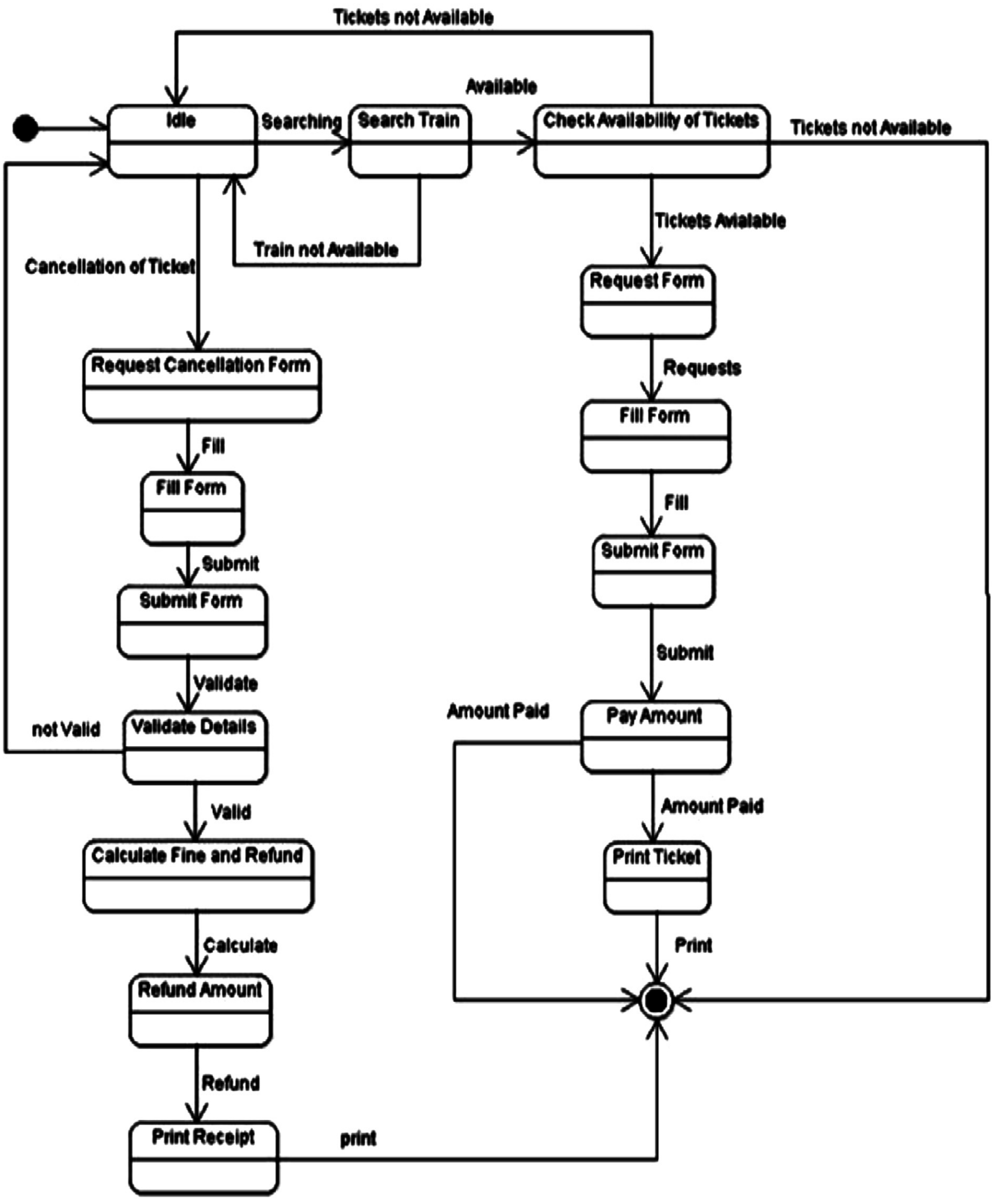

System requires for the Train number
Passengers enters the Train number
System validates and displays the train name and days available
System requires for the travelling date
Passenger enter the travelling date
System validates the data entered by the passenger
System requires for class of travel
System displays the error message
Passenger enters the class of travel
Passenger leaves the system
Passenger re-enters the required data
System validates the data entered
Passenger submits the completed form
System displays the error message
System displays the message
Passenger re-enters the required data

Activity diagram for Reservation

State Chart Diagram



Activity Diagram for Booking Ticket



State chart Diagram for Railway Reservation System

9. To draw the implementation view diagram: Component diagram & Deployment diagram

Ans :

Railway Reservation System Component Diagram

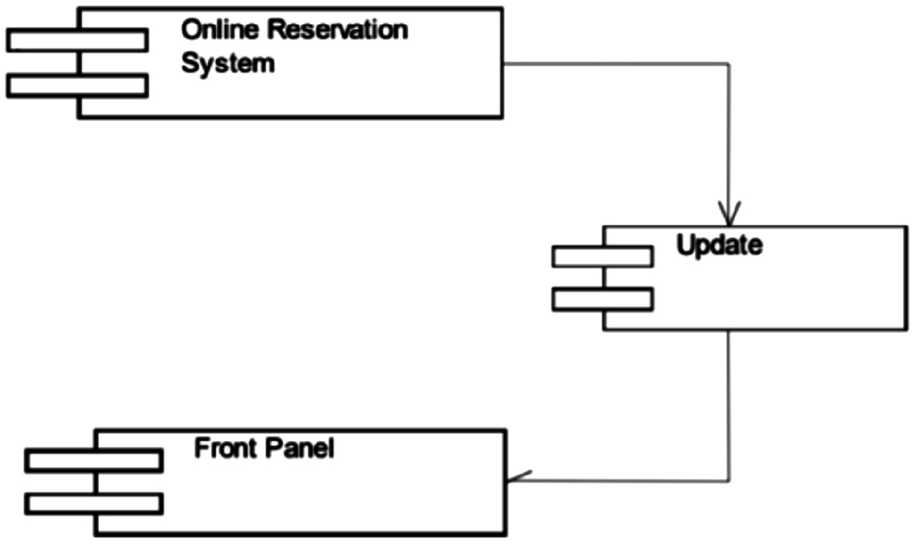
This is a **Component diagram of Railway Reservation System** which shows components, provided and required interfaces, ports, and relationships between the Train Schedule, Ticket, Booking, Customer and Payment. This type of diagrams is used in Component-Based Development (CBD) to describe systems with Service-Oriented Architecture (SOA). **Railway Reservation System UML component diagram,** describes the organization and wiring of the physical components in a system.

Components of UML Component Diagram of Railway Reservation System:

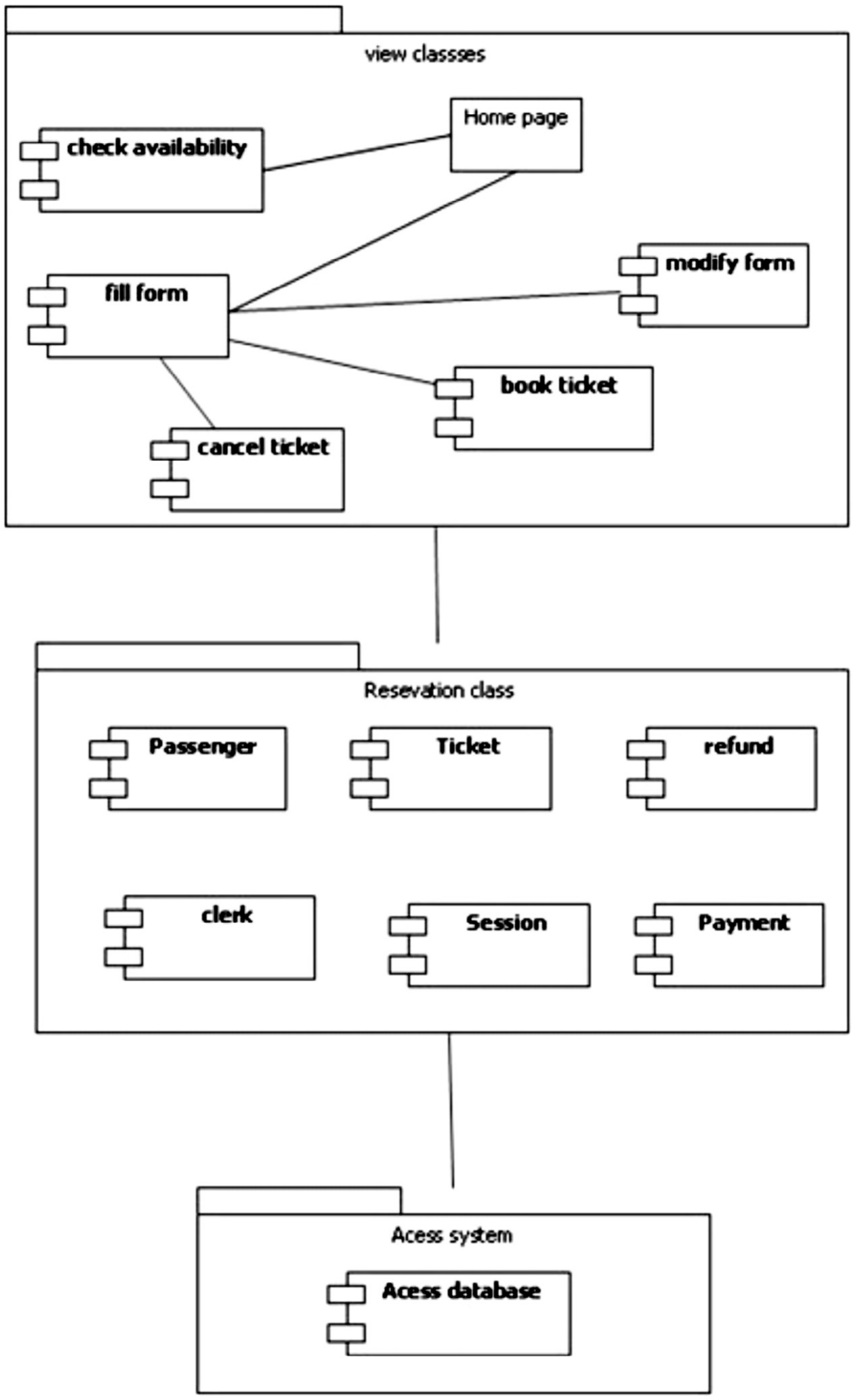
- Train Schedule Component
- Ticket Component
- Booking Component
- Customer Component
- Payment Component

Features of Railway Reservation System Component Diagram:

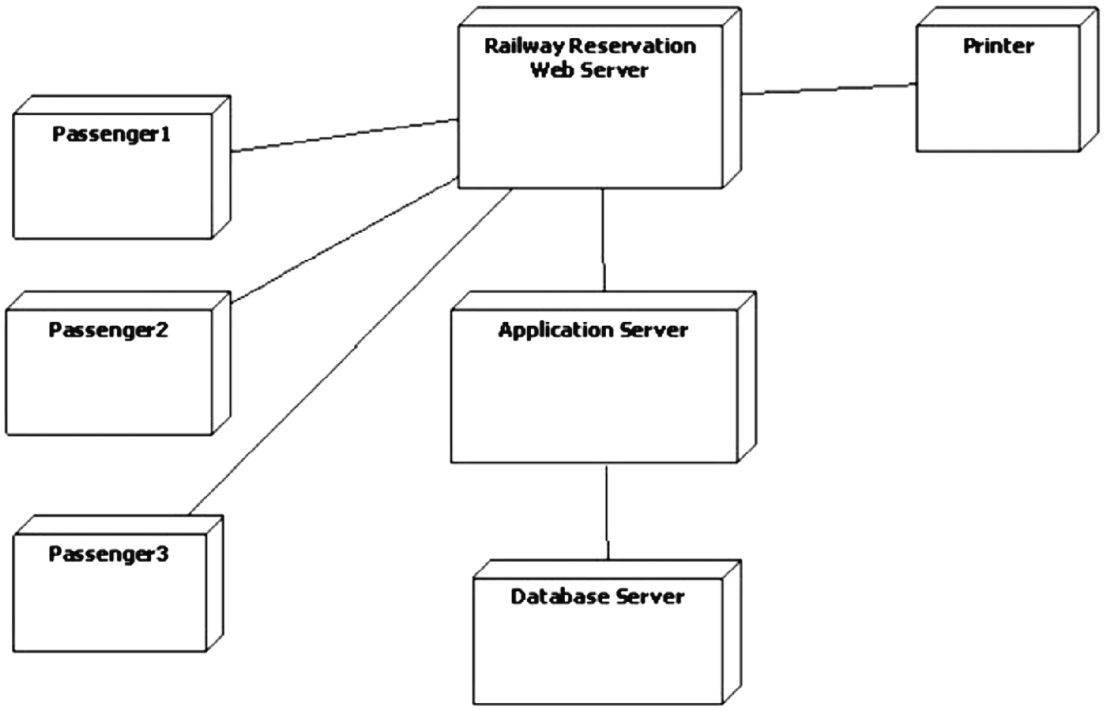
- You can show the models the components of Railway Reservation System.
- Model the database schema of Railway Reservation System
- Model the executables of an application of Railway Reservation System
- Model the system's source code of Railway Reservation System



Component diagram for Online Ticket Reservation System


view classses
Home page
check availability
fill form
modify form
book ticket
cancel ticket
Resevation class
Passenger
Ticket
refund
clerk
Session
Payment
Acess system
Acess database

Railway Reservation System Deployment Diagram



10. To perform various testing using the testing tool unit testing, integration testing

Ans :

Unit Tests

Starting from the bottom the first test level is "Unit Testing". It involves checking that each feature specified in the "Component Design" has been implemented in the component.

In theory an independent tester should do this, but in practice the developer usually does it, as they are the only people who understand how a component works. The problem with a component is that it performs only a small part of the functionality of a system, and it relies on co-operating with other parts of the system, which may not have been built yet. To overcome this, the developer either builds, or uses special software to trick the component into believe it is working in a fully functional system.

The summary of unit tests is provided below

For user

Unit	Purpose
Search Train	This unit search the trains.
Train details	This unit shows the trains of a particular source to destination in a particular date and a specific seat .
Book ticket	This unit user can select a particulartrain and book ticket .
Login	This unit login the registered user and create an account for a new user.
Fill the bookingform	User fill the form to book ticket.
payment	User fill the form and pay the money with the help of credit card.
Ticket no	This unit allows to show his ticket no.
Cancel ticket	This unit allows user to cancel ticket.

For Administrator

Unit	Purpose
Administrator Login	This unit detects the authorization of the Administrator.
Change	the traindetails. This unit allows the administrator tochange the train details.

Integration Testing

As the components are constructed and tested they are then linked together to check if they work with each other. It is a fact that two components that have passed all their tests, when connected to each other produce one new component full of faults. These tests can be done by specialists, or by the developers.

Integration Testing is not focused on what the components are doing but on how they communicate with each other, as specified in the "System Design". The "System Design" defines relationships between components.

The tests are organized to check all the interfaces, until all the components have been built and interfaced to each other producing the whole system.

System Testing

Once the entire system has been built then it has to be tested against the "System

Specification" to check if it delivers the features required. It is still developer focused, although specialist developers known as systems testers are normally employed to do it.

In essence System Testing is not about checking the individual parts of the design, but about checking the system as a whole. In fact it is one giant component.

System testing can involve a number of specialist types of test to see if all the functional and non - functional requirements have been met. In addition to functional requirements these may include the following types of testing for the non - functional requirements:

- **Performance**- Are the performance criteria met?
- **Volume**-Can large volumes of information be handled?
- **Stress**- Can peak volumes of information be handled?
- **Documentation** - Is the documentation usable for the system?
- **Robustness** - Does the system remain stable under adverse circumstances? The system was found to perform its function properly under all conditions.

Acceptance Testing

Acceptance Testing checks the system against the "Requirements". It is similar to systems testing in that the whole system is checked but the important difference is the change in focus:

Systems testing checks that the system that was specified has been delivered. Acceptance Testing checks that the system will deliver what was requested.

The customer should always do acceptance testing and not the developer. The customer knows what is required from the system to achieve value in the business and is the only person qualified to make that judgment. This testing is more of getting the answer forweather is the software delivered as defined by the customer. It's like getting a green flag from the customer that the software is up to the expectation and ready to be used.

4. E-Commerce Systems

1. Phases in software development project, overview, need, coverage of topics.

Ans :

Online Shopping System helps in buying of goods, products and services online by choosing the listed products from website (E-Commerce site). The proposed system helps in building a website to buy, sell products or goods online using internet connection. Purchasing of goods online, user can choose different products based on categories, online payments, delivery services and hence covering the disadvantages of the existing system and making the buying easier and helping the vendors to reach wider market.

The system helps in buying of goods, products and services online by choosing the listed products from website (E-Commerce site).

In day to day life, we will need to buy lots of goods or products from a shop. It may be food items, electronic items, house hold items etc. Now a days, it is really hard to get some time to go out and get them by ourselves due to busy life style or lots of works. In order to solve this, B2C E-Commerce websites have been started. Using these websites, we can buy goods or products online just by visiting the website and ordering the item online by making payments online.

This existing system of buying goods has several disadvantages. It requires lots of time to travel to the particular shop to buy the goods. Since everyone is leading busy life now a days, time means a lot to everyone. Also there are expenses for travelling from house to shop. More over the shop from where we would like to buy some thing may not be open $24 \times 7 \times 365$. Hence we have to adjust our time with the shopkeeper's time or vendor's time.

In order to overcome these, we have e-commerce solution, i.e one place where we can get all required goods/products online. The proposed system helps in building a website to buy, sell products or goods online using internet connection. Purchasing of goods online, user can choose different products based on categories, online payments, delivery services and hence covering the disadvantages of the existing system and making the buying easier and helping the vendors to reach wider market.

Advantages of the new System Proposed

- Choose products faster and easier at one place.
- Saves time of travelling to the vendor/seller's place.
- Good/Trusted & Tension free delivery services. Products bought online will be delivered to the footsteps of the buyer free of cost (may be varied based on the vendor/seller).
- Alerts and real time reporting through Emails (to both vendor as well as buyer).
- Reports generated can be saved for future references.
- Inventory reports for the vendor/seller on daily, monthly, yearly basis.

Future Scope of the Project

- Most generic consumer to consumer e-commerce website, which covers almost all possible categories, with 2 level listing.
- Maximize benefits and minimize the disadvantages of a common e-commerce website.
- User friendly, Vendor friendly environment.

2. To assign the requirement engineering tasks

Ans :

Any shopping website that is concerned will be able to attract more customers only if the items purchased will be delivered on time. The user interface should be simple and easy to understand even by the common people. The back end should have a strong database.

Functional Requirement

This section provides requirement overview of the system. Various functional modules that can be implemented by the system will be:-

Description

- **Registration:** If customer wants to buy the product then he/she must be registered, unregistered user can’t go to the shopping cart.
- **Login:** Customer logins to the system by entering valid user id and password for the shopping.
- **Changes to Cart:** Changes to cart means the customer after login or registration can make order or cancel order of the product from the shopping cart.
- **Payment:** For customer there are many type of secure billing will be prepaid as debit or credit card, post paid as after shipping, check or bank draft. The security will provide by the third party like Pay-Pal etc.
- **Logout:** After the payment or surf the product the customer will logged out.
- **Report Generation:** After all transaction the system can generate the portable document file (.pdf) and then sent one copy to the customer’s Email-address and another one for the system data base to calculate the monthly transaction .

Technical Issues

This system will work on client-server architecture. It will require an internet server and which will be able to run PHP application. The system should support some commonly used browser such as IE etc.

Interface Requirement

Various interfaces for the product could be-

- Login Page
- Registration Form

There will be a screen displaying information about product that the shop having. If the customers select the buy button then another screen of shopping cart will be opened. After all transaction the system makes the selling report as portable document file (.pdf) and sent to the customer E-mail address.

The features that can be included in the online shopping platform application are as follows:

- **Customer database management system**: The information of the customers doing the online shopping must be maintained in a well organized way.
- **Description**: There should be proper description that must be given to the items that are kept for sale.
- **Price**: The price of the item should also be mentioned along with the description to prevent any further confusion.

- **Category**: There must be various categories like clothes, accessories, electronic gadgets and so on which will help in easy searching for the items by the customers.
- **Delivery boy**: There should be some delivery boys available to deliver the items that have been purchased by the customers. Each area must be assigned different delivery boys
- **On time delivery**: The items purchased by the customers must be delivered on time without any delay.

Module Description

The system consists of the following modules :

- **Master Maintenance:** This module consists of information about the products and services. This includes two sub-modules, **Product master** and **Price master**.
- **Product master** includes the information about particular product, such as product number, item, name, category, images of products, description, features, and constraints of products. All these will be entered to the database through product master and hence made available in the website. In Other words product master is the admin area for the vendors/sellers where they can put information about their products which are to be displayed in the website.
- **Price master** deals with the cost of the product, discounts applicable for the particular product of a vendor/seller.
- **Transactions:** In this module, management of shopping cart is done. This module will add the bought item to the shopping cart, where all items that are to be purchased can be reviewed once again after the item is bought from the cart. Payment will be done on Delivery of the items (Cash On Delivery).
- **Reporting:** In this module all reports will be generated. Whenever a item is sold, or customer orders a product, its vendor should be sent an alert via email immediately so that he can ship that item soon. This module has 3 sub modules; Stock Reports, Order Reports and Delivery Reports.
- **Stock Report** will produce reports of the quantity of the products available and product status.
- **Order Report** will have the list of products ordered and the customer details who have bought that product, which are undelivered.
- **Delivery Reports** will generate products list, which are delivered to customers.
- **House Keeping:** This module takes care of data which are older than a certain period. It will allow the vendor to archive the reports generated or transaction and business history reported by Reporting module.

3. To perform the system analysis: Requirement analysis, SRS.

Ans :

User Characteristics

There are 3 kinds of users for the proposed system.

- **Administrators**: Administrators are the ones who adds or administers the categories for the products, and administers the Vendors.
- **Vendors/Sellers** : Vendors/Sellers will add their products to the database, which will be seen in the website to the end users or say customers who can buy the products by selecting the one they need. Vendors will have the special privileges than the end users, and have ability to manage the products added by them.

- **End Users/Customers**: The end user will be the one who visits the website and buys products online from the ones added by the Vendors/Sellers.

User Interfaces

Each part of the user interface intends to be as user friendly as possible. The fonts and buttons used will be intended to be very fast and easy to load on web pages. The pages will be kept light in space so that it won't take a long time for the page to load.

Hardware Interfaces

- **Processor:** Pentium or Higher.
- **RAM:** 312MB or Higher.

Software Interfaces

- **Operating System:** Unix, Linux, Mac, Windows etc..
- **Development tool:** PHP, Hypertext Preprocessor, JavaScript, Ajax
- **Data Base:** MySQL

Communication Interface

The Website Order system shall send an e-mail confirmation to the customer that the items they ordered will be delivered to the shipping address along with user identification.

Functional Requirements

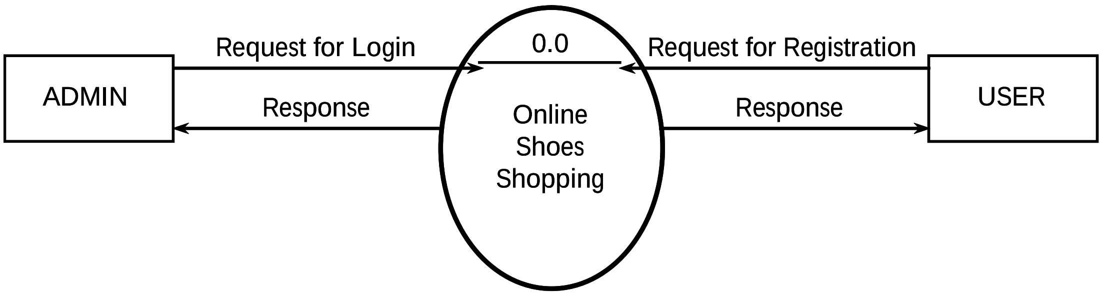
- **Master Maintenance**: This module consists of information about the products and services. This includes two sub-modules, Product master and Price master.
- **Product Master:** Product master includes the information about particular product, such as product number, item, name, category, images of products, description, features, constraints of products, which are to be displayed on the website.
- **Price master:** Price master deals with the cost of the product, discounts applicable for the particular product of a vendor/seller.
- **Transactions:** All transactions undergoing in the website will be controlled and managed by this module. Transactions in the sense, Shopping Cart management.
- **Reporting:** This module deals with report management of the entire system. This includes three sub-modules Stock Report, Order Report and Delivery Report.
- **Order Report:** Order Report will have the list of products ordered and the customer details who have bought that product, which are undelivered.
- **Delivery Report:** Delivery Reports will generate products list, which are delivered to customers.
- **Housekeeping Module:** This module deals with backing up of data for future references and hence to reduce the database size.

4. To perform the function-oriented diagram: DFD and Structured chart.

Ans :

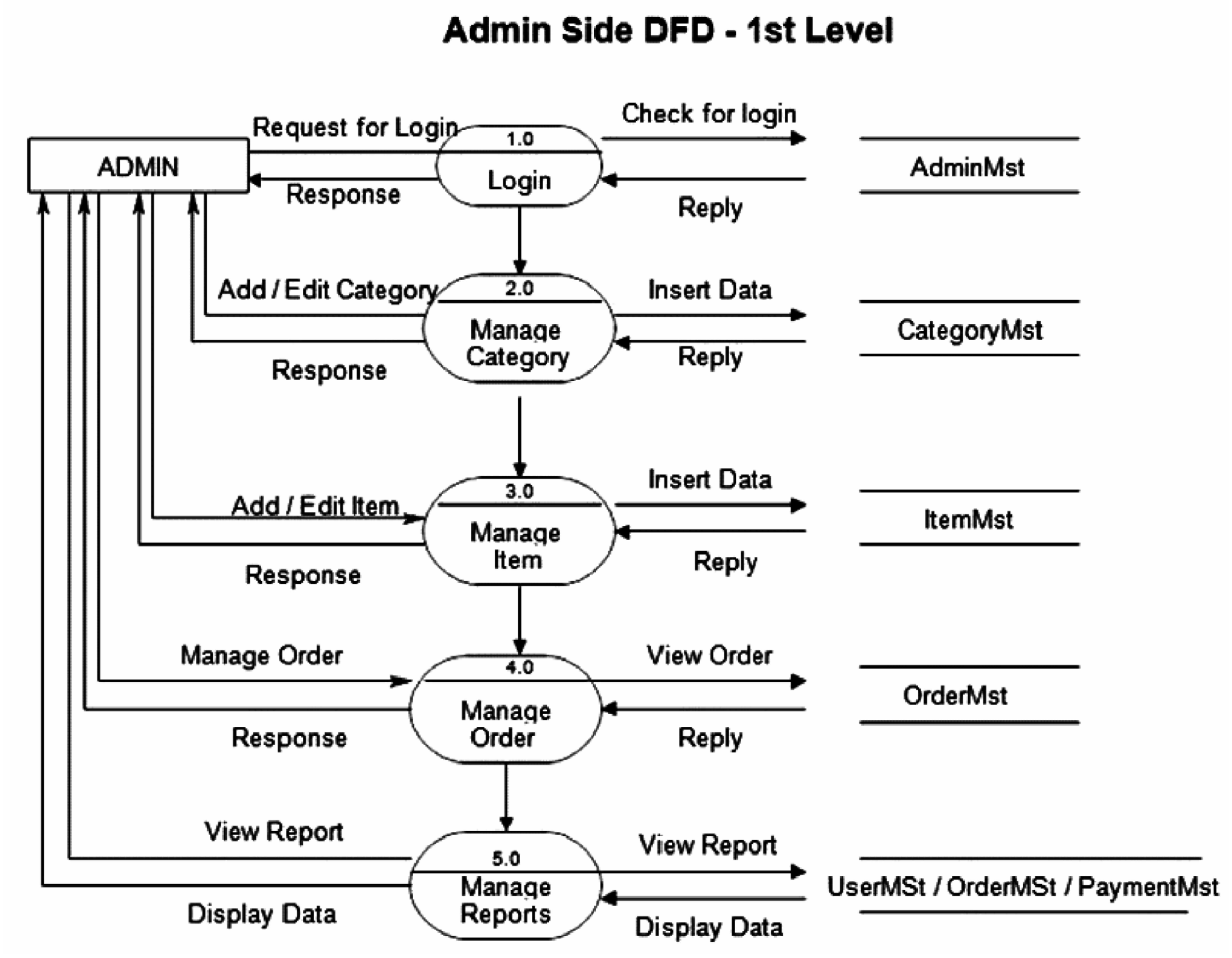
The context level data flow diagram (dfd) is describe the whole system. The (o) level dfd describe the all user module who operate the system. Below data flow diagram of online shopping site shows the two user can operate the system Admin and Member user.

DFD level-0

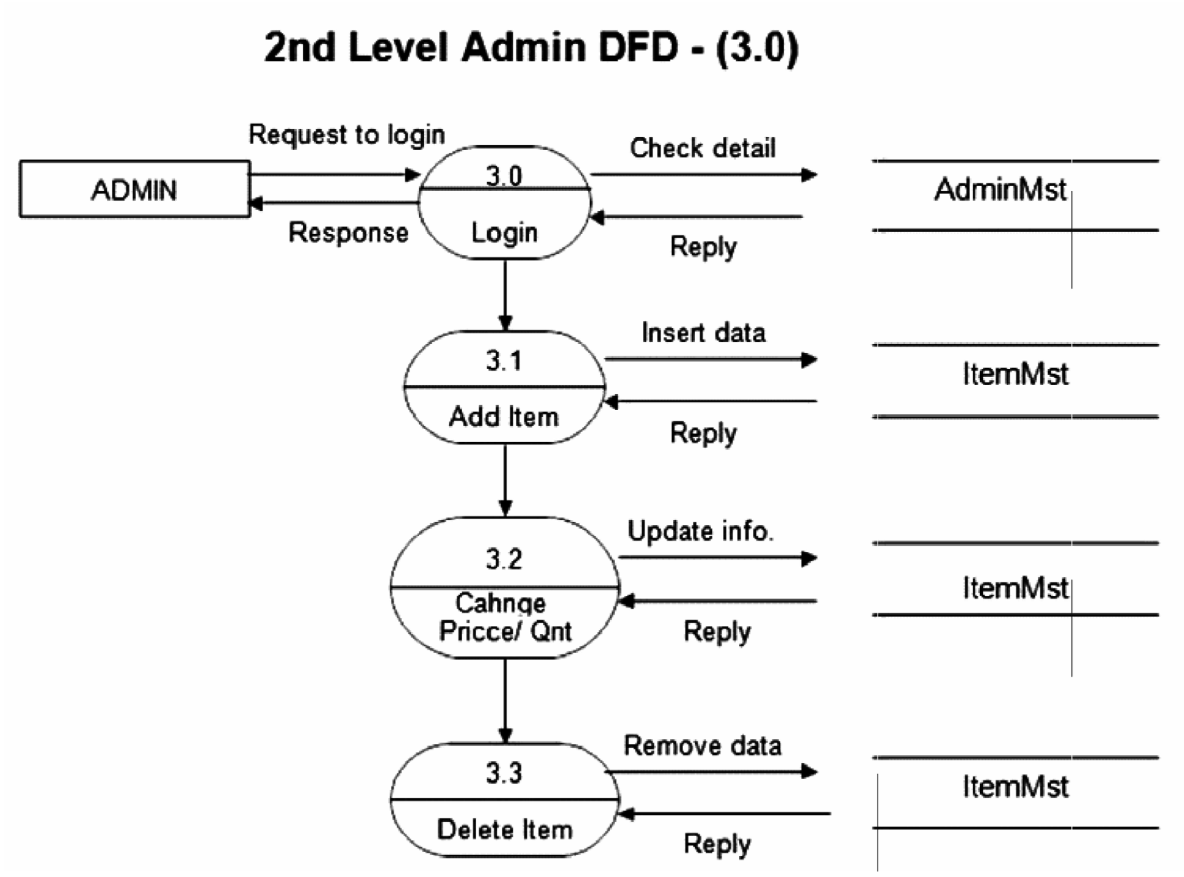


1st Level Admin Side DFD

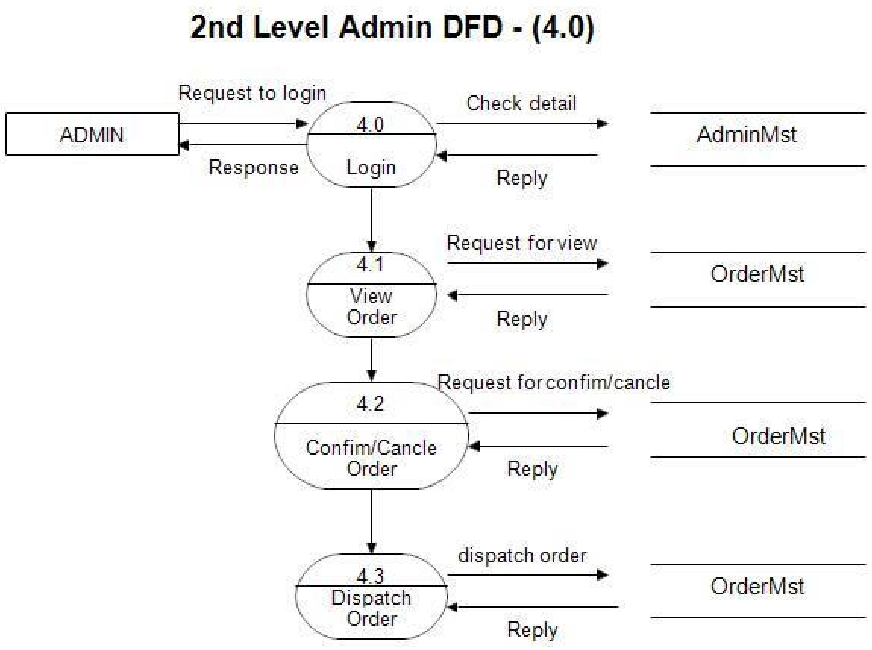
The Admin side DFD describe the functionality of Admin, Admin is a owner of the website. Admin can first add category of item and then add items by category wise. and admin can manage order and payment detail.



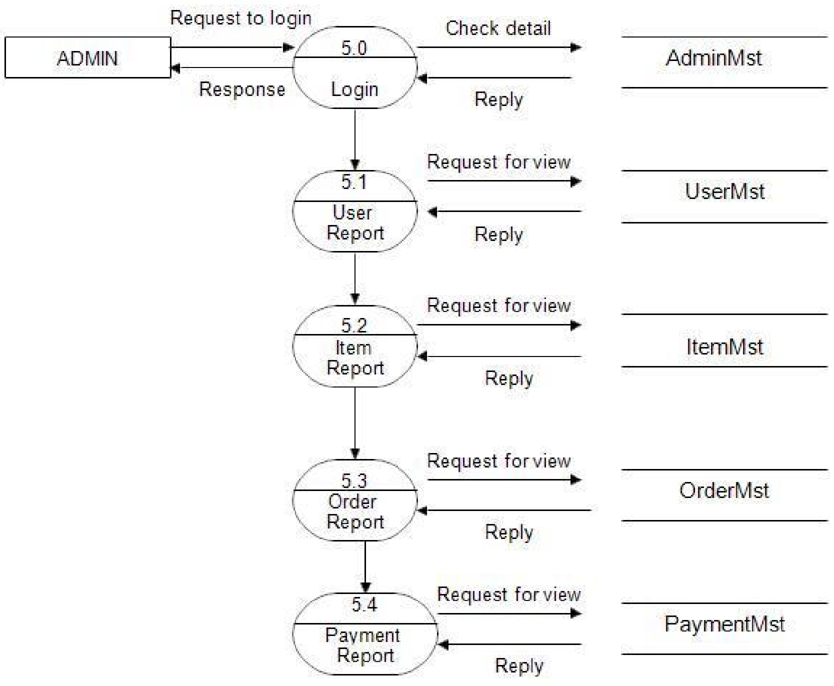
2nd Level – Admin side DFD (3.0)



2nd Level – Admin side DFD (4.0)

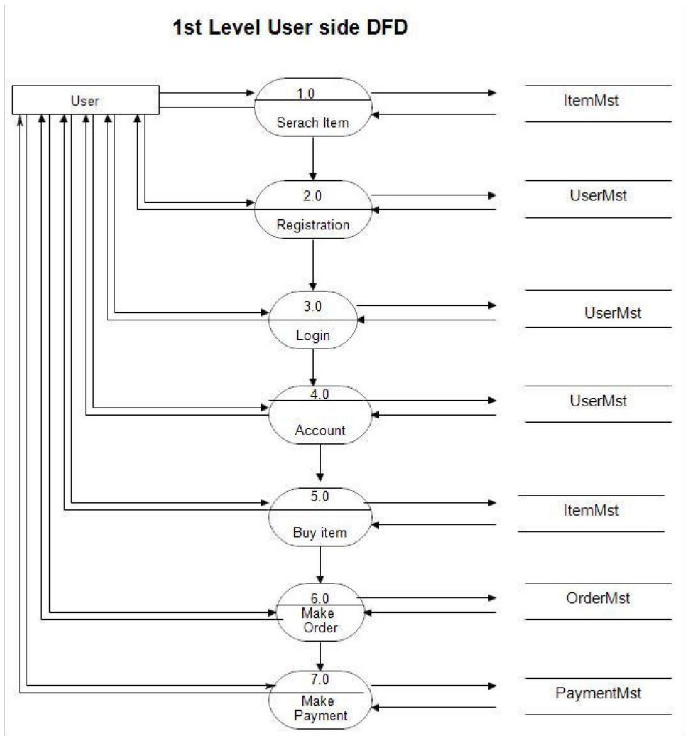


2nd Level – Admin side DFD (5.0)

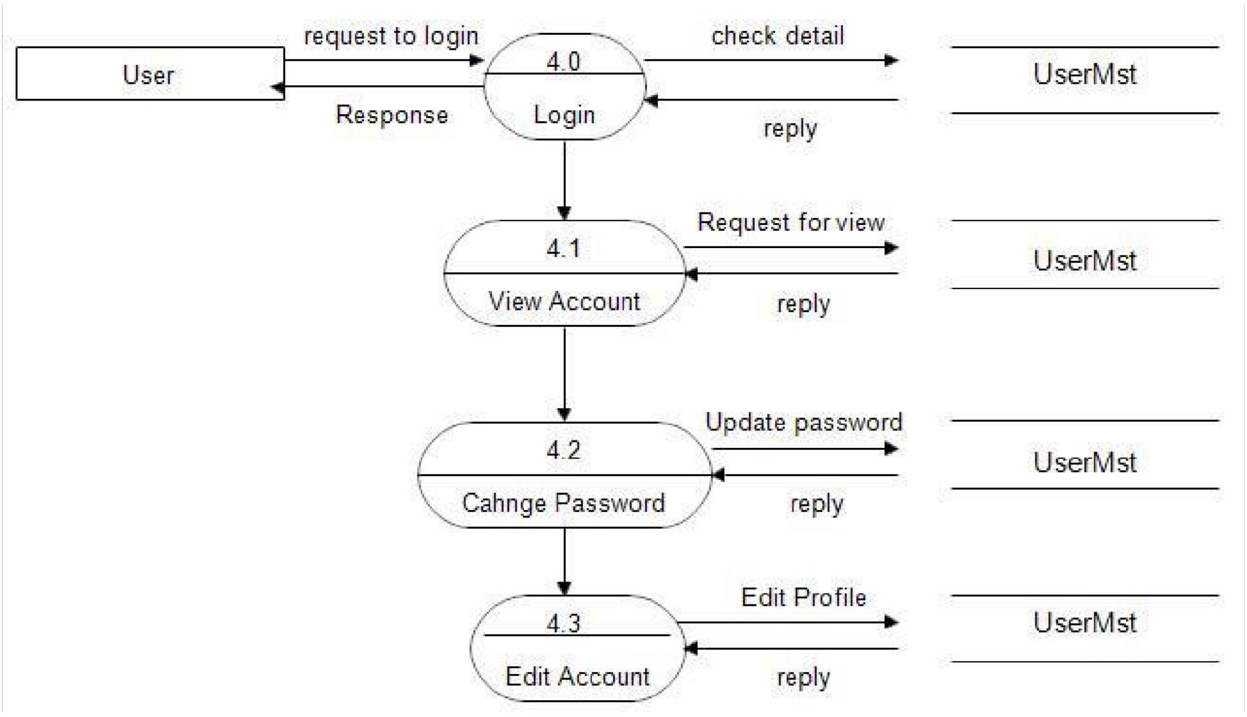


1st level – User side Data flow Diagram

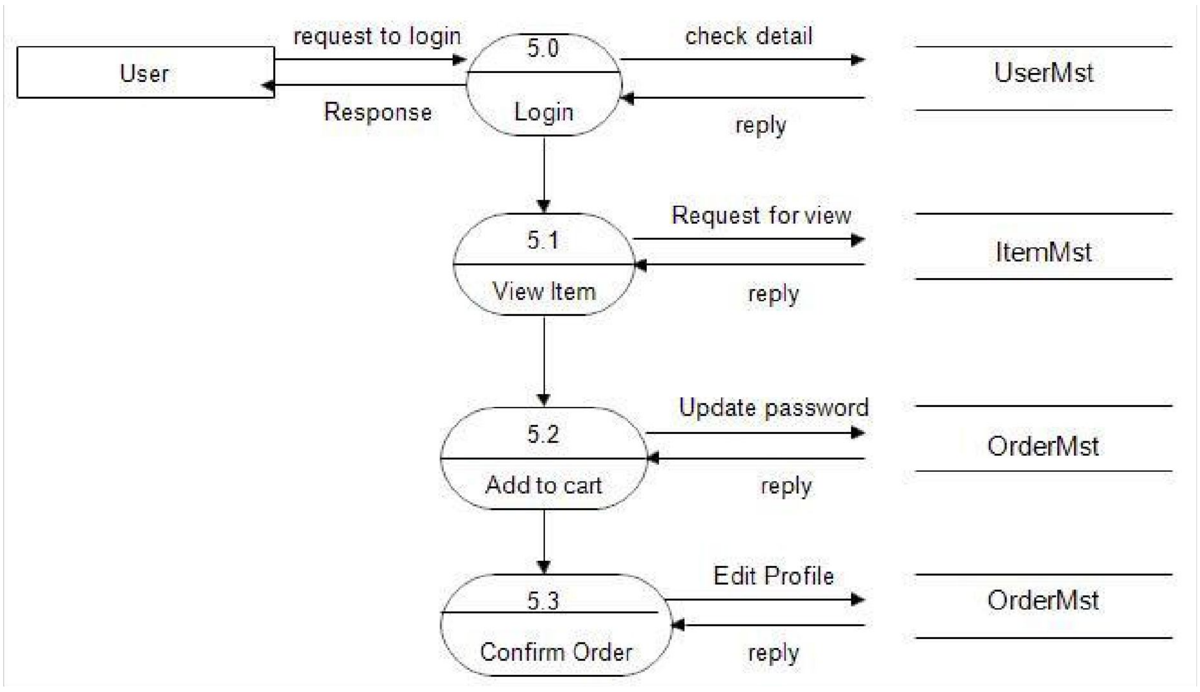
The user is all people who operate or visit our website. User is a customer of a website. User can first select product for buy, user must have to register in our system for purchase any item from our website. after register he can login to site and buy item by making online payment through any bank debit card or credit card.



2nd level – User side DFD (4.0)



2nd level – User side DFD (5.0)

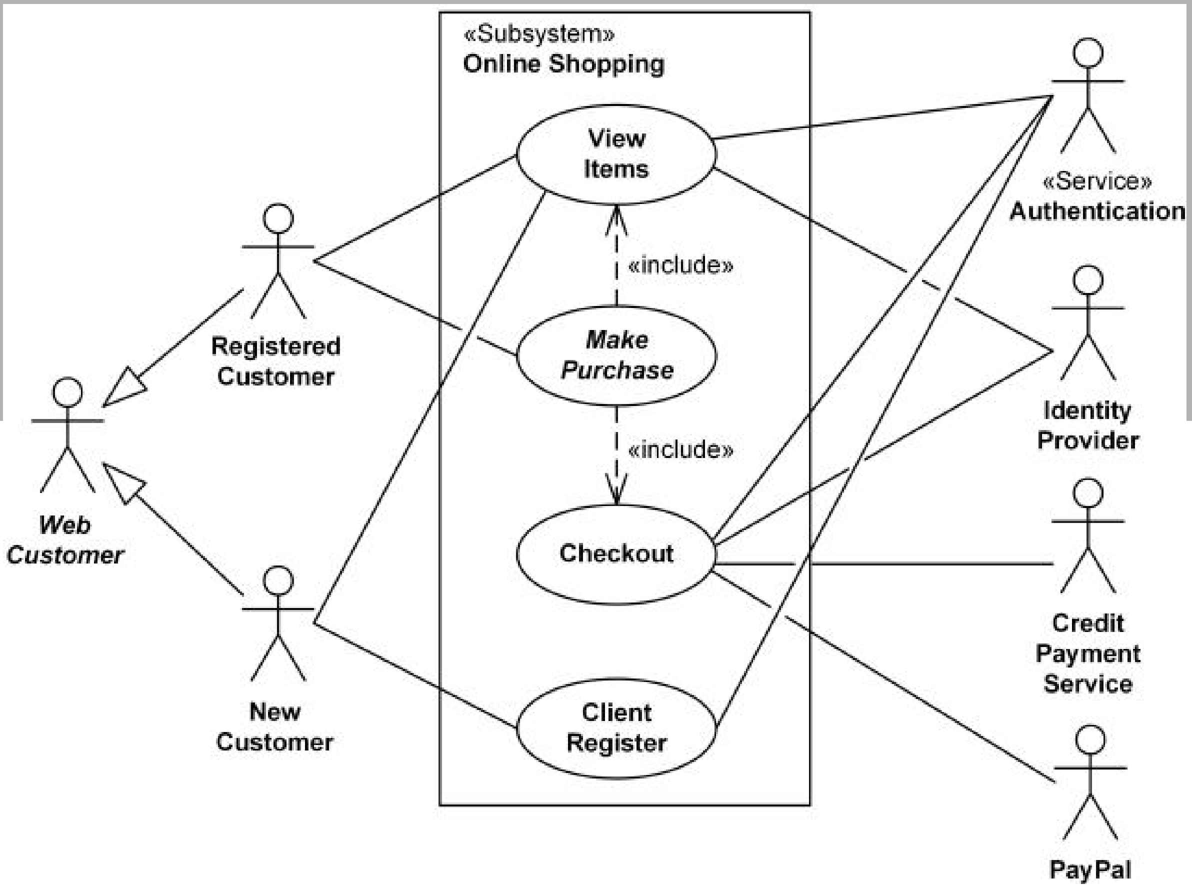


5. To perform the user's view analysis: Use case diagram.

Ans :

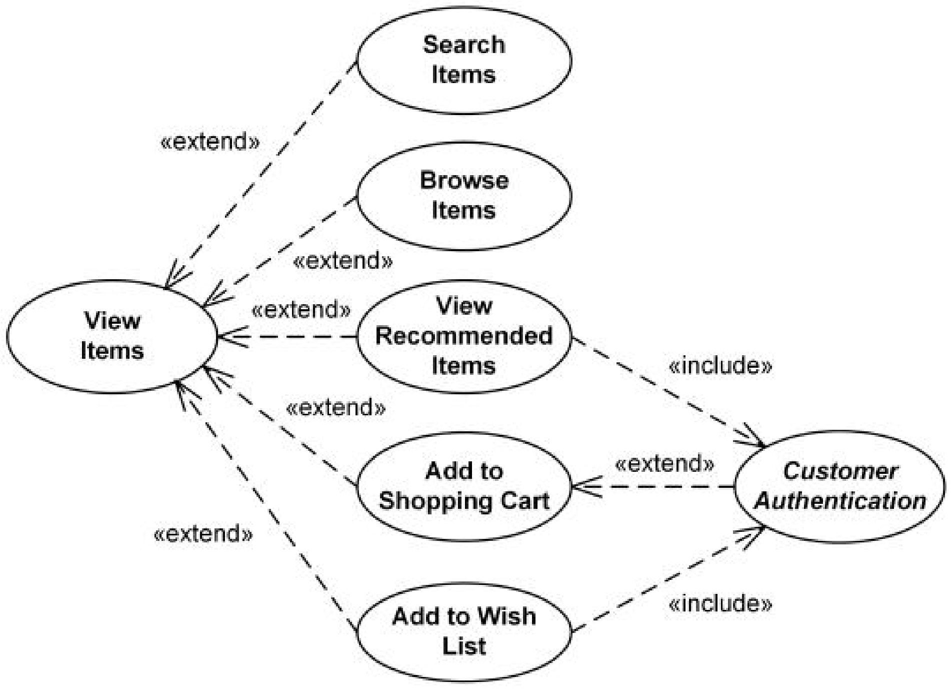
Use Case Diagram

Web Customer actor uses some web site to make purchases online. Top level use cases are View Items, Make Purchase and Client Register. View Items use case could be used by customer as top level use case if customer only wants to find and see some products. This use case could also be used as a part of Make Purchase use case. Client Register use case allows customer to register on the web site, for example to get some coupons or be invited to private sales. Note, that Checkout use case is included use case not available by itself - checkout is part of making purchase. Except for the Web Customer actor there are several other actors which will be described below with detailed use cases.

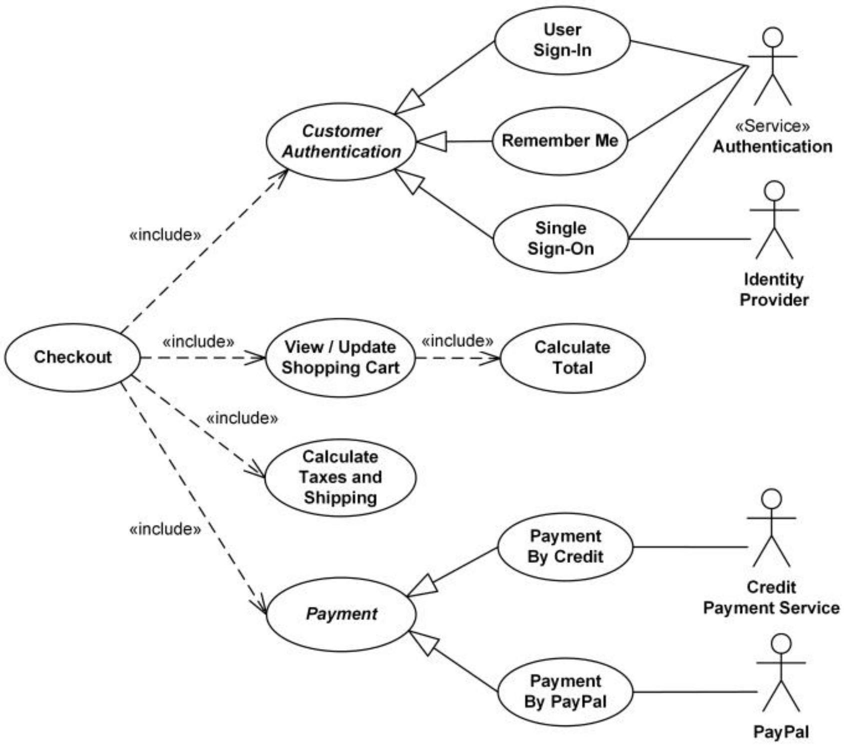


View Items use case is extended by several optional use cases - customer may search for items, browse catalog, view items recommended for him/her, add items to shopping cart or wish list. All these use cases are extending use cases because they provide some optional functions allowing customer to find item.

Customer Authentication use case is included in View Recommended Items and Add to Wish List because both require the customer to be authenticated. At the same time, item could be added to the shopping cart without user authentication.



Checkout use case includes several required uses cases. Web customer should be authenticated. It could be done through user login page, user authentication cookie ("Remember me") or Single Sign-On (SSO). Web site authentication service is used in all these use cases, while SSO also requires participation of external identity provider.



6. To draw the structural view diagram: Class diagram, object diagram

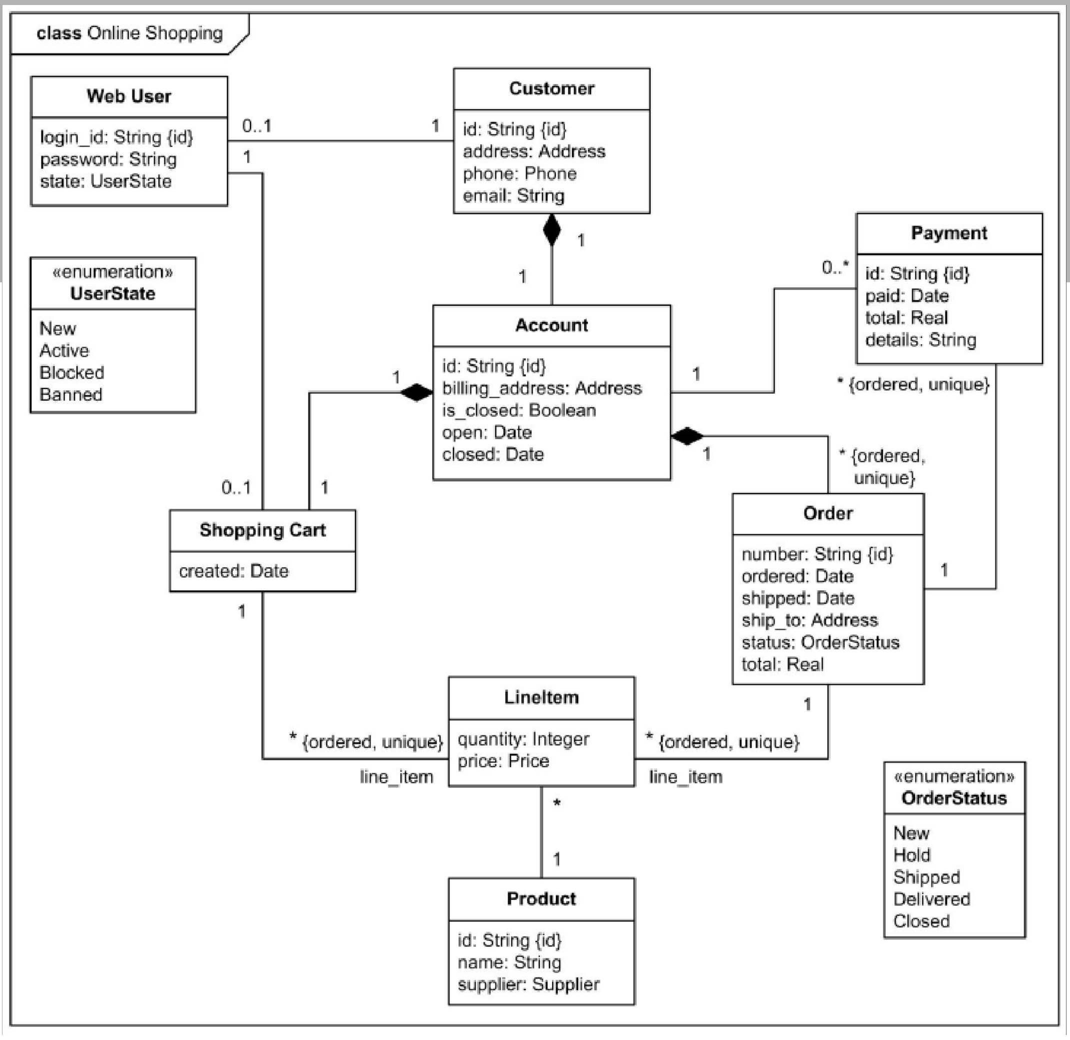
Ans :

Class Diagram

Each customer has unique id and is linked to exactly one account. Account owns shopping cart and orders. Customer could register as a web user to be able to buy items online. Customer is not required to be a web user because purchases could also be made by phone or by ordering from catalogues. Web user has login name which also serves as unique id. Web user could be in several states - new, active, temporary blocked, or banned, and be linked to a shopping cart. Shopping cart belongs to account.

Account owns customer orders. Customer may have no orders. Customer orders are sorted and unique. Each order could refer to several payments, possibly none. Every payment has unique id and is related to exactly one account.

Each order has current order status. Both order and shopping cart have line items linked to a specific product. Each line item is related to exactly one product. A product could be associated to many line items or no item at all.

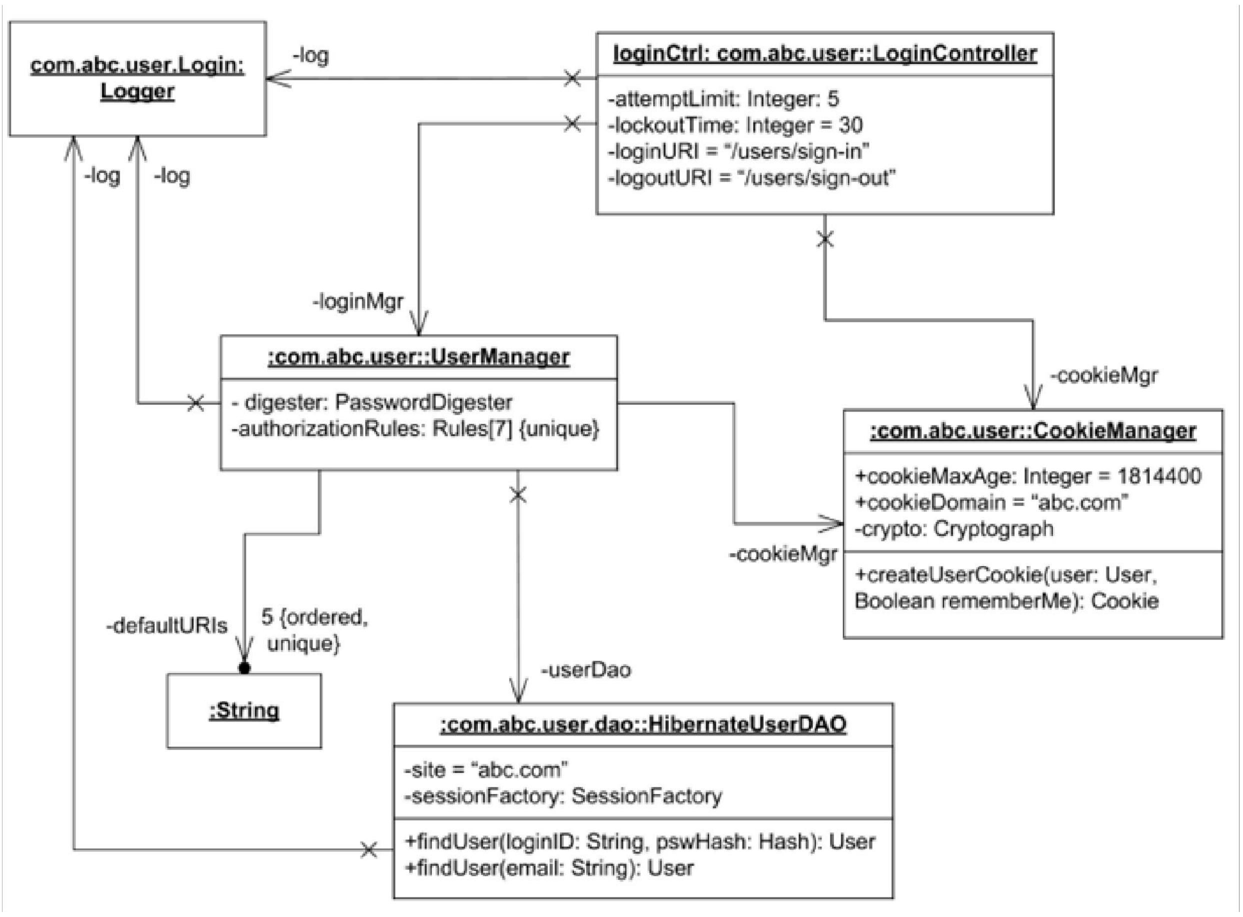


Object Diagram

This is an example of object diagram which shows some runtime objects related to web user login process. Class instance loginCtrl of the LoginController has several slots with structural features of Integer and String types and corresponding value specifications.

The instance of LoginController is also associated with instances of UserManager, CookieManager, and Logger. LoginController, UserManager, and HibernateUserDAO (Data Access Object) share a single instance of Logger.

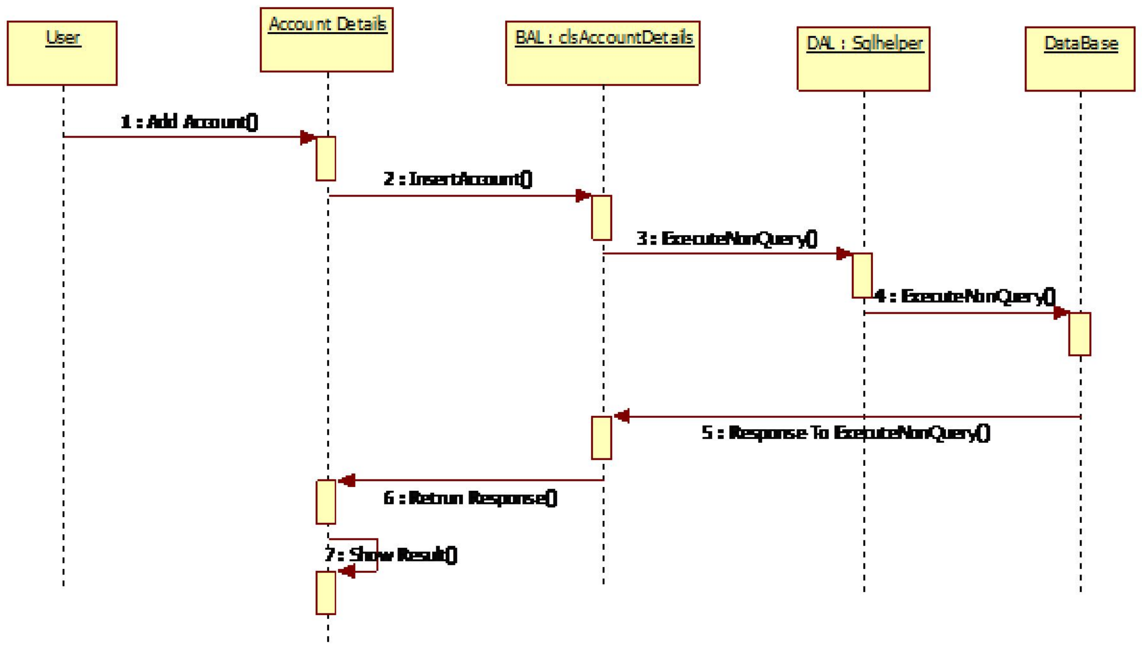
User Manager has private attribute defaultURIs which is ordered collection (array) of 5 unique Strings. Instance of the CookieManager has two public structural features with specified values.

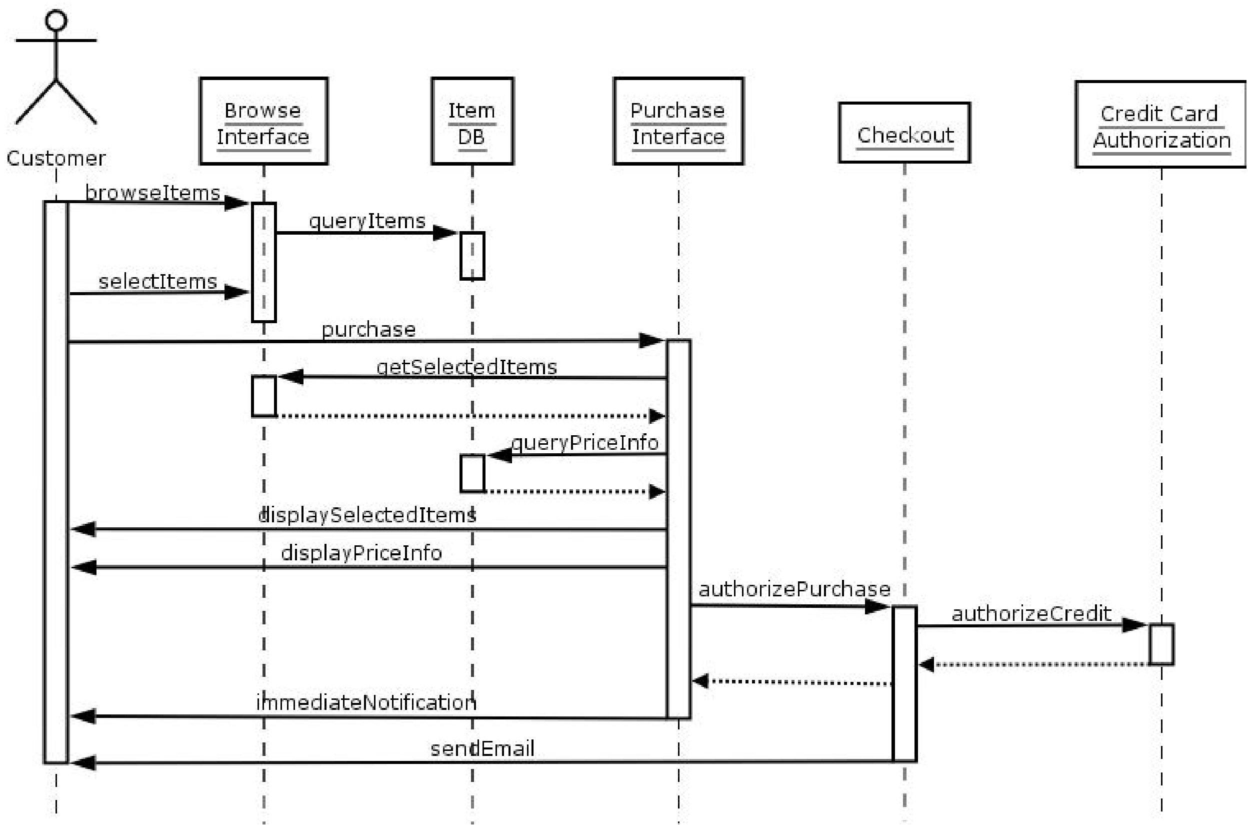

com.abc.user.Login: Logger
-log
loginCtrl: com.abc.user::LoginController
-attemptLimit: Integer: 5
-lockoutTime: Integer = 30
-loginURI = "/users/sign-in"
-logoutURI = "/users/sign-out"
-log
-log
-loginMgr
:com.abc.user::UserManager
- digester: PasswordDigester
-authorizationRules: Rules[7] {unique}
-cookieMgr
:com.abc.user::CookieManager
+cookieMaxAge: Integer = 1814400
+cookieDomain = "abc.com"
-crypto: Cryptograph
+createUserCookie(user: User, Boolean rememberMe): Cookie
-cookieMgr
-defaultURIs
5 {ordered, unique}
-userDao
:String
:com.abc.user.dao::HibernateUserDAO
-site = "abc.com"
-sessionFactory: SessionFactory
+findUser(loginID: String, pswHash: Hash): User
+findUser(email: String): User

7. To draw the behavioural view diagram: Sequence diagram, Collaboration diagram.

Ans :

Draw a sequence diagram for online shopping. If the customer is using the website for the first time to order he/she needs to register. After login a customer can select the items and view their details. The items can be added to the shopping cart one by one. The order will be placed at the end. Once the customer wishes to place the order the system will be requesting to enter the credit card details to finalize the payment. The card details are verified from the bank. On receiving the verification of the payment the order of the customer is acknowledged and shipped.

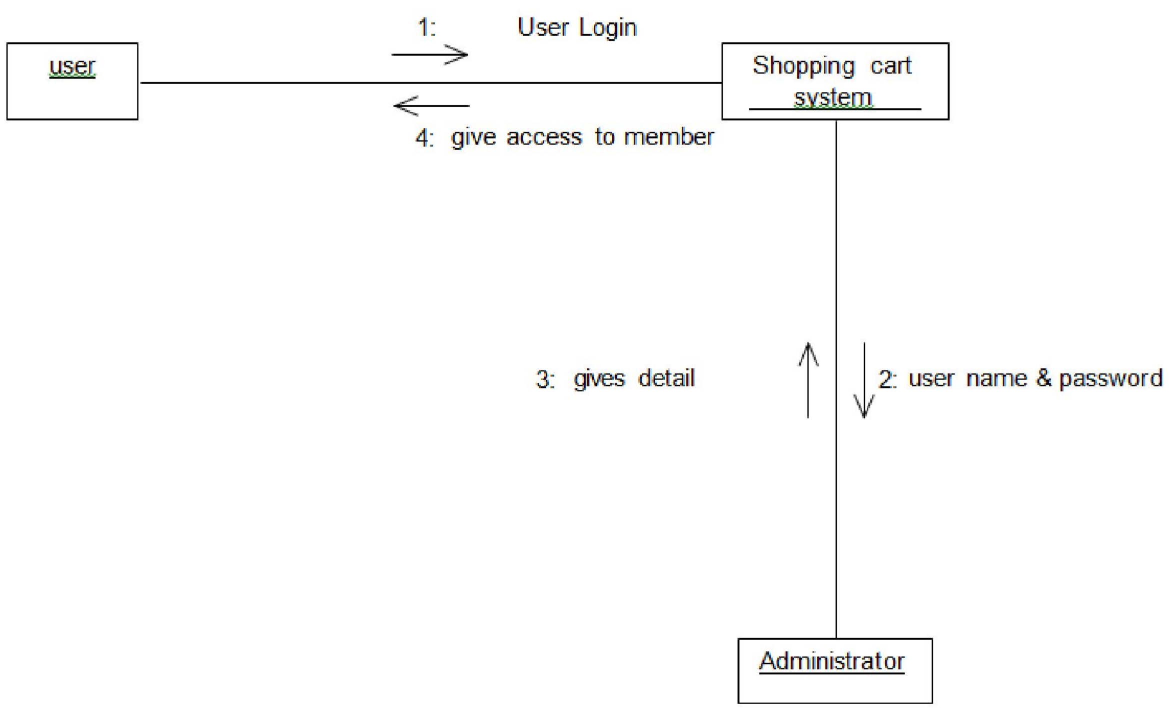

User
Account Details
BAL : clsAccountDetails
DAL : Sqlhelper
DataBase
1 : Add Account()
2 : InsertAccount()
3 : ExecuteNonQuery()
4 : ExecuteNonQuery()
5 : Response To ExecuteNonQuery()
6 : Retrun Response()
7 : Show Result()


Customer
Browse Interface
Item DB
Purchase Interface
Checkout
Credit Card Authorization
browseItems
queryItems
selectItems
purchase
getSelectedItems
queryPriceInfo
displaySelectedItems
displayPriceInfo
authorizePurchase
authorizeCredit
immediateNotification
sendEmail

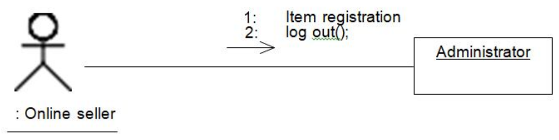
Collaboration diagrams

They are also interaction diagrams. They convey the same information as sequence diagrams, but they focus on object roles instead of the times that messages are sent. In a sequence diagram, object roles are the vertices and messages are the connecting links.

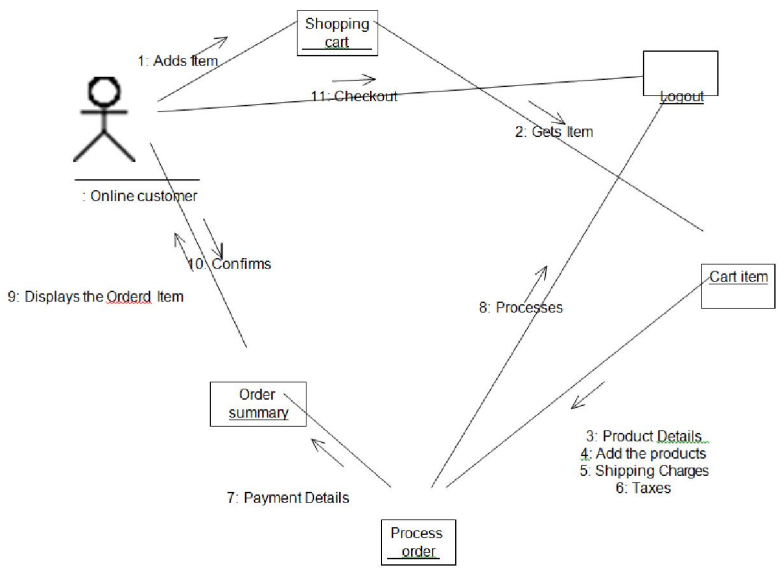
- Collaboration diagrams show (used to model) how objects interact and their roles.
- They are very similar to sequence diagrams. Actually they are considered as a **cross** between class and sequence diagram.
- Sequence Diagrams are arranged according to Time.
- Collaboration Diagrams represent the structural organization of object.
- Forms a context for interactions.
  - May realize use cases.
  - May be associated with operations.
  - May describe the static structure of classes.
- Collaboration diagrams contain the following:
  - Class roles (subsystems/objects/classes/actors/ external systems) as before.
  - Association roles (pathways or links over which messages flow).
  - Message flows (messages sent between class roles).



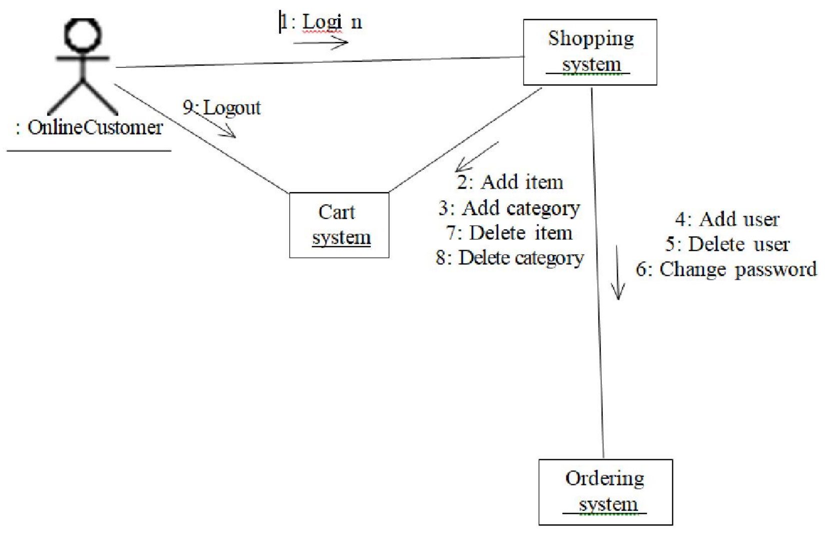
Collaboration Diagram for User registration



Collaboration Diagram for Seller registeration



Collaboration Diagram for Online Customer



Collaboration Diagram for Online Administrator

8. To draw the behavioural view diagram: State-chart diagram, Activity diagram.

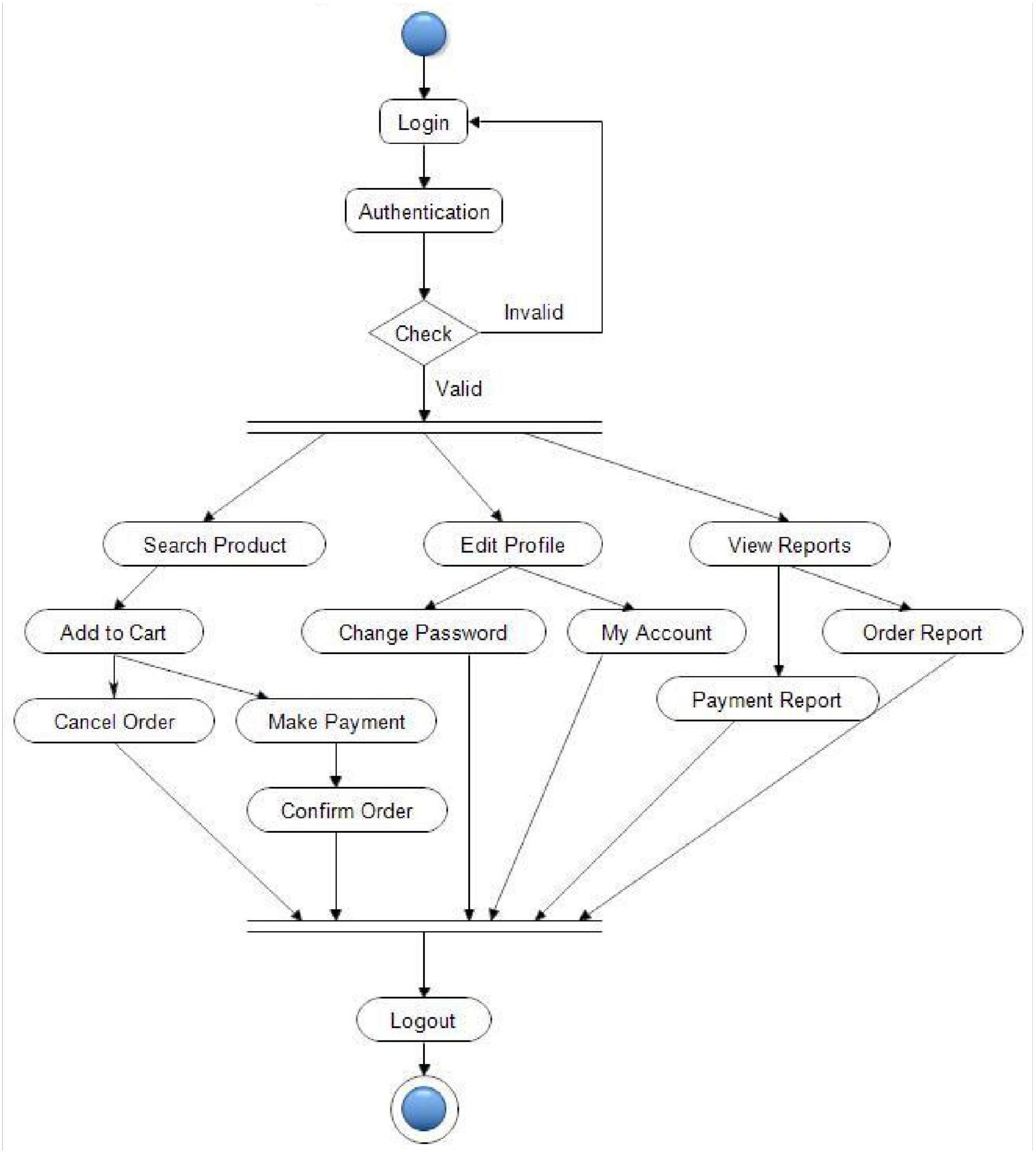
Ans :

Activity Diagram

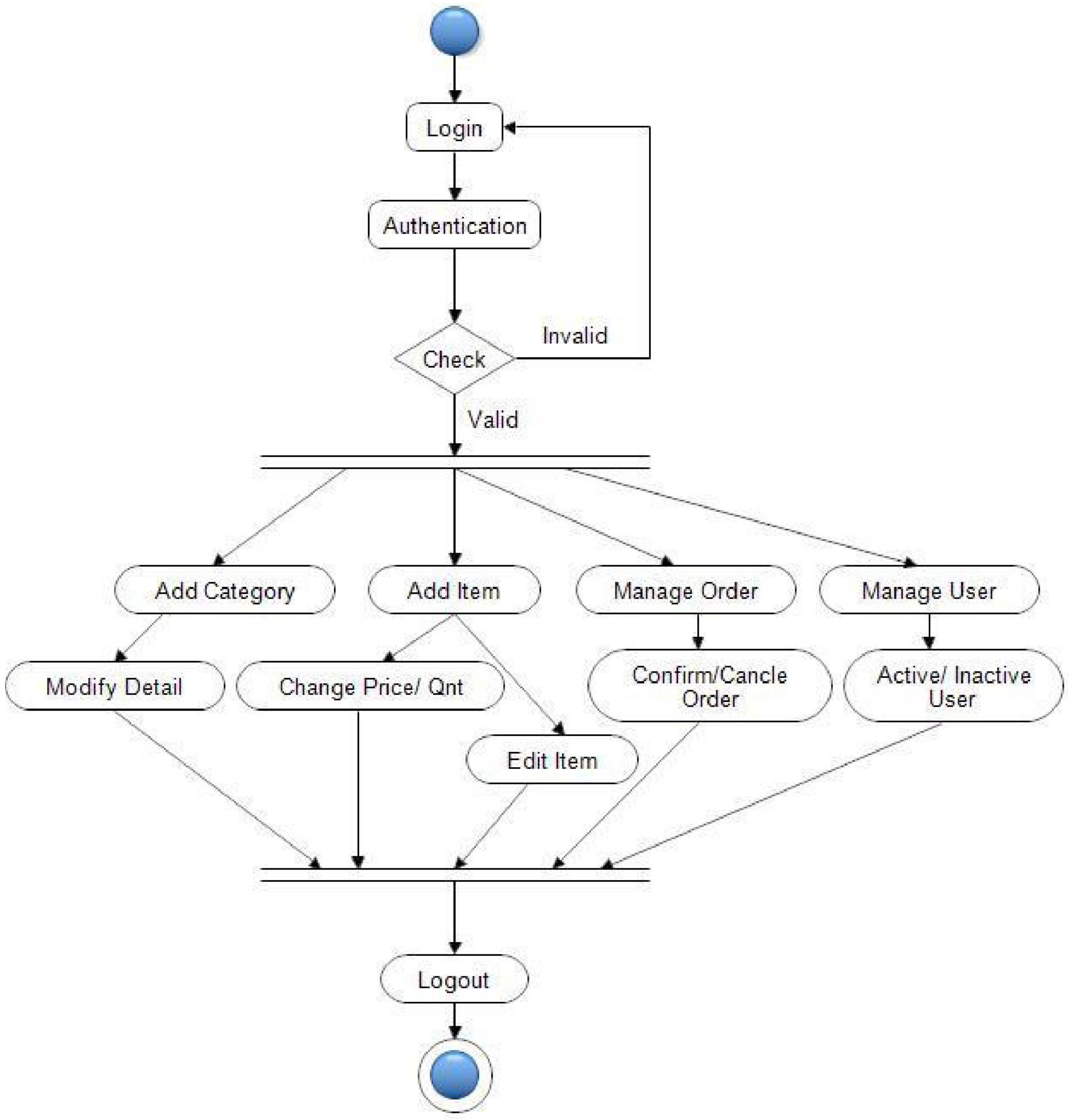
The activity diagram used to describe flow of activity through a series of actions. Activity diagram is a important diagram to describe the system. The activity described as a action or operation of the system.

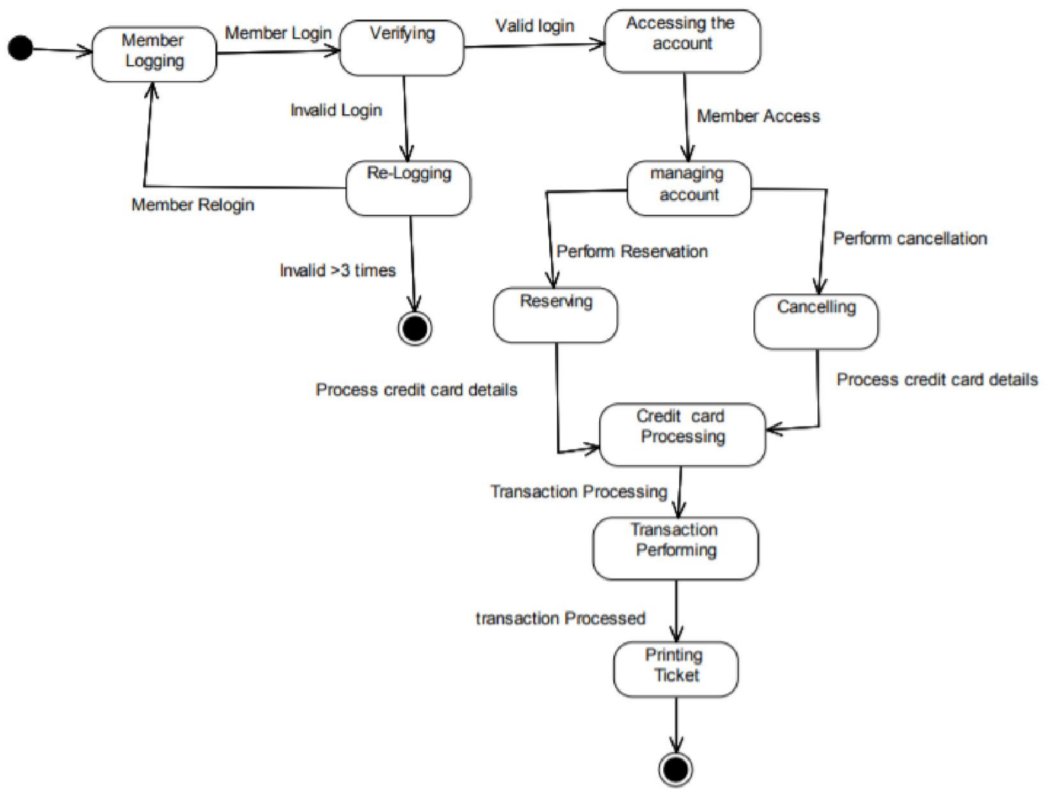
Activity Diagram for User Side

In User side activity diagram describe all the functionality or operation of users can do on our website.

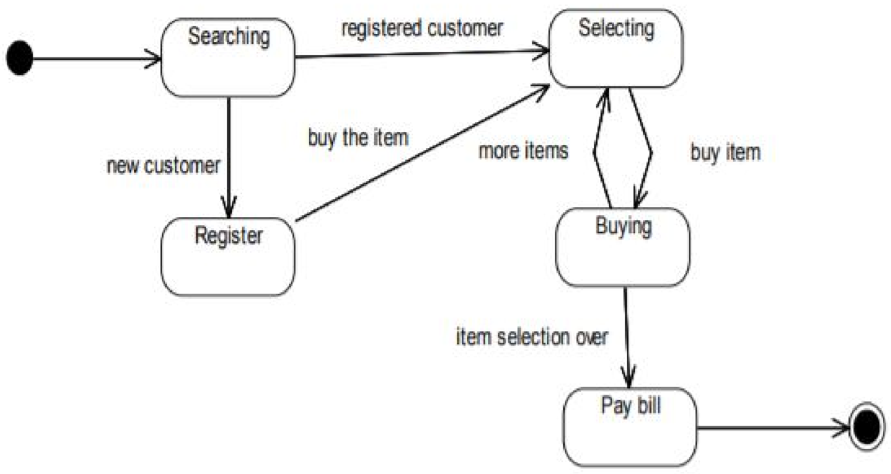


Activity diagram for Admin side





State Chart diagram for Online Shopping System



State diagram for customer.

9. To draw the implementation view diagram: Component and Deployment diagrams.

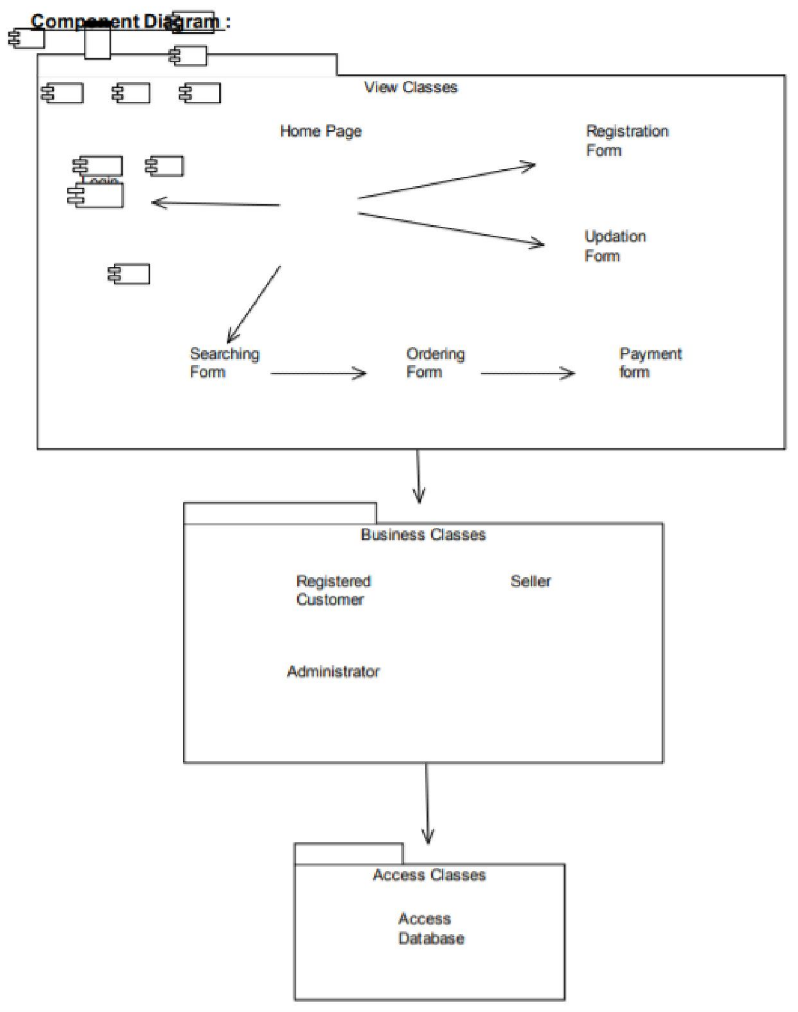
Ans :

Component Diagram

Component diagrams are different in terms of nature and behavior. Component diagrams are used to model physical aspects of a system. Physical aspects are the elements like executables, libraries, files, documents etc that resides in a node. So component diagrams are used to visualize the organization and relationships among components in a system. These diagrams are also used to make executable systems.

Description of the component diagram for online shopping system is as follows :

- Data base server contains all the database tables. It contains Administrator, Registered Customer, Seller, Item and Payment.
- Application server contains Access classes package and Business classes package and view layer classes, i.e. view classes package.
- Clients are the nodes having no processing capabilities. Only browser is there on this node to send a request.

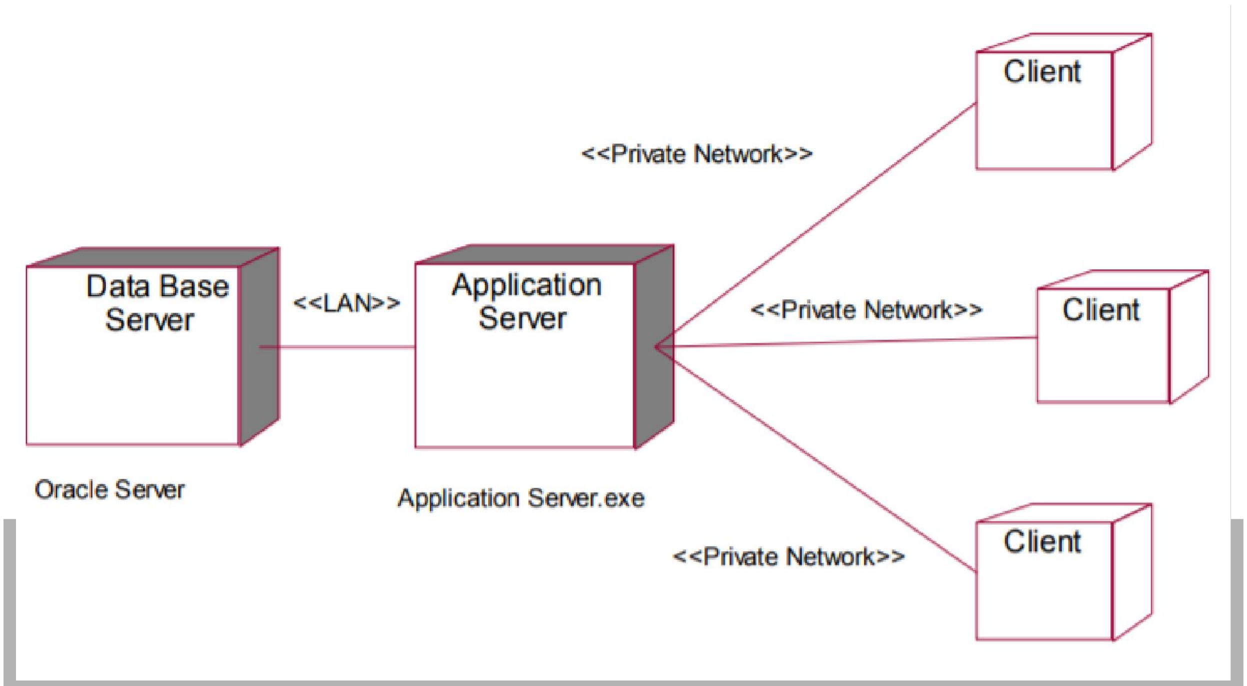


Deployment Diagram

Deployment diagrams are used to visualize the topology of the physical components of a system where the software components are deployed.

So deployment diagrams are used to describe the static deployment view of a system.

Deployment diagrams consist of nodes and their relationships.



10. To perform various testing using the testing tool unit testing, integration testing

Ans :

eCommerce testing is defined as testing of an eCommerce (online shopping) application. It helps in the prevention of errors and adds value to the product by ensuring conformity to client requirements.

The objective of testing is to ensure

- ➢ Software reliability
- ➢ Software quality
- ➢ System Assurance
- ➢ Optimum performance and capacity utilization

Setting up an E-commerce system is a complex process and subject to many market-specific variables. To maintain the integrity of the E Commerce system, testing becomes compulsory

Types of Testing for E-commerce System

A common type of testing included into e commerce system is

S.No.	Type of Testing	Testing Process
1	Browser compatibility	Lack of support for early browsers Browser specific extensions Browser testing should cover the main platforms
2	Page display	Incorrect display of pages Runtime error messages Poor page download time Dead hyperlink, plugin dependency, font sizing, etc.
3	Session Management	Session Expiration Session storage
4	Usability	Non-intuitive design Poor site navigation Catalog navigation Lack of help-support
5	Content Analysis	Misleading, offensive and litigious content Royalty free images and copyright infringement Personalization functionality Availability 24/7
6	Availability	Denial of service attacks Unacceptable levels of unavailability
7	Back-up and Recovery	Failure or fall over recovery Backup failure Fault tolerance
8	Transactions	Transaction Integrity Throughput Auditing
9	Shopping order processing and purchasing	Shopping cart functionality Order processing Payment processing Order tracking
10	Internationalization	Language support Language display Cultural sensitivity Regional Accounting
11	Operational business procedures	How well e-procedure copes Observe for bottlenecks
12	System Integration	Data Interface format Interface frequency and activation Updates Interface volume capacity Integrated performance
13	Performance	Performance bottlenecks Load handling Scalability analysis
14	Login and Security	Login capability Penetration and access control Insecure information transmission Web attacks Computer viruses Digital signatures

The following are a few things to test

- Is it going to auto scroll?
- If yes, at what interval will the image be refreshed?
- When the user hovers over it, is it still going to scroll to the next one?
- Can it be hovered on?
- Can it be clicked on?
- If yes, is it taking you to the right page and right deal?
- Is it loading along with the rest of the page or loads last in comparison to the other elements on the page?
- Can the rest of the content be viewed?
- Does it render the same way in different browsers and different screen resolutions?

Product Details Page

Once a user finds a product either through search or by browsing or by clicking on it from the homepage, the user will be taken to the product information page.

Check

- Image or images of the product
- Price of the product
- Product specifications
- Reviews
- Check out options
- Delivery options
- Shipping information
- In-stock/Out of stock
- Multiple color or variations options
- Breadcrumb navigation for the categories (highlighted in Red below). If navigation such as that is displayed, make sure every element of it is functional.

5. Banking System

1. Phases in software development project, overview, need, coverage of topics

Ans :

A computer based management system is designed to handle all the primary information required to calculate monthly statements of customer account which include monthly statement of any month. Separate database is maintained to handle all the details required for the correct statement calculation and generation.

This project intends to introduce more user friendliness in the various activities such as record updation, maintenance, and searching. The searching of record has been made quite simple as all the details of the customer can be obtained by simply keying in the identification or account number of that customer. Similarly, record maintenance and updation can also be accomplished by using the account number with all the details being automatically generated. These details are also being promptly automatically updated in the master file thus keeping the record absolutely up-to-date.

The main objective of our project is providing the different typed of customers facility, the main objective of this system is to find out the actual customer service. Etc.

- It should fulfill almost all the process requirements of any Bank.
- It should increase the productivity of bank by utilizing the working hours more and more, with minimum manpower.

This project includes the entireupgraded feature required for the computerization banking system. This system is very easy to use, so that any user can use without getting pre-knowledge about this. Its very much user friendly and meet almost all daily working process requirements. This system is completely GUI based and can be use by mouse and as well as keyboard. This system is melded in such a way that has got all features to upgrade without making much change in existing components.

This banking process consists of five divisions. There are customer details, creating a new account, withdrawing money, loan details and depositing money. The customer details consist of customer name, address, phone number, account number.

To create a new account verifies the rules. Enter the account andthen get an account number from a database. To withdraw money checks the balance in our account and then get the money. The loan details consist of loan types like home loans, car loans, education loans etc.To deposit money enter the account number and give the account to be deposited.

Some customers avoid online banking as they perceive it as being too vulnerable to fraud. The security measures employed by most banks are never 100% safe, but in practice the number of fraud victims due to online banking is very small.

The Existing System

The system will check the user's existence in the database and provide the set of services with respect to the role of the user. The application is based on three-tier architecture. The cipher key obtained will help to find the fraud application. The business logic helps in authenticating the application, authorizing the users and providing services. The technologies are chosen by keeping the compatibility and performance as the constraints for the application.

Further Drawbacks of the Existing System

The following are the drawbacks of the existing manual System.

- **Time Delay:** In the existing system, information related to all transactions is stored in different registers. Since all the transactions are stored in different registers it takes lot of time to prepare different reports.
- **Redundancy:** As the information passes through different registers, each register is consolidated and sent to next register. So the same information is being tabulated at each register, which involves lot of complication and duplication in work, thus it causes redundancy.
- **Accuracy:** Since the same data is compiled at different sections, the possibility of tabulating data wrongly increases. Also if the data is more, validations become difficult. This may result in loss of accuracy of data.
- **Information Retrieval:** As the information is stored in the particular Format, it can only be retrieved in the same format. But if it is to be retrieve in different format, it is not possible.
- **Storage Media:** In the existing system, data transaction being stored on too long registers it is very difficult to refer after some time.
- **Reports:** At the various reports are tabulated manually. They are not such Attractive and require more time. They do not provide adequate help in maintaining the accounts.
- **Enquiry:** Enquiry for different level of information is much more difficult. On
- Line enquiry of data is not possible.

Proposed System

System analysis will be performed to determine if it is feasible to design information based on policies and plans of the organization and on user requirements and to eliminate the weaknesses of the present system.

General requirements are :

1. The new system should be cost effective.
2. To augment management, improve productivity and services.
3. To enhance User/System interface.
4. To improve information qualify and usability.
5. To upgrade system's reliability, availability, flexibility and growth potential.

Developers Responsibilities Overview

The developer is responsible for :

1. Developing the system, which meets the SRS and solving all the requirements of the system?
2. Demonstrating the system and installing the system at client's location after the acceptance testing is successful.
3. Submitting the required user manual describing the system interfaces to work on it and also the documents of the system.
4. Conducting any user training that might be needed for using the system.
5. Maintaining the system for a period of one year after installation

2. To assign the requirement engineering tasks

Ans :

Functional Requirements

- **Inputs:** The major inputs for "Online Banking"can be categorized module wise. Basically all the information is managed by the software and in order to access the information one has to produce one's identity by entering the user-id and password. Every user has theirown domain of access beyond which the access is dynamically refrained rather denied.
- **Output:** The major outputs of the system are tables and reports. Tables are created dynamically to meet the requirements on demand. Reports, as it is obvious, carry the gist of the whole information that flows across the institution.

  This application must be able to produce output at different modules for different inputs.

Performance Requirements

Performance is measured in terms of reports generated weekly and monthly.

Software and Hardware Specifications

Hardware

Processor	:	Intel Pentium III or Above
Ram	:	256 MB or more
Cache	:	512 KB
Hard disk	:	16 GB hard disk recommended for primary partition.

Software

Operating system	:	Windows XP or later
Front End Software	:	ASP.NET (C# .NET)
Back End Software	:	SQL Server 2005

Feasibility Study

Techinical Feasibility

Evaluating the technical feasibility is the trickiest part of a feasibility study. This is because, at this point in time, not too many-detailed design of the system, making it difficult to access issues like performance, costs on (on account of the kind of technology to be deployed) etc.A number of issues have to be considered while doing a technical analysis.

Understand the different technologies involved in the proposed system :

- Before commencing the project, we have to be very clear about what are the
- Technologies that are to be required for the development of the new system.

Find out whether the organization currently possesses the required technologies :

- Is the required technology available with the organization?
- If so is the capacity sufficient?

Operational Feasibility

Proposed projects are beneficial only if they can be turned into information systems that will meet the organizations operating requirements. Simply stated, this test of feasibility asks if the system will work when it is developed and installed. Are there major barriers to Implementation? Here are questions that will help test the operational feasibility of a project :

➢ Is there sufficient support for the project from management from users?

➢ If the current system is well liked and used to the extent that persons will not be able to see reasons for change, there may be resistance.

➢ Are the current business methods acceptable to the user? If they are not,

➢ Users may welcome a change that will bring about a more operational and useful systems.

➢ Have the user been involved in the planning and development of the project?

➢ Early involvement reduces the chances of resistance to the system and in General and increases the likelihood of successful project.

Economic Feasibility

Economic feasibility attempts 2 weigh the costs of developing and implementing a new system, against the benefits that would accrue from having the new system in place. This feasibility study gives the top management the economic justification for the new system.

A simple economic analysis which gives the actual comparison of costs and benefits are much more meaningful in this case. In addition, this proves to be a useful point of reference to compare actual costs as the project progresses. There could be various types of intangible benefits on account of automation. These could include increased customer satisfaction, improvement in product quality better decision making timeliness of information, expediting activities, improved accuracy of operations, better documentation and record keeping, faster retrieval of information, better employee morale.

3. To perform the system analysis: Requirement analysis, SRS

Ans :

Requirement Specification

Product Perspective

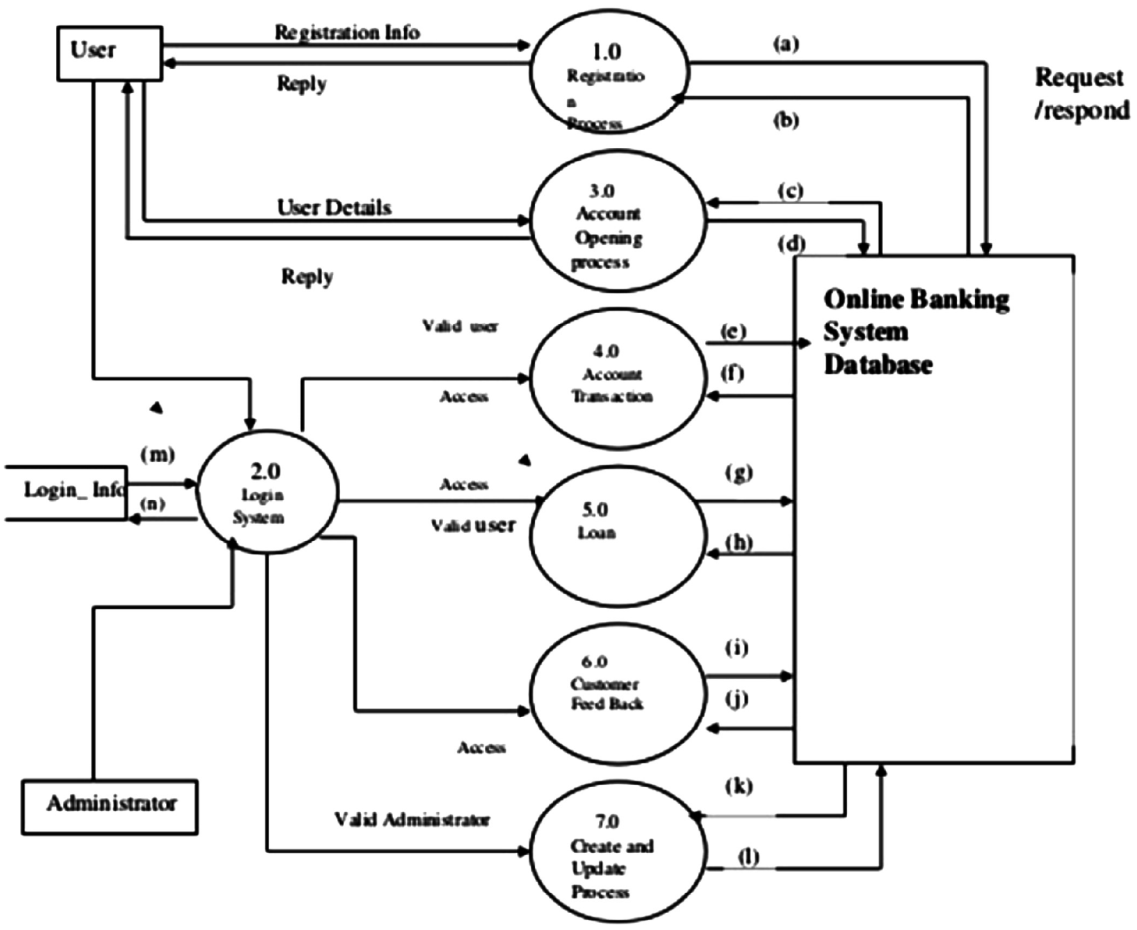
The Online Banking System is a package to be used by Bank customer to improve the efficiency of customer, Bank employees and Bank. The Online Banking System to be developed benefits greatly the members and the customer of Bank of baroda. The system provides money transfer catalog and Account details to members and helps them on the Personal Account management. The real time system can keep the account record catalog updated all the time so that the members get the updated information all the time.

The complete overview of the system is as shown in the overview diagram below:

The product to be developed has interactions with the users: bank Members who are the customer and worker in the bank.

The product has to interact with other systems like: Internet, Billing System and the Bank database System.

Over all working of Internet Banking



Product Functions

The Online banking System provides online real time information about the funds available in the bank and the user information. The Product functions are more or less the same as described in the product perspective. The functions of the system include the system providing different type of services based on the type of users.

- The member should be provided with the updated information about the Accounts.
- Provisions for the members to borrow the books they want, if all the other required rules hold good.
- The member is given a provision to check his account information and change the account information any time in the given valid period.
- The members are provided with the books available roster and allowed to choose the books, which they want to use in the coming up days.
- The Banker can get the information about the members who have performed fund transfer.

User characteristics

The users of the system are bank members, banker and the administrators who maintain the system. The members and the banker are assumed to have basic knowledge of the computers and Internet browsing. The administrators of the system to have more knowledge of the internals of the system and is able to rectify the small problems that may arise due to disk crashes, power failures and other catastrophes to maintain the system. The proper user interface, users manual, online help and the guide to install and maintain the system must be sufficient to educate the users on how to use the system without any problems.

Constraints

- The infor mation of all the users must be stored in a database that is accessible by the Online banking System.
- The banking information system must be compatible with the Internet applications.
- The Online banking System is connected to the server and is running all 24 hours a day.
- The users access the Online banking System from any computer that has Internet browsing capabilities and an Internet connection.
- The users must have their correct usernames and passwords to enter into the Online banking System.

Assumptions and dependencies

- The users have sufficient knowledge of computers.
- The computer should have Internet connection and Internet server capabilities.
- The users know the English language, as the user interface will be provided in English

Usability

- The system shall allow the users to access the system from the Internet using HTML or it's derivative technologies. The system uses a web browser as an interface.
- Since all users are familiar with the general usage of browsers, no specific training is required.
- The system is user friendly and self-explanatory.

Reliability

The system has to be very reliable due to the importance of data and the damages incorrect or incomplete data can do.

Availability

The system is available 100% for the user and is used 24 hrs a day and 365 days a year. The system shall be operational 24 hours a day and 7 days a week.

- **Mean Time Between Failures (MTBF) -** The system will be developed in such a way that it may fail once in a year.
- **Mean Time to Repair (MTTR) -** Even if the system fails, the system will be recovered back up within an hour or less.

Accuracy

The accuracy of the system is limited by the accuracy of the speed at which the employees of the bank and users of the bank use the system.

- **Maximum Bugs or Defect Rate -** Not specified.
- **Access Reliability -** The system shall provide 100% access reliability.

- **Information Security Requirement -** The system shall support the UHCL information security requirements and use the same standard as the UHCL information security requirements.
- **Billing System Data Compatibility -** The member balance amount that will be calculated and sent to the billing system shall be compatible with the data types and design constraints of the billing system.

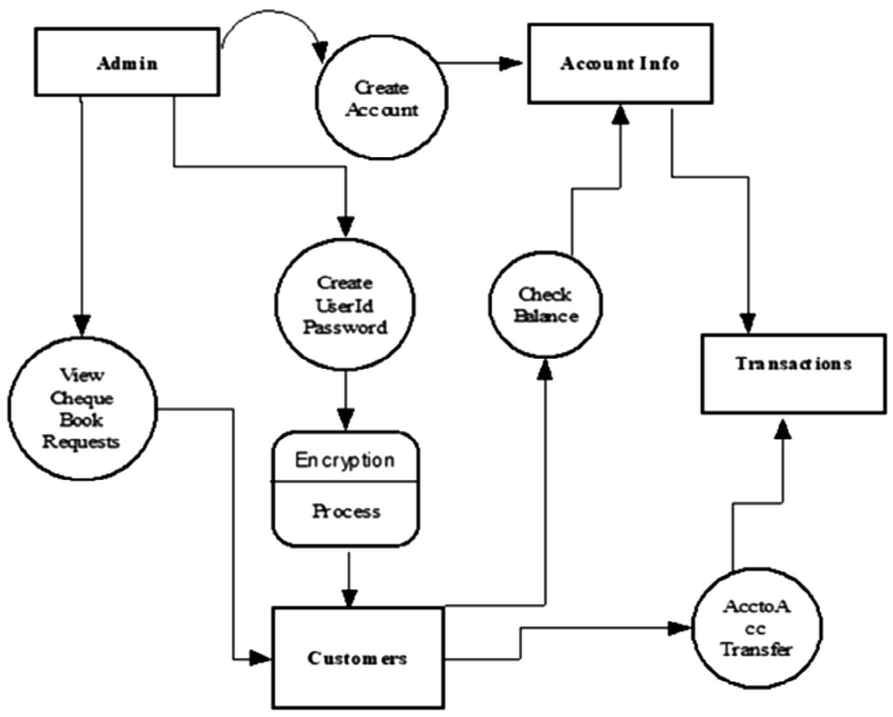
Maintenance

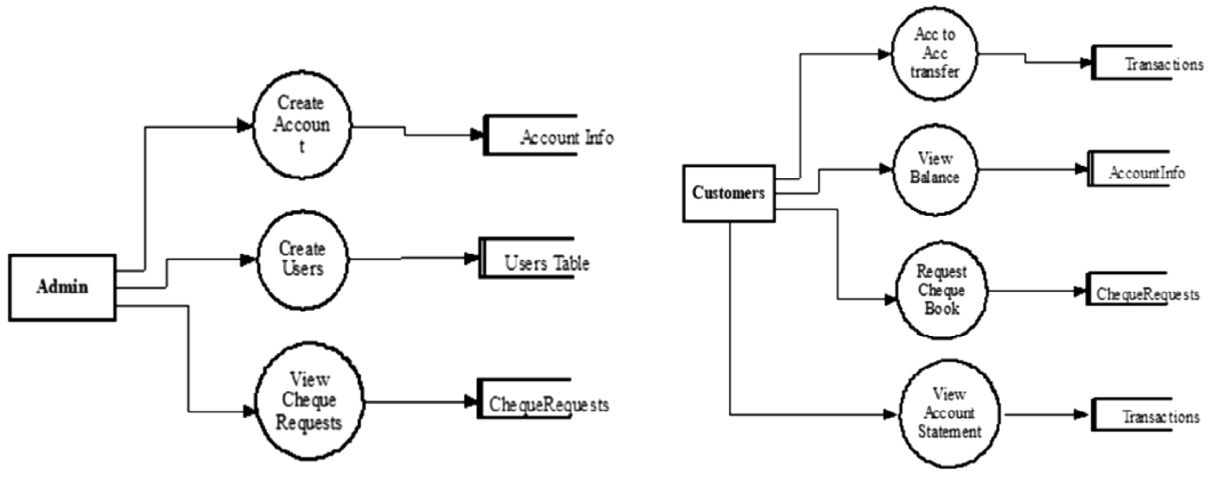
The maintenance of the system shall be done as per the maintenance contract.

- **Standards -** The coding standards and naming conventions will be as per the American standards.

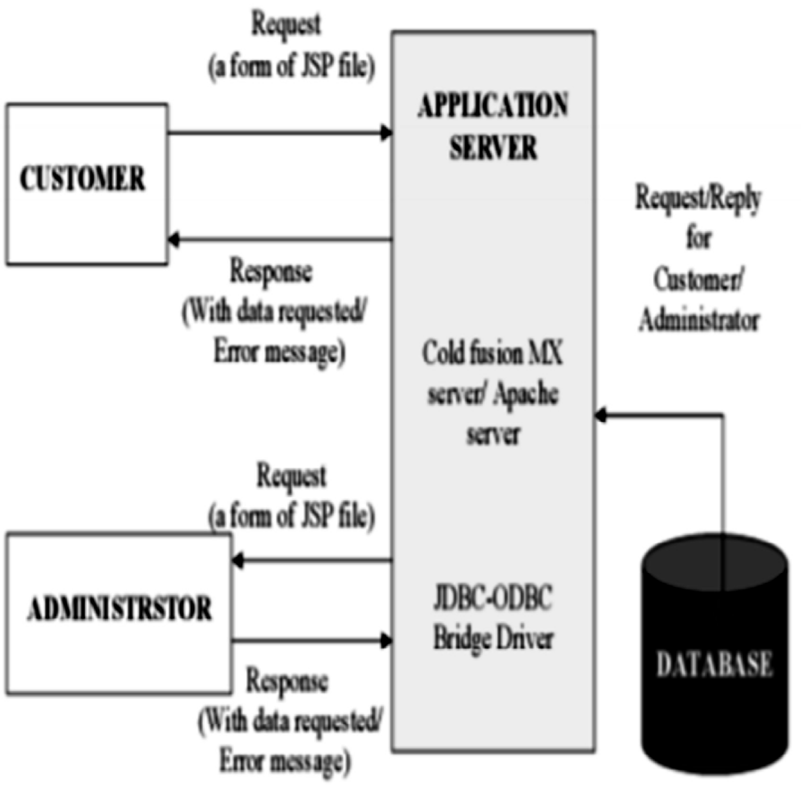
4. To perform the function-oriented diagram: DFD and Structured chart.

Ans :



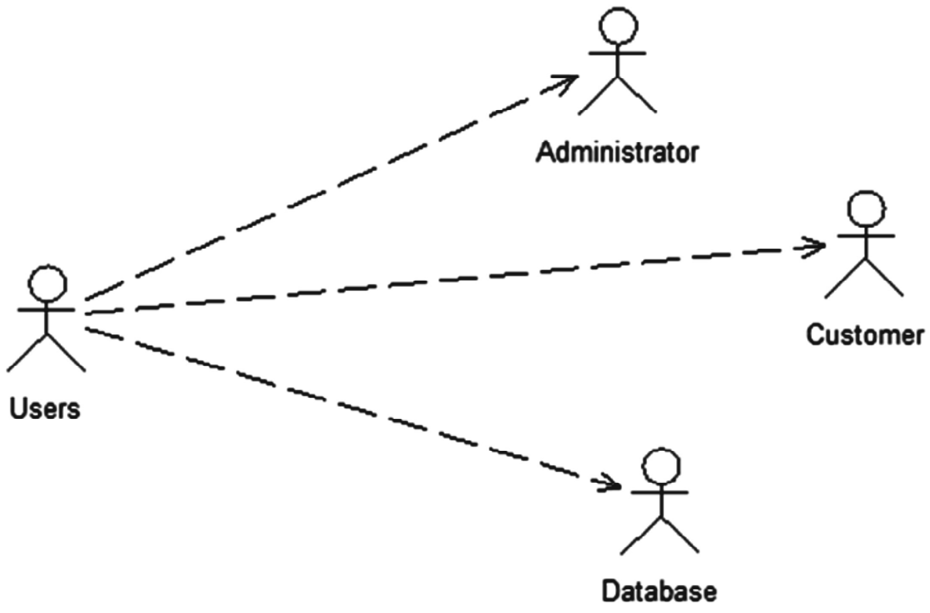


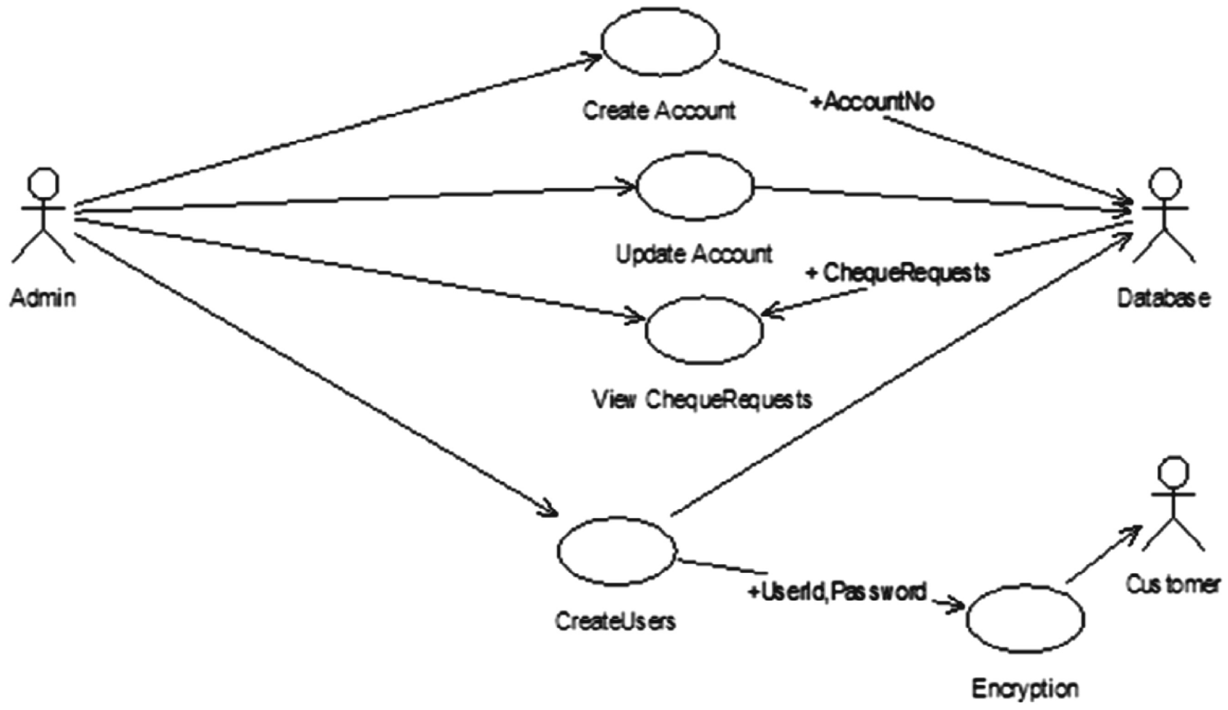
Architecture of On-Line Banking System

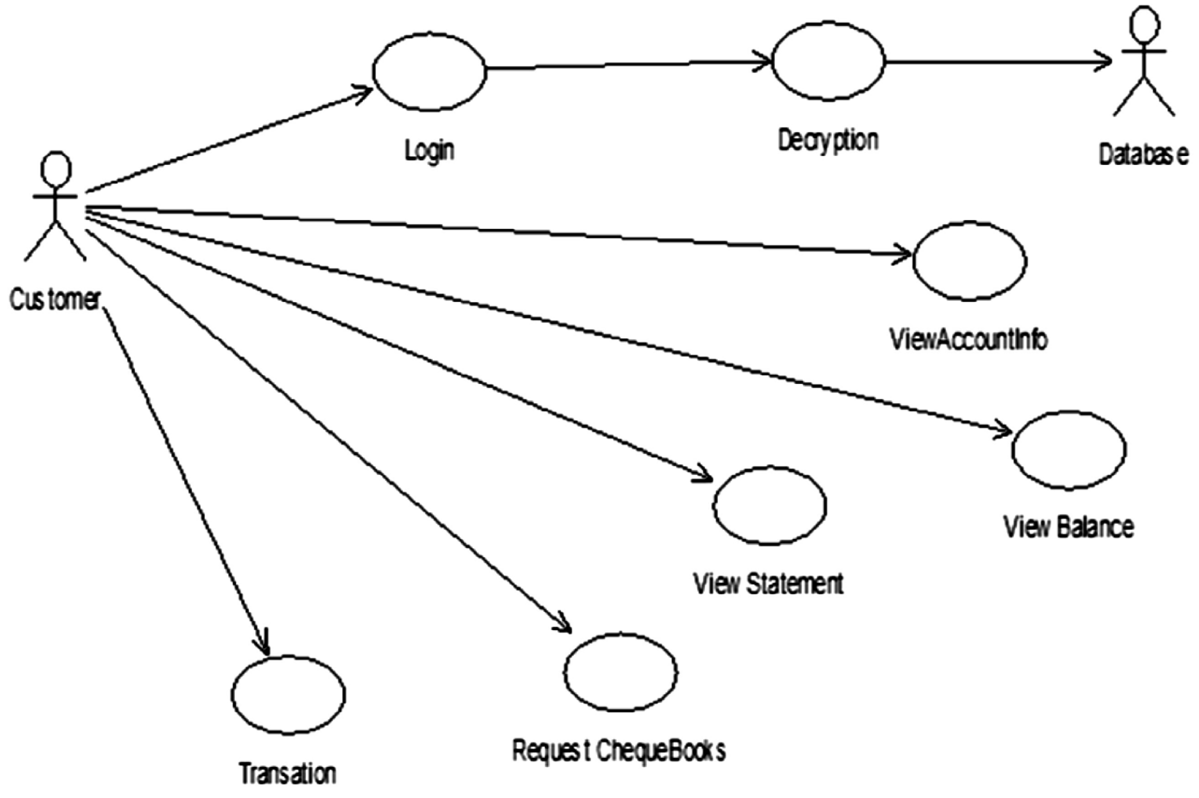


5. To perform the user's view analysis: Use case diagram

Ans :



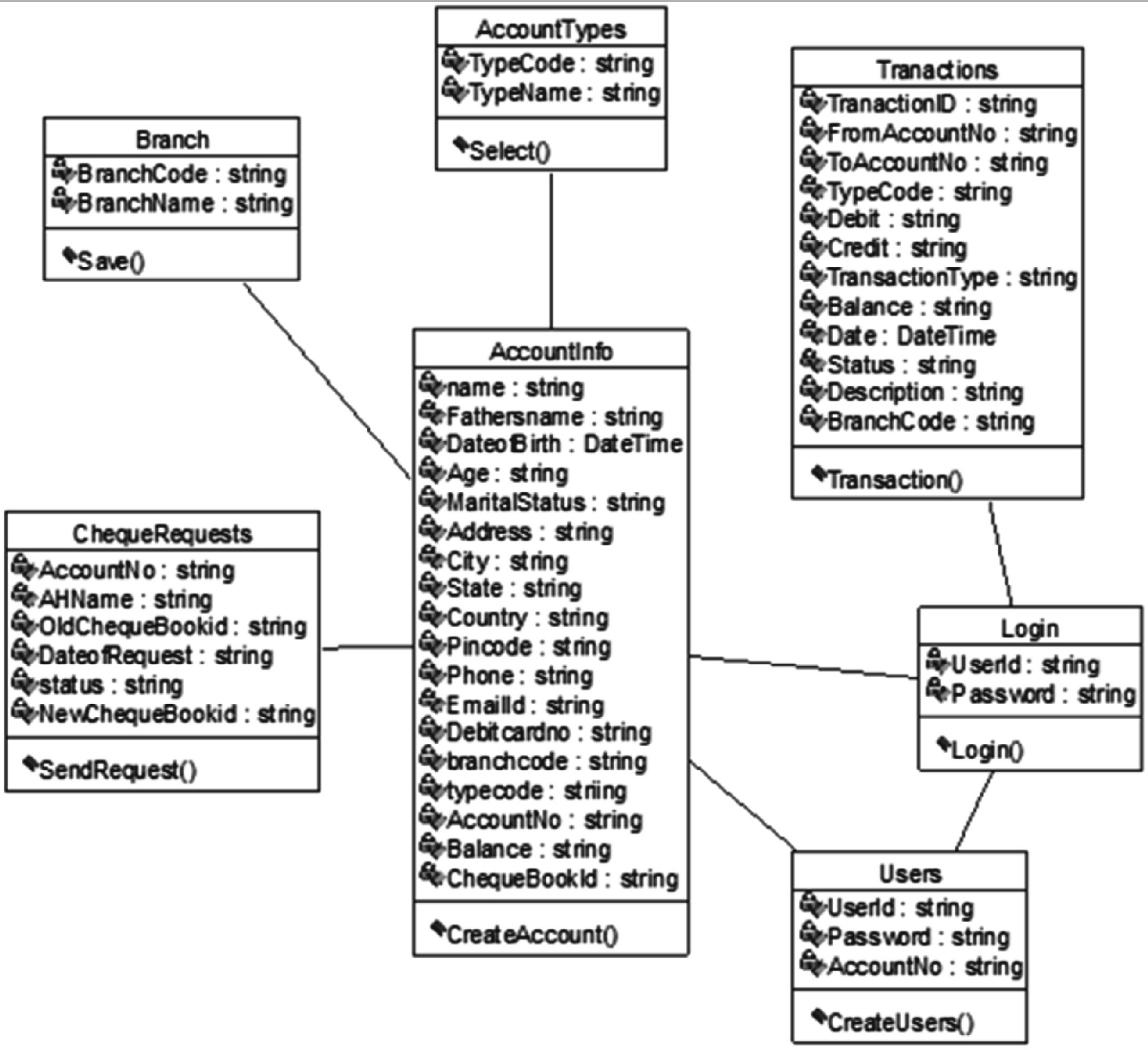

Create Account
+AccountNo
Admin
Update Account
Database
+ ChequeRequests
View ChequeRequests
Customer
CreateUsers
+UserId,Password
Encryption

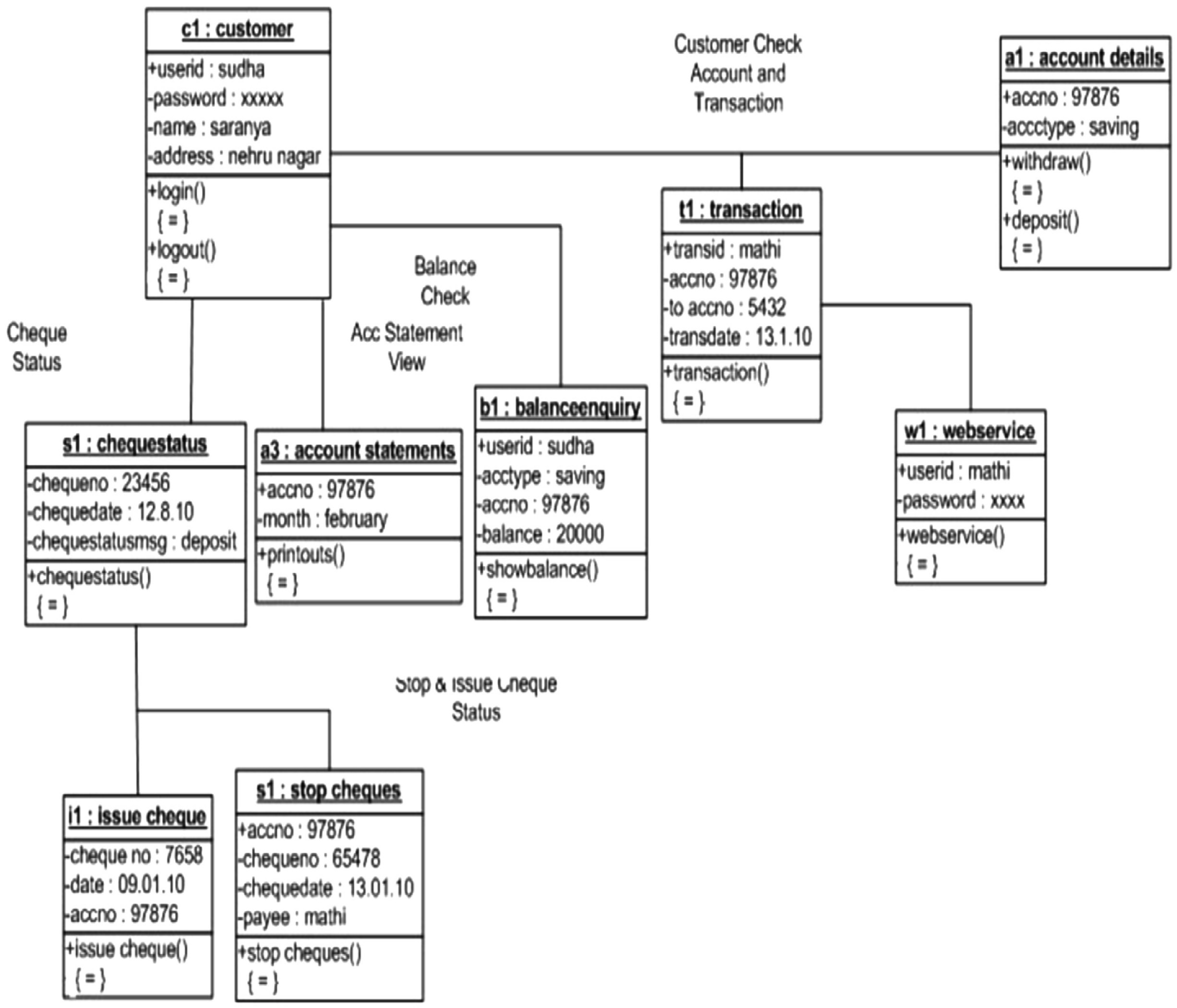

Login
Decryption
Database
Customer
ViewAccountInfo
View Balance
View Statement
Transation
Request ChequeBooks

6. To draw the structural view diagram: Class diagram, object diagram

Ans :

Static models of a system describe the structural relationships that hold between the Pieces of data manipulated by the system. They describe how data is parcelled out into Objects, how those objects are categorized, and what relationships can hold between them. They do not describe the behavior of the system, nor how the data in a system evolves over time. These aspects are described by various types of dynamic model. The most important kinds of static model are object diagrams and class diagrams. An object diagram provides a 'snapshot' of a system, showing the objects that actually exist at a given moment and the links between them. Many different object diagrams can be drawn for a system, each representing the state of the system at a given instant. An object diagram shows the data that is held by a system at a given moment. This data may be represented as individual objects, as attribute values stored inside these objects, or as link between objects



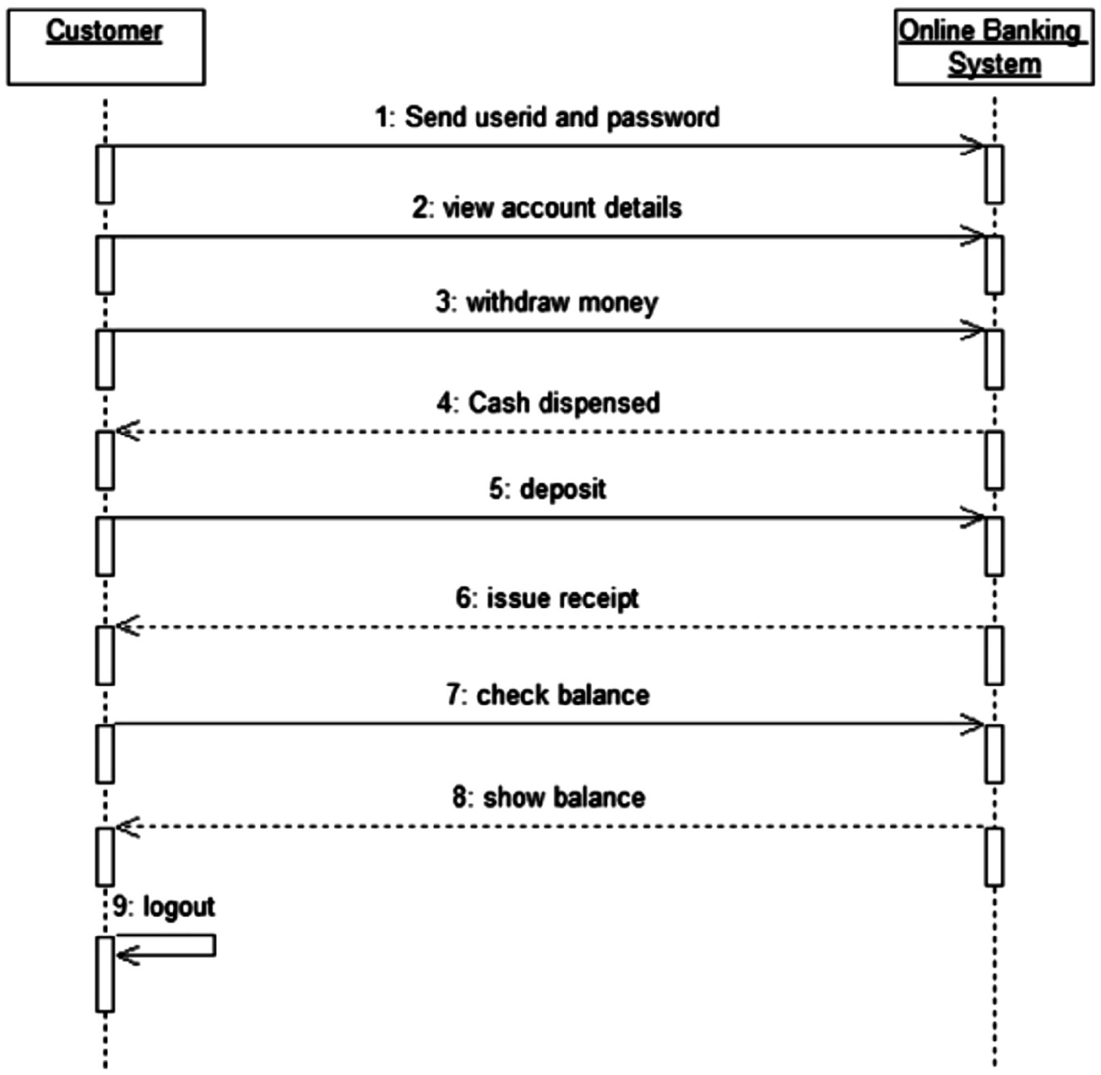


7. To draw the behavioural view diagram: Sequence diagram, Collaboration diagram

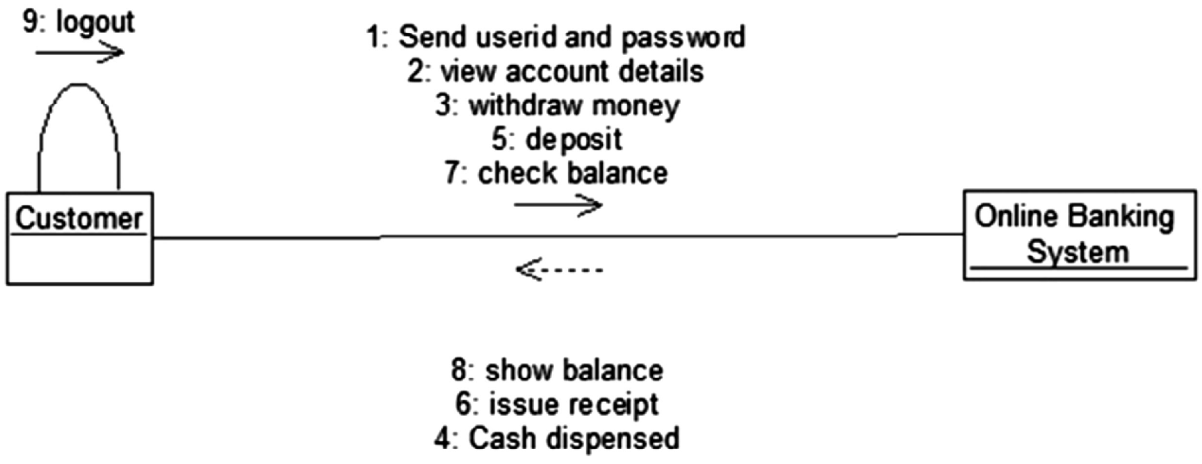
Ans :

Sequence Diagram

Because of the speed, flexibility, and efficiency that it offers, the Internet has become the means for conducting growing numbers of transactions between suppliers and large international corporations. In this way, the Internet has opened new markets to the world and has accelerated the diffusion of knowledge. The meaning of Internet markets or online business has been widely used in these days. The success of the business depends on its flexibility, availability and security. Since that the web-based systems should have a special way to design the system and implement it. Now a days, the Internet Banking System widely used and the banks looking to provide the best quality system with highly available, fast response, secure and safe to use.



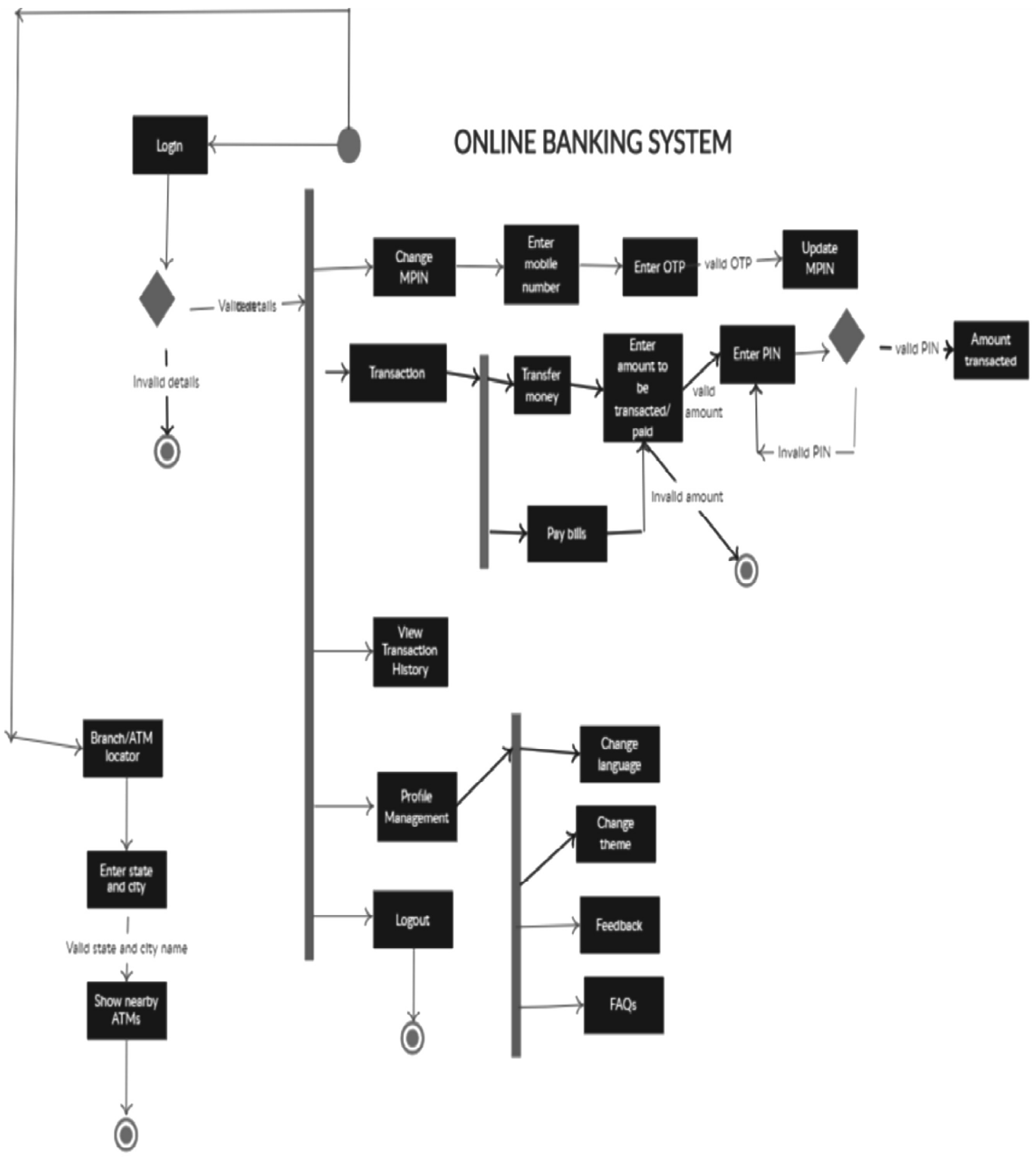
Collaboration Diagram



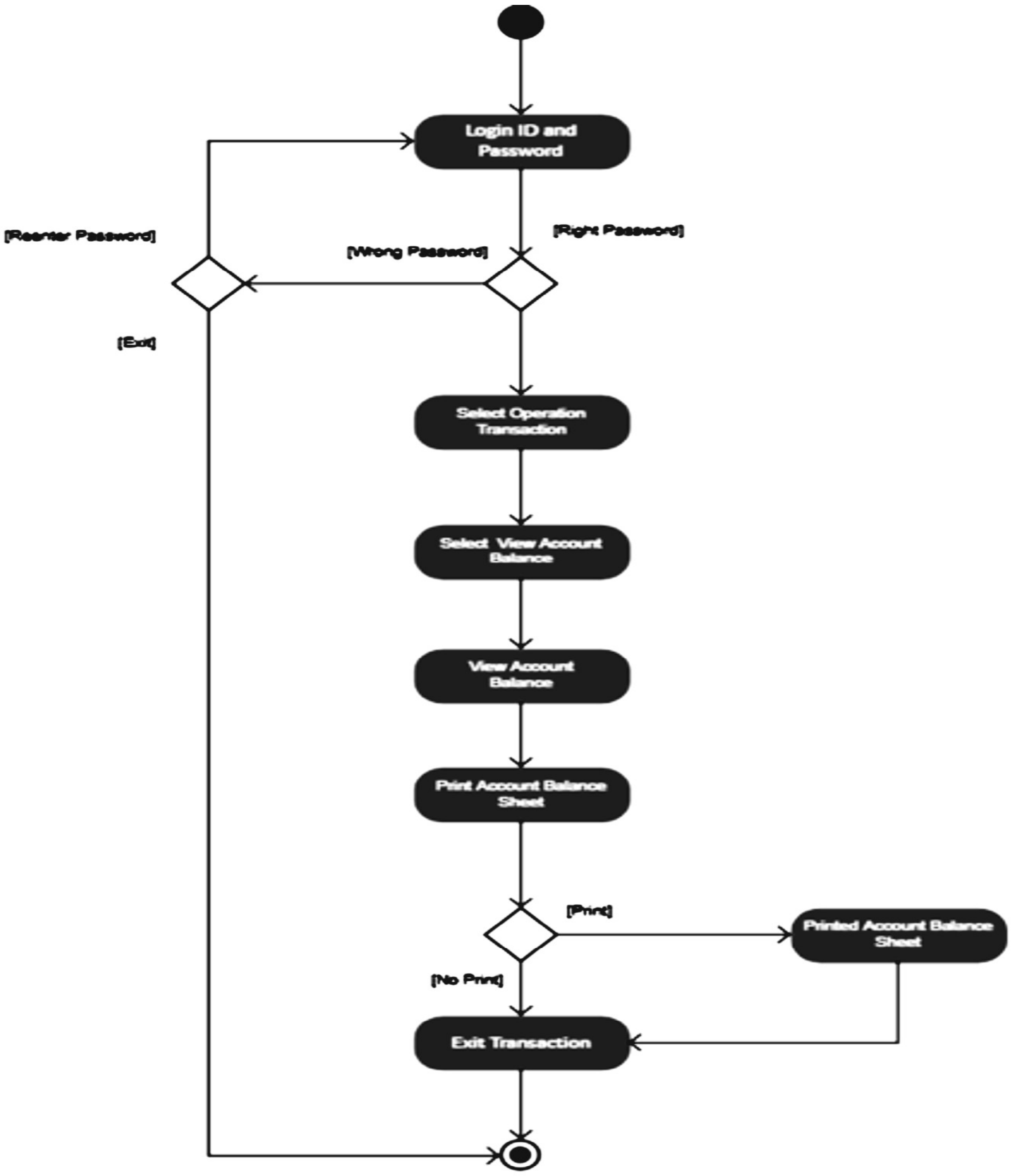
8. To draw the behavioural view diagram: State-chart diagram, Activity diagram

Ans :

State-Chart Diagram



Activity Diagram



9. To draw the implementation view diagram: Component Diagram, Deployment Diagram

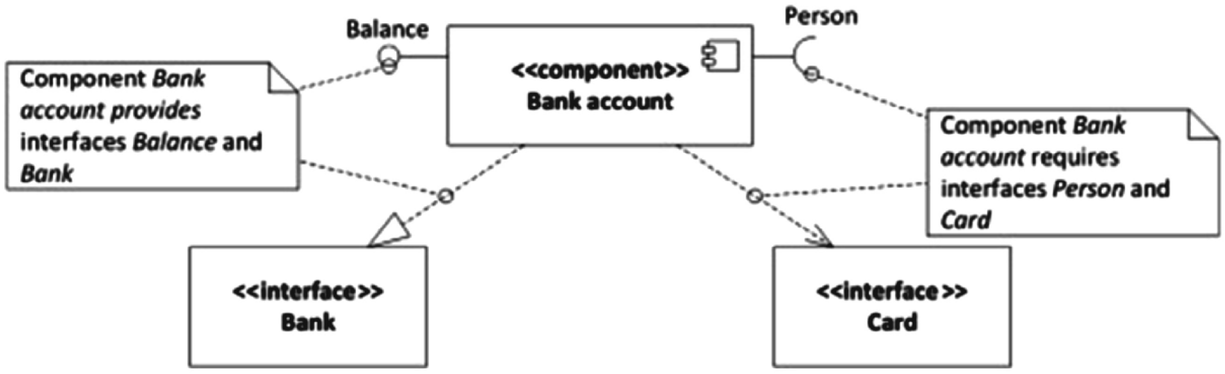
Ans :

Component Diagram

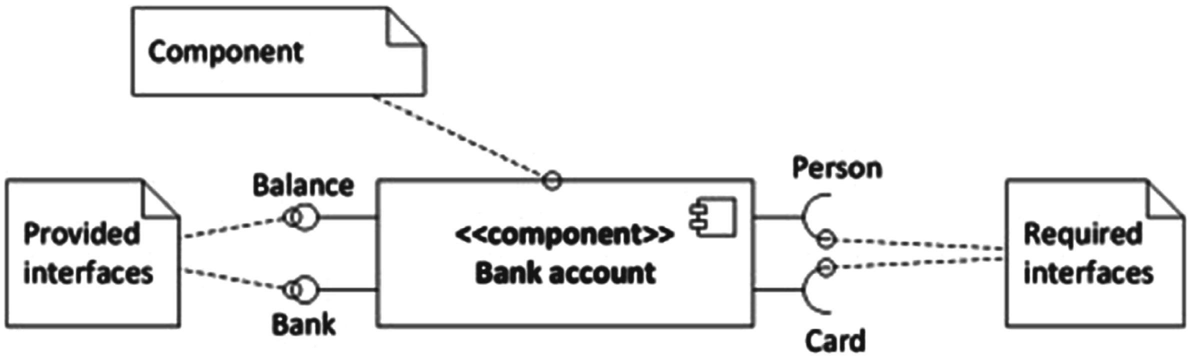
A component diagram shows the internal parts, connectors, and ports that implement a component. When the component is instantiated, copies of its internal parts are also instantiated. The UML component diagram shows how a software system will be composed of a set of deployable components—dynamic-link library (DLL) files, executable files, or web services—that interact through well-defined interfaces and which have their internal details hidden.

The component diagram contains the following elements:

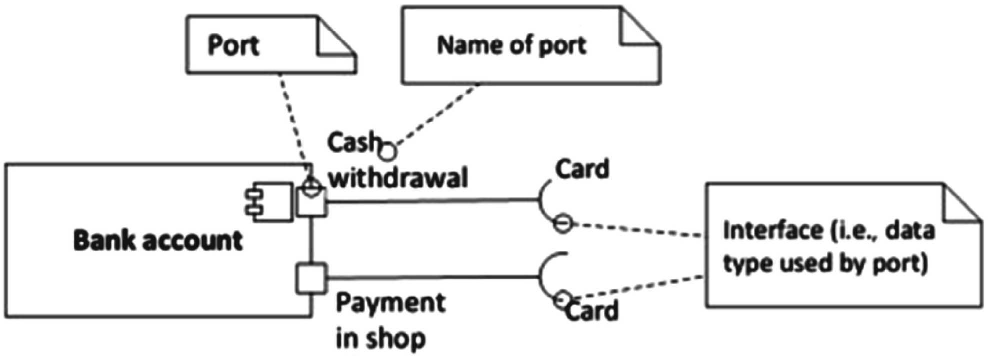
- **Interface** - specifies a contract consisting of a set of coherent public attributes and operations for a class. Any instance of a class that realizes the interface must fulfill that contract. Since interfaces are declarations, they are not instantiable. Instead, an interface specification is implemented by an instance of a class. Each class may implement more than one interface and each interface may be implemented by a number of different classes.


Balance
Person
Component *Bank account provides* interfaces *Balance* and *Bank*
<<component>>
Bank account
Component *Bank account* requires interfaces *Person* and *Card*
<<interface>>
Bank
<<interface >>
Card

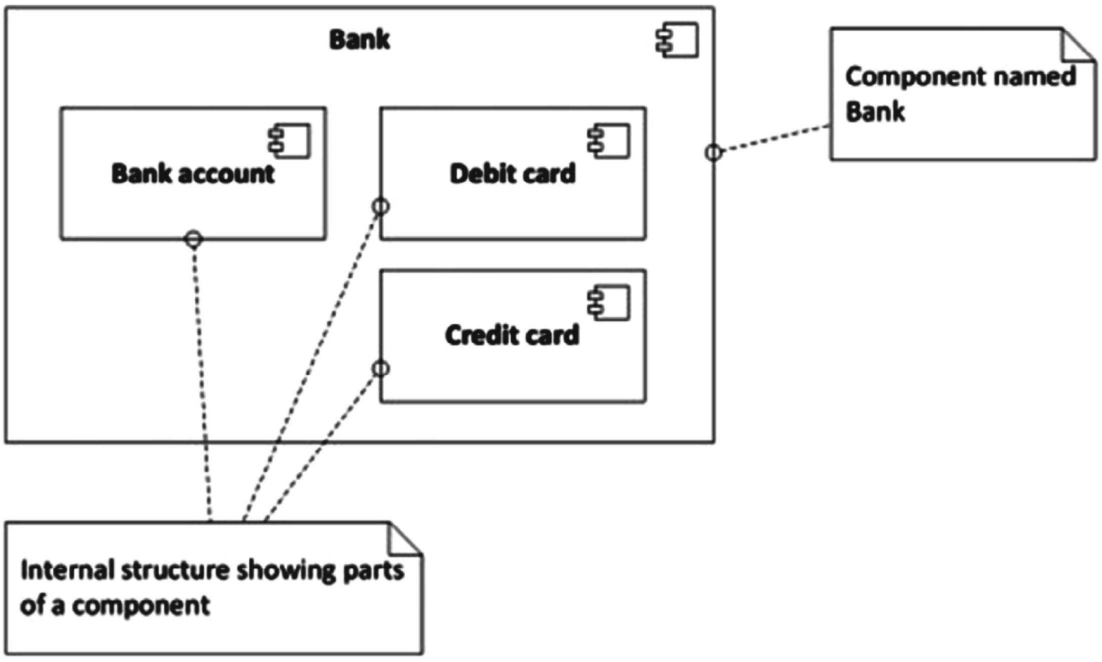
- **Component**—represents a modular part of a system that encapsulates its contents, it defines its behavior in terms of provided and required interfaces. As such, a component serves as a type whose conformance is defined by these provided and required interfaces (encompassing both their static as well as dynamic semantics). One component may therefore be substituted by another only if the two are type conformant.



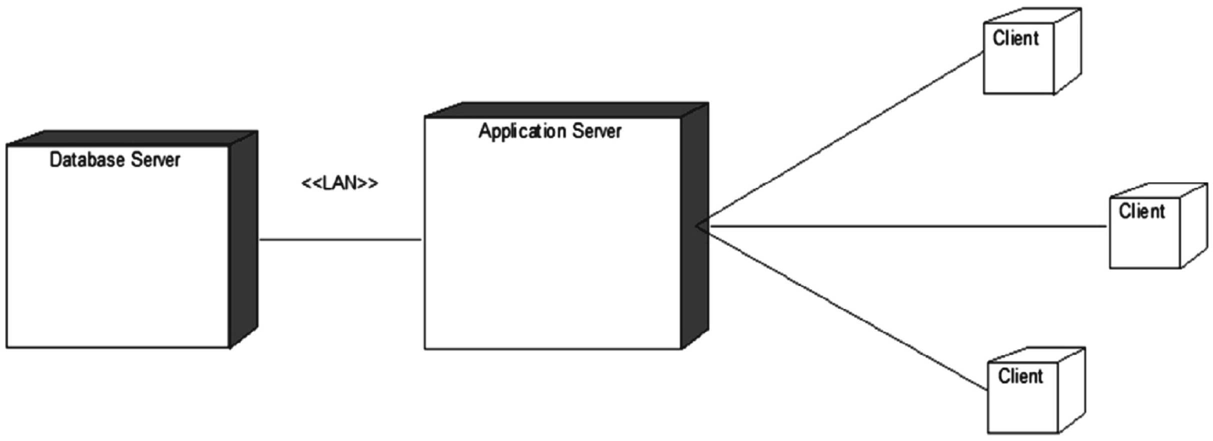
- **Port** - an explicit window into an encapsulated component. All of the interactions into and out of such component pass through ports. Each port provides or requires one or more specific interfaces. There can be multiple ports providing or requiring the same interface. It allows greater control over implementation and interaction with other components. Considers component with two named ports that each requires the same interface. The first port Cash withdrawal is used when bank's client takes out cash from automated teller machine (ATM) using his card. The other port named Payment in shop is used when making payments with card at shop.



- **Internal structure** - used to specify structure of a complex component, i.e., typically components are composed of smaller components thus building up the system.



Deployment Diagram



10. To perform various testing using the testing tool unit testing, integration testing

Ans :

The implementation phase of software development is concerned with translating design specification into source code. The preliminary goal of implementation is to write source code and internal documentation so that conformance of the code to its specifications can be easily verified, and so that debugging, testing and modifications are eased. This goal can be achieved by making the source code as clear and straightforword as possible. Simplicity, clarity and elegance are the hallmark of good programs, obscurity, cleverness, and complexity are indications of inadequate design and misdirected thinking.

Source code clarity is enhanced by structured coding techniques, by good coding style, by, appropriate supporting documents, by good internal comments, and by feature provided in modern programming languages.

The implementation team should be provided with a well-defined set of software requirement, an architectural design specification, and a detailed design description. Each team member must understand the objectives of implementation.

Terms in Testing Fundamental

1. **Error -** The term error is used in two ways. It refers to the difference between the actual output of software and the correct output, in this interpretation, error is essential a measure of the difference between actual and ideal. Error is also toused to refer to human action that result in software containing a defect or fault.
2. **Fault -** Fault is a condition that causes to fail in performing its required function. A fault is a basic reason for software malfunction and is synonymous with the commonly used term Bug.
3. **Failure -** Failure is the inability of a system or component to perform a required function according to its specifications. A software failure occurs if the behavior of the software is the different from the specified behavior. Failure may be caused due to functional or performance reasons.

   (a) Unit Testing - The term unit testing comprises the sets of tests performed by an individual programmer prior to integration of the unit into a larger system.

   A program unit is usually small enough that the programmer who developed it can test it in great detail, and certainly in greater detail than will be possible when the unit is integrated into an evolving software product. In the unit testing the programs are tested separately, independent of each other. Since the check is done at the program level, it is also called program teasing.

   (b) Module Testing - A module and encapsulates related component. So can be tested without other system module.

   (c) Subsystem Testing - Subsystem testing may be independently design and implemented common problems are sub-system interface mistake in this checking we concentrate on it.

There are four categories of tests that a programmer will typically perform on a program unit.

1. Functional test
2. Performance test
3. Stress test
4. Structure test

Functional Test

Functional test cases involve exercising the code with Nominal input values for which expected results are known; as well as boundary values (minimum values, maximum values and values on and just outside the functional boundaries) and special values.

Performance Test

Performance testing determines the amount of execution time spent in various parts of the unit, program throughput, response time, and device utilization by the program unit. A certain amount of avoid expending too much effort on fine-tuning of a program unit that contributes little to the over all performance of the entire system. Performance testing is most productive at the subsystem and system levels.

Stress Test

Stress test are those designed to intentionally break the unit. A great deal can be learned about the strengths and limitations of a program by examining the manner in which a program unit breaks.

Structure Test

Structure tests are concerned with exercising the internal logic of a program and traversing particular execution paths. Some authors refer collectively to functional performance and stress testing as “black box” testing. While structure testing is referred to as “white box” or “glass box” testing. The major activities in structural testing are deciding which path to exercise, deriving test date to exercise those paths, determining the test coverage criterion to be used, executing the test, and measuring the test coverage achieved when the test cases are exercised.

Challenges in Testing Internet Banking Applications

Different internet connections and browsers - Customers would login from different system and different browsers, like Mozilla, IE, Google chrome, Opera and the like. Also, user would be using different operating systems. Internet connections is yet another parameter. Software testing need to consider all these factors – browsers, Operating systems, internet connections and thorough testing need to be done. Page performance and all functionalities need to be tested thoroughly to ensure that customer can continue to operate the system smoothly.

Time to market - banks are constantly in a rush to update new features to entice customers. This might result in less time to test the application. This may seriously affect the quality of the internet banking application.

Different Types Of Testing In Internet Banking Applications

➢ **Usability testing** – Internet banking application would be used by many users- some would be technically sound and others may lack technical skills. Application should be simple so that even people who are not so much into technical side should be able to handle the application well. Website need to be tested for simple and efficient design so that any user would be able to navigate through internet banking application without assistance.

- **Security testing** – Banking applications are the key targets of hackers and groups that commit fraudulent activities. Vulnerability scanning and penetration testing can reveal proliferation of defects and further system susceptibilities.
- **Functional testing** – functional testing encompasses checking for all the requirements and specifications
- **Performance testing** – Some days may have spike in banking activities – especially in festivals or period during which there is an offer. Performance failures can affect the reputation of the financial institution badly.
- **Database testing** – This needs to be done to make sure that ensure that data integrity, data migration, validation and rules testing is fine.

Testing Of Online Banking Application :

1. In each release, business critical scenarios need to be tested in multiple cycles to make sure that functionalities are working as per the expectation.
2. Browser testing need to be done – for example Google chrome, Mozilla Firefox, Opera, IE and the like. Version testing also need to be done. Testing also need to be done in android and iOS devices to make sure that user interface is stable across all platforms.
3. Test cases need to be reviewed and modified after each release according to the new functionalities and changes.
4. If any defect is identified in production, that scenario need to be incorporated into the test scenario to make sure that issue will not occur in future releases.
5. Shakedown testing need to be performed after every build to make sure that environment is stable.
6. Latest versions of supporting tools and internal banking tools need to be tested that online banking application works fine across all platforms.
7. Once a defect is raised, it need to be captured and logs need to be attached. This would make it easier for the development team to analyze the root cause.
8. Major functionalities need to be tested after signoff till release to make sure that every functionality is working as per the expectation.
9. Clarification documents and emails need to be saved to make it useful for future releases.
10. Previous release learning or functionality need to be documented in a shared document.
11. Different types of test data needs to be saved in a shared document.
12. It would also be good if you have test data set up process in a shared location so that people in team can refer to and understand

Sample Test Cases For Net Banking Application :

1. Verify that user is able to login with valid username and password
2. Verify that user is able to perform basic financial transactions
3. Verify that user is able to add a beneficiary with valid name and account details
4. Verify that user is able to make financial transactions to added beneficiary
5. Verify that user is able to add decimal number into amount (limited by 2 numbers)
6. Verify that user is not able to add negative number into amount field.
7. Verify that user is allowed to transfer money only if there is proper account balance.
8. Verify that there is a confirmation check for financial transactions
9. Verify that user is given an acknowledgment receipt upon successful financial transaction.
10. Verify that customer is able to send money to multiple people
11. Verify that user is allowed to change password
12. Verify that account details reflect financial transactions also.
13. Verify that user with invalid password is not allowed to login.
14. Verify that after repeated attempts to login with incorrect password(as per the limits), user should be blocked.
15. Verify that time-out feature is implemented
16. Verify that if either of the username or password is blank, user is not allowed to login. User should be given an alert also.

Debugging

Defect testing is intended to find areas where the program does not confirm to its specifications. Tests are designed to reveal the presence of defect in the system.When defect have been found in the program. There must be discovered and removed. This is called “Debugging”.

FACULTY OF INFORMATICS

M.C.A II Year III - Semester Examination

MODEL PAPER - I

SOFTWARE ENGINEERING

Time : 3 Hours] [Max. Marks : 70

Answer all the question according to the internal choice **(5 × 14 = 70)**

			ANSWERS
1.	(a)	Define Software.	**(Unit-I, Q.No.1)**
	(b)	Define Schdule.	**(Unit-I, Q.No.2)**
		OR	
2.		Describe Waterfall Model.	**(Unit-I, Q.No.9)**
3.	(a)	Explain about Requirement Process.	**(Unit-II, Q.No.4)**
	(b)	Explain the concept of Software Architecture Document (SAD).	**(Unit-II, Q.No.19)**
		OR	
4.	(a)	Describe Various advantages of an SRS. What are various types of errors that occur in SRS?	**(Unit-II, Q.No.7)**
	(b)	Explain different approaches of Analysis in Software Engineering.	**(Unit-II, Q.No.13)**
5.		Explain about Effort and Schedule estimates of Software Projects.	**(Unit-III, Q.No.2)**
		OR	
6.		What are different methods used for monitoring a project?	**(Unit-III, Q.No.6)**
7.		What are the metrics of Coding?	**(Unit-IV, Q.No.6)**
		OR	
8.		Explain Black Box Testing in detail.	**(Unit-IV, Q.No.11)**
9.		What is the Capability Maturity Model?	**(Unit-V, Q.No.15)**
		OR	
10.		Define Reverse Engineering. State the uses of Reengineering.	**(Unit-V, Q.No.7)**

FACULTY OF INFORMATICS

M.C.A II Year III - Semester Examination

MODEL PAPER - II

SOFTWARE ENGINEERING

Time : 3 Hours] [Max. Marks : 70

Answer all the question according to the internal choice **(5 × 14 = 70)**

ANSWERS

1. What are various Software Engineering Problems? Explain. **(Unit-I, Q.No.3)**

OR

2. Explain in detail about Rational Unified Process Model and Time Box Model. **(Unit-I, Q.No.11)**

3. (a) What are Software Requirements Specification and Describe various characteristics of software requirements specification? **(Unit-II, Q.No.1)**

   (b) Explain the SRS with use case Modeling. **(Unit-II, Q.No.10)**

OR

4. (a) Explain the role of software architecture. **(Unit-II, Q.No.15)**

   (b) Explain the process of developing use cases. **(Unit-II, Q.No.12)**

5. Explain the concept of detailed scheduling. **(Unit-III, Q.No.7)**

OR

6. What are the different metrics used in software engineering in various stages? **(Unit-III, Q.No.13)**

7. Describe white box testing in software engineering. **(Unit-IV, Q.No.10)**

OR

8. Explain the methodology of Coding in Software Engineering. What are its Features? **(Unit-IV, Q.No.2)**

9. Explain the concept of Software Re-engineering. **(Unit-V, Q.No.6)**

OR

10. Describe in detail the frame work of PCCM. **(Unit-V, Q.No.16)**

FACULTY OF INFORMATICS

M.C.A II Year III - Semester Examination

MODEL PAPER - III

SOFTWARE ENGINEERING

Time : 3 Hours] [Max. Marks : 70

Answer all the question according to the internal choice (5 × 14 = 70)

ANSWERS

1. (a) Explain the Paradigm of Software Engineering and needs. **(Unit-I, Q.No.5)**

   (b) What are the differences between computer science and software engineering? **(Unit-I, Q.No.15)**

OR

2. Explain Prototype Model and Iterative Development Model. **(Unit-I, Q.No.10)**

3. (a) What is Software Requirements Specification? Explain Structure and Characteristics of SRS. **(Unit-II, Q.No.6)**

   (b) How would you build software architecture document? **(Unit-II, Q.No.18)**

OR

4. (a) List & Explain various components of an SRS. **(Unit-II, Q.No.8)**

   (b) What are the different views of Software Architecture? **(Unit-II, Q.No.16)**

5. Explain briefly about software design and different types of approaches. **(Unit-III, Q.No.8)**

OR

6. What are metrics, Measurements & Models of Project Management & Software Management? **(Unit-III, Q.No.12)**

7. Describe the Coding Guidelines in Software Engineering. **(Unit-IV, Q.No.3)**

OR

8. What are Software Testing Metric? **(Unit-IV, Q.No.13)**

9. What is Software Maintenance? Describe the types of Software Maintanance. **(Unit-V, Q.No.1)**

OR

10. What is Process Change Management? Explain in Detail. **(Unit-V, Q.No.14)**

FACULTY OF INFORMATICS

M.C.A. (2 years Course) III - Semester (CBCS) (Main) Examination

April/May - 2023

SOFTWARE ENGINEERING

Time : 3 Hours Max. Marks : 70

Note : I. Answer one question from each unit. All questions carry equal marks.

II. Missing data, if any, may be suitably assumed.

Answers

Unit-I

1. (a) Define software engineering. What are the major objectives of software engineering? **(Unit-I, Q.No. 4)**

Ans :

The main goal of Software Engineering is to develop software applications for improving quality, budget, and time efficiency. Software Engineering ensures that the software that has to be built should be consistent, correct, also on budget, on time, and within the required requirements.

(b) Explain cost schedule and quality of software. **(Unit-I, Q.No. 3)**

(OR)

2. (a) Write about waterfall model and its disadvantages. **(Unit-I, Q.No. 9)**

(b) Explain about project management process. **(Unit-I, Q.No. 14)**

Unit-II

3. (a) Give a use case scenario with an example. **(Unit-II, Q.No. 12)**

(b) Explain the general structure of an SRS. **(Unit-II, Q.No. 1)**

(OR)

4. (a) Write short note on shared data styles.

Ans :

In the shared-data style, the pattern of interaction is dominated by the exchange of persistent data. The data has multiple accessors and at least one shared-data store for retaining persistent data.

Database systems and knowledge-based systems are examples of this style. One feature of a shared-data style is the method by which the data consumer discovers that data of interest is available. If the shared-data store informs data consumers of the arrival of interesting data, the shared-data style is called a blackboard. If the consumer has responsibility for retrieving data, the shared-data style is called a repository. In modern systems, these distinctions have been blurred, as many database management systems that were originally repositories now provide a triggering mechanism that turns them into blackboards.

The shared-data style, summarized in Table is organized around one or more shared-data stores, which store data that other components may read and write. Component types include shared-data stores and data accessors. The general computational model associated with shared-data systems is that data accessors perform calculations that require data from the data store and writing results to one or more data stores. That data can be viewed and acted on by other data accessors. In a pure shared-data system, data accessors interact only through the shared-data store(s). However, many shared-data systems also allow direct interactions between nonstore elements. The data-store components of a shared-data system provide shared access to data, support data persistence, manage concurrent access to data, provide fault tolerance, support access control, and handle the distribution and caching of data values.

Summary of the shared-data style

Elements	• Component types: shared-data repositories and data accessors. • Connector types: data reading and writing.
Relations	Attachment relation determines which data accessors are connected to which data repositories.
Computational model	Communication between data accessors is mediated by a shared-data store. Control may be initiated by the data accessors or the data store.
Properties	Same as defined by the C&C viewtype and refined as follows: types of data stored, data performance-oriented properties, data distribution.
Topology	Data accessors are attached to connectors that are attached to the data store(s).

(b) Explain peer to peer and publish subscribe styles.

Ans :

The peer-to-peer style presents a view of the system that partitions the application by area of collaboration. Peers interact directly among themselves and can play the role of both clients and servers, assuming whatever role is needed for the task at hand. This partitioning provides flexibility for deploying the system across a distributed system platform. Because peers have access to the latest data, the load on any given component acting as a server is reduced, and the responsibilities that might have required more server capacity and infrastructure to support it are distributed. This can decrease the need for other communication for updating data and for central server storage but at the expense of storing the data locally.

Unit-III

5. (a) Explain project monitoring. **(Unit-III, Q.No. 6)**

   (b) Briefly describe about quality planning. **(Unit-III, Q.No. 4)**

(OR)

6. (a) Describe about function oriented design. **(Unit-III, Q.No. 10)**

   (b) Explain about detailed scheduling. **(Unit-III, Q.No. 7)**

Unit-IV

7. (a) Explain the testing objectives and its principles.

Ans :

Objectives

- Detecting bugs as soon as feasible in any situation.
- Avoiding errors in a project's and product's final versions.
- Inspect to see whether the customer requirements criterion has been satisfied.
- Last but not least, the primary purpose of testing is to gauge the project and product level of quality.

There are seven principles in software testing:

1. Testing shows the presence of defects

The goal of software testing is to make the software fail. Software testing reduces the presence of defects. Software testing talks about the presence of defects and doesn't talk about the absence of defects. Software testing can ensure that defects are present but it can not prove that software is defect-free. Even multiple testing can never ensure that software is 100% bug-free. Testing can reduce the number of defects but not remove all defects.

2. **Exhaustive testing is not possible**

   It is the process of testing the functionality of the software in all possible inputs (valid or invalid) and pre-conditions is known as exhaustive testing. Exhaustive testing is impossible means the software can never test at every test case. It can test only some test cases and assume that the software is correct and it will produce the correct output in every test case. If the software will test every test case then it will take more cost, effort, etc., which is impractical.

3. **Early Testing**

   To find the defect in the software, early test activity shall be started. The defect detected in the early phases of SDLC will be very less expensive. For better performance of software, software testing will start at the initial phase i.e. testing will perform at the requirement analysis phase.

4. **Defect clustering**

   In a project, a small number of modules can contain most of the defects. Pareto Principle to software testing state that 80% of software defect comes from 20% of modules.

5. **Pesticide paradox**

   Repeating the same test cases, again and again, will not find new bugs. So it is necessary to review the test cases and add or update test cases to find new bugs.

6. **Testing is context-dependent**

   The testing approach depends on the context of the software developed. Different types of software need to perform different types of testing. For example, The testing of the e-commerce site is different from the testing of the Android application.

7. **Absence of errors fallacy**

   If a built software is 99% bug-free but it does not follow the user requirement then it is unusable. It is not only necessary that software is 99% bug-free but it is also mandatory to fulfill all the customer requirements.

(b) Explain code inspection and summarize the report of an inspection. **(Unit-IV, Q.No. 8)**

(OR)

8. (a) What do you mean by system testing? Explain in detail.

Ans :

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. In system testing, integration testing passed components are taken as input.

The goal of integration testing is to detect any irregularity between the units that are integrated together. System testing detects defects within both the integrated units and the whole system. The result of system testing is the observed behavior of a component or a system when it is tested.

System Testing is carried out on the whole system in the context of either system requirement specifications or functional requirement specifications or in the context of both. System testing tests the design and behavior of the system and also the expectations of the customer.

(b) Write short note on white box testing? **(Unit-IV, Q.No. 10)**

Unit-V

9. (a) Justify the statement "Software maintenance is costlier". **(Unit-V, Q.No. 1)**

   (b) What is reengineering? **(Unit-V, Q.No. 2)**

10. Write short note on :

   (a) CMMI **(Unit-V, Q.No. 12)**

   (b) SPI trends **(Unit-V, Q.No. 11)**

FACULTY OF INFORMATICS

M.C.A. (2 years Course) III - Semester (CBCS) (Main) Examination

October / November - 2023

SOFTWARE ENGINEERING

Time : 3 Hours Max. Marks : 70

Note : I. Answer one question from each unit. All questions carry equal marks.

II. Missing data, if any, may be suitably assumed.

Unit-I

1. (a) Define software engineering. Explain the changing nature of software. **(Unit-I, Q.No. 4)**

(b) Explain cost, schedule and quality of software. **(Unit-I, Q.No. 3)**

(OR)

2. (a) Differentiate between process and project.

Ans :

S.No.	Nature	Process	Project
1.	**Objective**	A "process" has an objective that is typically defined around the ongoing operation of the process. For example, "provide ongoing maintenance for GM vehicles"	A "project" has an objective or outcome to be accomplished and the project ends when that objective is accomplished. That objective might be broadly-defined and might change or be further elaborated as the project is in progress. For example, "find a replacement ignition switch that will solve the problem with GM vehicles".
2.	**Time Duration**	A "process" is generally ongoing and doesn't normally have an end.	A "project" has a beginning and an end (although the beginning and end may not be well-defined when the project starts and the end might be a long time in the future).
3.	**Process Orientation**	A "process" is a repetitive sequence of tasks and the tasks are known at the outset since it is repetitive.	The sequence of tasks in a "project" is not normally repetitive and may not be known at the outset of the project.

(b) Explain prototyping model of software development model. **(Unit-I, Q.No. 10)**

Unit-II

3. (a) Explain the components of SRS. **(Unit-II, Q.No. 8)**

(b) Mention the values of good SRS. **(Unit-II, Q.No. 3)**

(OR)

4. (a) Explain component and connector view. **(Unit-II, Q.No. 16)**

(b) Explain the role of software architecture. **(Unit-II, Q.No. 15)**

Unit-III

5. (a) Discuss the concept of Risk assessment and Risk control. **(Unit-III, Q.No. 5)**

(b) Explain quality planning. **(Unit-III, Q.No. 4)**

(OR)

6. (a) Explain main object oriented concepts. **(Unit-III, Q.No. 10)**

   (b) Describe structured Design Methodology and function oriented design. **(Unit-III, Q.No. 8,10)**

Unit-IV

7. (a) Explain about unit testing. **(Unit-IV, Q.No. 6)**

   (b) Explain code inspection. **(Unit-IV, Q.No. 8)**

(OR)

8. (a) Explain different levels of testing. **(Unit-IV, Q.No. 9, 10)**

   (b) Write about black box testing? **(Unit-IV, Q.No. 11)**

Unit-V

9. (a) Explain the concept of software maintenance process. **(Unit-V, Q.No. 1)**

   (b) Write short note on software reengineering. **(Unit-V, Q.No. 2)**

(OR)

10. (a) Explain about SPI process. **(Unit-V, Q.No. 11)**

    (b) Write short note on PCMM. **(Unit-V, Q.No. 14)**