D.N.R.COLLEGE (AUTONOMOUS): BHIMAVARAM

M.Sc COMPUTER SCIENCE DEPARTMENT

Object Oriented Software Engineering

II M.Sc(Computer Science)&MCA

Presented by K.SUPARNA

UNIT- I

Introduction to Object Oriented Software Engineering: Nature of the Software, Types of Software, Software Engineering Activities, Software Quality

Introduction to Object Orientation: Data Abstraction, Inheritance & Polymorphism, Reusability in Software Engineering, Examples: Postal Codes, Geometric Points.

Requirements Engineering: Domain Analysis, Problem Definition and Scope, Types of Requirements, Techniques for Gathering and Analyzing Requirements, Requirement Documents, Reviewing Requirements,

Case Studies: GPS based Automobile Navigation System, Simple Chat Instant Messaging System.

UNIT- II

Unified Modeling Language & Use Case Modeling: Introduction to UML, Modeling Concepts, Types of UML Diagrams with Examples; User-Centred Design, Characteristics of Users, Developing Use Case Models Of Systems, Use Case Diagram, Use Case Descriptions, The Basics of User Interface Design, Usability Principles.

Class Design and Class Diagrams: Essentials of UML Class Diagrams, Associations And Multiplicity, Generalization, Instance Diagrams, Advanced Features of Class Diagrams,

Process of Developing Class Diagrams, Interaction and Behavioral Diagrams: Interaction Diagrams, State Diagrams, Activity Diagrams, Component and Deployment Diagrams.

UNIT- III

Software Design and Architecture: Design Process, Principles Leading to Good Design, Techniques for Making Good Design Decisions, Good Design Document, Software Architecture, Architectural Patterns: The Multilayer, Client-Server, Broker, Transaction Processing, Pipe & Filter And MVC Architectural Patterns.

Design Patterns: Abstraction-Occurrence, General Hierarchical, Play-Role, Singleton, Observer, Delegation, Adaptor, Façade, Immutable, Read-Only Interface and Proxy Patterns.

UNIT- IV

Software Testing: Effective and Efficient Testing, Defects in Ordinary Algorithms, Numerical Algorithms, Timing and Co-ordination, Stress and Unusual Situations, Testing Strategies for Large Systems.

Software Project Management: Introduction to Software Project Management, Activities of Software Project Management, Software Engineering Teams, Software Cost Estimation,

Project Scheduling, Tracking And Monitoring.

Software Process Models: Waterfall Model, The Phased Released Model, The Spiral Model, Evolutionary Model, The Concurrent Engineering Model, Rational Unified Process.

UNIT- I

Introduction to Object Oriented Software Engineering:

Object-oriented software engineering (commonly known by acronym **OOSE**) is an object-modeling language and methodology.

An **object-modeling language** is a standardized set of symbols used to model a software system using an object-oriented framework.

A modeling language is usually associated with a methodology for object-oriented development. The modeling language defines the elements of the model.

E.g., that a model has classes, methods, object properties, etc. The methodology defines the steps developers and users need to take to develop and maintain a software system. Steps such as Define requirements, Develop code, and Test system.

Methodology is the systematic, theoretical analysis of the methods applied to a field of study. It comprises the theoretical analysis of the body of methods and principles associated with a branch of knowledge.

The term **software engineering** is the product of two words, **software**, and **engineering**. The **software** is a collection of integrated programs.

Software subsists of carefully-organized instructions and code written by developers on any of various particular computer languages. Computer programs and related documentation such as requirements, design models and user manuals.

Engineering is the application of scientific and practical knowledge to invent, design,build, maintain, and improve frameworks, processes, etc.

Define software process.

Software process is defined as the structured set of activities that are required to develop the software system.

What are the fundamental activities of a software process?

1. Specification.
2. Design and implementation.
3. Validation.
4. Evolution

What is System Engineering?

System Engineering means designing, implementing, deploying and operating systems which include hardware ,software and people.

What is requirement engineering?

Requirement engineering is the process of establishing the services that the customer requires from the system and the constraints under which it operates and is developed.

What is Software Engineering?

Software Engineering is an engineering branch related to the evolution of software product using well-defined scientific principles, techniques, and procedures. The result of software engineering is an effective and reliable software product.

Why is Software Engineering required?

Software Engineering is required due to the following reasons:

- To manage Large software
- For more Scalability
- Cost Management
- To manage the dynamic nature of software
- For better quality Management

****

Need of Software Engineering

Huge Programming: It is simpler to manufacture a wall than to a house or building, similarly, as the measure of programming become extensive engineering has to step to give it a scientific process.

Adaptability: If the software procedure were not based on scientific and engineering ideas, it would be simpler to re-create new software than to scale an existing one.

Cost: As the hardware industry has demonstrated its skills and huge manufacturing has let down the cost of computer and electronic hardware. But the cost of programming remains high if the proper process is not adapted.

Dynamic Nature: The continually growing and adapting nature of programming hugely depends upon the environment in which the client works. If the quality of the software is continually changing, new upgrades need to be done in the existing one.

Quality Management: Better procedure of software development provides a better and quality software product.

****

Characteristics of a good software engineer

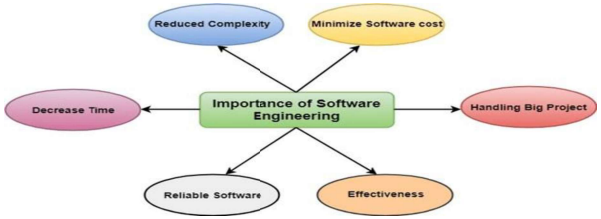
The features that good software engineers should possess are as follows:

- Exposure to systematic methods, i.e., familiarity with software engineeringprinciples.

- Good technical knowledge of the project range (Domain knowledge).
- Good programming abilities.
- Good communication skills. These skills comprise of oral, written, and interpersonal skills.
- High motivation.
- Sound knowledge of fundamentals of computer science.
- Intelligence.
- Ability to work in a team
- Discipline, etc.

****

Importance of Software Engineering



The importance of Software engineering is as follows:

1. **Reduces complexity:** Big software is always complicated and challenging to progress. Software engineering has a great solution to reduce the complication of any project. Software engineering divides big problems into various small issues. And then start solving each small issue one by one. All these small problems are solved independently to each other.
2. **To minimize software cost:** Software needs a lot of hard work and software engineers are highly paid experts. A lot of manpower is required to develop software with a large number of codes. But in software engineering, programmers project everything and decrease all those things that are not needed. In turn, the cost for software productions becomes less as compared to any software that does not use software engineering method.
3. **To decrease time:** Anything that is not made according to the project always wastes time. And if you are making great software, then you may need to run many codes to get the definitive running code. This is a very time-consuming procedure, and if it is not well handled, then this can take a lot of time. So if you are making your software according to the software engineering method, then it will decrease a lot of time.
4. **Handling big projects:** Big projects are not done in a couple of days, and they need lots of patience, planning, and management. And to invest six and seven months of any company, it requires heaps of planning, direction, testing, and maintenance. No one can say that he has given four months of a company to the task, and the project is still in its first stage. Because the company has provided many resources to the plan and it should be completed. So to handle a big project without any problem, the company has to go for a software engineering method.
5. **Reliable software:** Software should be secure, means if you have delivered the software, then it should work for at least its given time or subscription. And if any bugs come in the software, the company is responsible for solving all these bugs. Because in software engineering, testing and maintenance are given, so there is no worry of its reliability.
6. **Effectiveness:** Effectiveness comes if anything has made according to the standards. Software standards are the big target of companies to make it more effective. So Software becomes more effective in the act with the help of software engineering.

****

NATURE OF SOFTWARE: The nature of the software medium has manyconsequences for systems engineering (SE) of software-intensive systems.

These four properties are:

1. complexity,
2. conformity,
3. changeability,
4. invisibility.

Software and software projects are unique for the following reasons:

- Software has no physical properties;
- Software is the product of intellect-intensive teamwork;
- Estimation and planning for software projects is characterized by a high degree ofuncertainty, which can be at best partially mitigated by best practices;
- Risk management for software projects is predominantly process-oriented;
- Software alone is useless, as it is always a part of a larger system; and
- Software is the most frequently changed element of software intensive systems.

****

TYPES OF THESOFTWARE

Software, which is abbreviated as SW or S/W, is a set of programs that enables the hardware to perform a specific task. All the programs that run the computer are software. The software can be of three types: system software, application software, and programming software.

System Software: The system software is the main software that runs the computer. When you turn on the computer, it activates the hardware and controls and coordinates their functioning. The application programs are also controlled by system software.

Some other examples of system software include:

- Operating System:
  - BIOS.
  - An assembler

2) Application Software:

Application software is a set of programs designed to perform a specific task. It does not control the working of a computer as it is designed for end-users. A computer can run without application software. Application software can be easily installed or uninstalled as required. It can be a single program or a collection of small programs. Microsoft Office.

Suite, Adobe Photoshop, and any other software like payroll software or income tax software are application software. As we know, they are designed to perform specific tasks. Accordingly, they can be of different types such as:

1. **Word Processing Software.**
2. **Spreadsheet Software.**
3. **Multimedia Software.**
4. **Enterprise Software.**

3) Programming Software:

It is a set or collection of tools that help developers in writing other software or programs. It assists them in creating, debugging, and maintaining software or programs or applications. We can say that these are facilitator software that helps translate programming language such as Java, C++, Python, etc., into machine language code. So, it is not used by end-users. For example, compilers, linkers, debuggers, interpreters, text editors, etc. This software is also called a programming tool or software development tool.

Some examples of programming software include:

1. **Eclipse:** It is a java language editor.
2. **Coda:** It is a programming language editor for Mac.
3. **Notepad++:** It is an open-source editor for windows.
4. **Sublime text:** It is a cross-platform code editor for Linux, Mac, and Windows.

1. **Networking and Web Applications Software –** Networking Software provides the required support necessary for computers to interact with each other and with data storage facilities. The networking software is also used when software is running on a network of computers (such as the World Wide Web). It includes all network management software, server software, security and encryption software, and software to develop web-based applications like HTML, PHP, XML, etc.
2. **Embedded Software –** This type of software is embedded into the hardware normally in the Read-Only Memory (ROM) as a part of a large system and is used to support certain functionality under the control conditions. Examples are software used in instrumentation and control applications like washing machines, satellites, microwaves, etc.
3. **Reservation Software –** A Reservation system is primarily used to store and retrieve information and perform transactions related to air travel, car rental, hotels, or other activities. They also provide access to bus and railway reservations, although these are not always integrated with the main system. These are also used to relay computerized information for users in the hotel industry, making a reservation and ensuring that the hotel is not overbooked.
4. **Business Software –** This category of software is used to support business applications and is the most widely used category of software. Examples are software for inventory management, accounts, banking, hospitals, schools, stock markets, etc.
5. **Entertainment Software –** Education and entertainment software provides a powerful tool for educational agencies, especially those that deal with educating young children. There is a wide range of entertainment software such as computer games, educational games, translation software, mapping software, etc.
6. **Artificial Intelligence Software –** Software like expert systems, decision support systems, pattern recognition software, artificial neural networks, etc. come under this category. They involve complex problems which are not affected by complex computations using non-numerical algorithms.
7. **Scientific Software –** Scientific and engineering software satisfies the needs of a scientific or engineering user to perform enterprise-specific tasks. Such software is written for specific applications using principles, techniques, and formulae specific to that field. Examples are software like MATLAB, AUTOCAD, PSPICE, ORCAD, etc.
8. **Utilities Software –** The programs coming under this category perform specific tasks and are different from other software in terms of size, cost, and complexity. Examples are anti-virus software, voice recognition software, compression programs, etc.
9. **Document Management Software –** Document Management Software is used to track, manage and store documents in order to reduce the paperwork. Such systems are capable of keeping a record of the various versions created and modified by different users (history tracking). They commonly provide storage, versioning, metadata, security, as well as indexing and retrieval capabilities.

****

Software Engineering Activities:

Software Development Life Cycle, SDLC for short, is a well-defined, structured sequence of stages in software engineering to develop the intended software product.

SDLC Activities:- SDLC provides a series of steps to be followed to design and develop a software product

efficiently. SDLC framework includes the following steps:

Communication: This is the first step where the user initiates the request for a desired software product. He contacts the service provider and tries to negotiate the terms. He submits his request to the service providing organization in writing.

Requirement Gathering: This step onwards the software development team works to carry on the project. The team holds discussions with various stakeholders from problem domain and tries to bring out as much information as possible on their requirements. The requirements are contemplated and segregated into user requirements, system requirements and functional requirements. The requirements are collected using a number of practices as given -

- studying the existing or obsolete system and software,
- conducting interviews of users and developers,
- referring to the database or
- Collecting answers from the questionnaires.

Feasibility Study: After requirement gathering, the team comes up with a rough plan of software process. At this step the team analyzes if software can be made to fulfill all requirements of the user and if there is any possibility of software being no more useful. It is found out, if the project is financially, practically and technologically feasible for the organization to take up. There are many algorithms available, which help the developers to conclude the feasibility of a software project.

System Analysis: At this step the developers decide a roadmap of their plan and try to bring up the best software model suitable for the project. System analysis includes Understanding of software product limitations, learning system related problems or changes to be done in existing systems beforehand, identifying and addressing the impact of project on organization and personnel etc. The project team analyzes the scope of the project and plans the schedule and resources accordingly.

Software Design: Next step is to bring down whole knowledge of requirements and analysis on the desk and design the software product. The inputs from users and information gathered in requirement gathering phase are the inputs of this step. The output of this step comes in the form of two designs; logical design and physical design. Engineers produce meta-data and data dictionaries, logical diagrams, data-flow diagrams and in some cases pseudo codes.

Coding: This step is also known as programming phase. The implementation of software design starts in terms of writing program code in the suitable programming language and developing error-free executable programs efficiently.

Testing: An estimate says that 50% of whole software development process should be tested. Errors may ruin the software from critical level to its own removal. Software testing is done while coding by the developers and thorough testing is conducted by testing experts at various levels of code such as module testing, program testing, product testing, in-house testing and testing the product at user's end. Early discovery of errors and their remedy is the key to reliable software.

Integration: Software may need to be integrated with the libraries, databases and other program(s). This stage of SDLC is involved in the integration of software with outer world entities.

Implementation: This means installing the software on user machines. At times, software needs post-installation configurations at user end. Software is tested for portability and adaptability and integration related issues are solved during implementation.

Operation and Maintenance: This phase confirms the software operation in terms of more efficiency and less errors. If required, the users are trained on, or aided with the documentation on how to operate the software and how to keep the software operational. The software is maintained timely by updating the code according to the changes taking place in user end environment or technology. This phase may face challenges from hidden bugs and real-world unidentified problems.

Disposition: As time elapses, the software may decline on the performance front. It may go completely obsolete or may need intense up gradation. Hence a pressing need to eliminate a major portion of the system arises. This phase includes archiving data and required software components, closing down the system, planning disposition activity and terminating system at appropriate end-of-system time.

****

Introduction to Object Orientation: Object-Oriented Design

In the object-oriented design method, the system is viewed as a collection of objects (i.e., entities). The state is distributed among the objects, and each object handles its state data. For example, in a Library Automation Software, each library representative may be a separate object with its data and functions to operate on these data. The tasks defined for one purpose cannot refer or change data of other objects. Objects have their internal data which represent their state. Similar objects create a class. In other words, each object is a member of some

class. Classes may inherit features from the super class.

The different terms related to object design are:

1. **Objects:** All entities involved in the solution design are known as objects. For example, person, banks, company, and users are considered as objects. Every entity has some attributes associated with it and has some methods to perform on the attributes.
2. **Classes:** A class is a generalized description of an object. An object is an instance of a class. A class defines all the attributes, which an object can have and methods, which represents the functionality of the object.
3. **Messages:** Objects communicate by message passing. Messages consist of the integrity of the target object, the name of the requested operation, and any other action needed to perform the function. Messages are often implemented as procedure or function calls.
4. **Abstraction** In object-oriented design, complexity is handled using abstraction. Abstraction is the removal of the irrelevant and the amplification of the essentials.
5. **Encapsulation:** Encapsulation is also called an information hiding concept. The data and operations are linked to a single unit. Encapsulation not only bundles essential information of an object together but also restricts access to the data and methods from the outside world.
6. **Inheritance:** OOD allows similar classes to stack up in a hierarchical manner where the lower or sub-classes can import, implement, and re-use allowed variables and functions from their immediate super classes. This property of OOD is called an inheritance. This makes it easier to define a specific class and to create generalized classes from specific ones.
7. **Polymorphism:** OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned the same name. This is known as polymorphism, which allows a single interface is performing functions for different types. Depending upon how the service is invoked, the respective portion of the code gets executed.

****

REUSABILITY IN SOFTWARE ENGINEERING:

Reuse: building on the work and experience of others.

The following are some of the types of reuse practiced by software engineers, in increasing order according to the potential amount of work that can be saved by the reuse.

Reuse of expertise: Software engineers who have many years of experience working on projects can often save considerable time when it comes to developing new systems because they do not need to re- think many issues: their past experience tells them what needs to be done. If such people write articles describing their experiences, this can help others to do better engineering work.

Reuse of standard designs and algorithms. There are thousands of algorithms and other aspects of designs described in various books, standards documents and articles. These represent a tremendous wealth for the software designer, since all he or she needs to do is to implement them if they are appropriate to the current task.

Reuse of libraries of classes or procedures, or of powerful commands built into languages and operating systems. Libraries and commands represent implemented algorithms, data structures and other facilities. Software developers always do this kind of reuse to some extent since all programming languages come with some basic libraries.

Reuse of frameworks. Frameworks are libraries containing the structure of entire applications or subsystems. To complete the application or subsystem, you merely need to fill in certain missing details. A framework can be written in any programming language.

Reuse of complete applications. You can take complete applications and add a small amount of extra software that makes the applications behave in special ways the client wants.

****

Example: POSTALCODES:

The example is divided into three elements. The first element is a hierarchy representing postal codes of different countries. The second element is a new exception class. The third element is the postal test class that allows the user to enter postal codes and test the facilities of the postal code hierarchy.

The postal code hierarchy:

The following are some design decisions you should study in postal code and its subclasses:

Postal code is declared as abstract, meaning that no instance can be created.

Two of its operations, validate and get country, are abstract, meaning that they mustbe given concrete implementations in subclasses.

The operation validate is protected, and is called by the constructor. It's concrete implementations in each subclass will throw a postal code exception if the format ofthe code is invalid.

All the instance variables are declared private. All other classes, including subclasses, can only access them using methods. This helps to improveencapsulation.

There is a to string method, as should be provided in most java classes. There are three examples of subclasses of postal code. Each of these implements the twoabstract operations. For example, the validate method of one subclass,

Canadian postal code, ensures that the format is XNX NXN, where N is a number and X is a letter; the first letter is also taken from a restricted set. The other implementations of validate ensure that US postal codes have an all- numeric format, while British postal codes adhere to their more complex alphanumeric format.

The postal code Exception class:

Postal Code Exception illustrates the concept of the user- defined exception class. Instances of this class are thrown when an error is found while validating a postal code.

A class that manipulates postal codes could choose to handle such exceptions in any way itwishes.

The user interface class Postal Test:

The user interface class, Postal Test, has only a static main method and one private static helper method called get Input. The code prompts the user for input and then attempts to create an instance of one of the subclasses of Postal Code. If a Postal Code Exception is thrown, it tries to create an instance of other subclasses until none remain. Then it prints out information about the result.

It would be possible to put all the code from Postal Test into Postal Code- the mainmethod in Postal Code would then simply be used to test the class.

****

GEOMETRIC POINTS

The classes described in this section represent points on a 2- dimensional plane. From mathematics, we know that to represent a point on a plane, you can use X and y coordinates, which are called Cartesian coordinates. Alternatively, you can use polar coordinates, represented by a radius(often called rho) and an angle(often called theta). In the code we have provided, you can interchangeably work with a given point as Cartesian coordinates or polar coordinates.

Point CP.	**Point CP Test**
Get X ().	Main ()
get Y()	
get Rho()	
get Theta()	
Convert Storage To Cartesian ()	
Convert Storage To Polar ()	
To String ()	

Classes for representing points using both Cartesian and polar coordinates. Only the operations are shown

Java already has classes for representing geometric points. Take a few moments to look at classes Point2D and point in the java documentation. We will call the point class presented here Point CP; it's main distinguishing feature from the built- in java classes is that it can handle both Cartesian and polar coordinates. We also provide a class called Point CP Test which, like Postal Test, simply provides a user interface for testing. The public methods of both classes are shown in figure.

Class Point CP contains two private instances variables that can store X and y, or else rho and theta. No matter which storage format is used, all four possible parameters can be computed. Users of the class can also call methods Convert Storage To Polar or Convert Storage To Cartesian in order to explicitly convert the internal storage of an instance to the alternative format.

****

Requirements Engineering

Domain Analysis:

The process by which a software engineer learns about the domain to betterunderstand the problem.

The domain is the general field of business or technology in which the clientswill use the software.

A domain expert is a person who has a deep knowledge of the domain.

Benefits of performing domain analysis:

1. **Faster development**- you will be able to communicate with the stakeholders moreeffectively, hence you will be able to establish requirements more rapidly.
2. **Better system**- knowing the subtleties of the domain will help ensure that the solutions you adopt will more effectively solve the customers problem.
3. **Anticipation of extensions**- armed with domain knowledge, you will obtaininsights into emerging trends and you will notice opportunities for future development.

Domain analysis document:

1. Introduction: name of the domain and give the motivation for performing the analysis.

2. Glossary: describe the meanings of all terms used in the domain that are either not part of everyday language or else have special meanings.

3. General knowledge about the domain: summarize important facts or rules that are widely known by the domain experts. Such knowledge includes scientific principles, business processes, analysis techniques and how any technology works.

4. Customers and users: describe who will or might buy the software, and in what industrial sectors they operate.

5. The environment: describe the equipment and systems used.

6. Tasks and procedures currently performed: make a list of what the various people do as they go about their work.

7. Competing software: describe what software is available to assist the users and customers, including software that is already in use and software on the market. Discussit's advantages and disadvantages.

8. Similarities to other domains: understanding what is generic versus what is specific will help you to create software that might be more reusable or more widely marketable.

****

PROBLEM DEFINITION AND SCOPE:

A problem can be expressed as difficulty the users or customers are facing (OR) as an opportunity that will result in some benefit such as improved productivityor sales.

The solution to the problem normally will entail developing software.

Defining the scope

Narrow the scope by defining a more precise problem List all the things you might imagine the system doing.

- Exclude some of these things if too broad
- Determine high- level goals if too narrow
- Example: A university registration system
- Initial list of problems with very broad scope:
- Browsing courses, registering, fee payment, room allocation, exam scheduling.
- Narrowed scope: browsing courses, registering, fee payment
- Scope of another system: room allocation, exam scheduling.

In the university registration example you could consider a students goal to be ' completing the registration process'. However, you can see that the students higher level goal might be, ' obtaining their degree in the shortest reasonable time while taking courses that they find most interest and fulfilling'. This new goal sheds a different light on the problem; you might consider adding features to the system that would not otherwise have occurred to you, such as actively proposing courses based on an analysis of the students academic and personal-interest profiles.

****

REQUIREMENTS:

It is a statement describing either

- An aspect of what the proposed system must do, or a constraint on the systemsdevelopment.
- In either case it must contribute in some way towards adequately solving the customers problem;
- The set of requirements as a whole represents a negotiated agreement among thestakeholders.

A collection of requirements is a requirements document.

TYPES OF REQUIREMENTS:

1. Functional requirements. Describes what system should do.

Functional requirements describe what the system should do; in other words, they describethe services provided for the users and for other systems.

The functional requirements should include

1. Everything that a user of the system would need to know regarding what the system does, and
2. Everything that would concern any other system that has to interface to this system.

The functional requirements can be further categorized as follows:

1. What inputs the system should accept
2. What outputs the system should produce
3. What data the system should store that other systems might use
4. What computations the system should perform
5. The timing and synchronization of the above

2.Non functional requirements:

Quality requirements- Constraints on the design to meet specified levels of quality.

Platform requirements- Constraints on the environment and technology of the system

Process requirements.-Constraints on the project plan and development methods.

Quality requirements

a. Quality requirements ensure the system possesses quality attributes such as usability, efficiency, reliability, maintainability and reusability.
b. The following are some of the main categories of quality requirements:
c. **Response time**: for systems that process a lot of data or use a network extensively, you should require that the system gives result or feedback to the user in a certain minimum time.
d. **Throughput**: computations or transactions per minute.
e. **Resource usage**: for systems that use non trivial amounts of such resources as memory and network bandwidth, you should specify the maximum amount of these resources that the system will consume.
f. **Reliability:** reliability is measured as the average amount of time between failures or the probability of a failure in a given period.
g. **Availability**: availability measures the amount of time that a server is runningand available to respond to users.
h. **Recovery from failure**: they state that if the hardware or software crashes, or the power fails, then the system will be able to recover within a certain amount of time, and with a certain minimal loss of data.
i. **Allowances for maintainability and enhancement**: in order to ensure that the system can be adapted in the future, you should describe changes that are anticipated for subsequent releases.
j. **Allowances for reusability**: it is desirable in many cases to specify that a certain percentage of the system,e.g.40% , measured in terms of lines of code, must be designed generically so that it can be reused.

Platform requirements

1. This type of requirements constraints the environment and technology of the system:
2. **Computing platform**: it is normally important to make it clear what hardware and operating system the software must be able to work on. Such requirements specify the least powerful platforms and declare that it must work on anything more recent or more powerful.
3. **Technology to be used:** common examples are to specify the programming language or database system. Such requirements are normally stated to ensurethat all systems in an organization use the same technology- this reduces theneed to train people in different technologies.

Process requirements:

The final type of requirements contains the project plan and development methods:

Development process to be used: in order to ensure quality, some requirements documents specify that certain processes be followed. A reference should be made to otherdocuments that describe the process.

Cost and delivery date: These are important constraints. However, they are usually not placed in the requirements document, but are found in the contract for the system or are left to a separate project plan document.

****

Techniques For Gathering And Analysis Requirements

You can gather requirements from the same sources of information as you used for domain analysis: i.e. from the various stakeholders, from other software systems, and from any documentation that might be available.

1.Observation: Taking a notebook and shadowing important potential users as they do their work, writing down everything they do.

You can also ask users to talk as they work, explaining what they are doing. You can videotape the session so that you can analyze it in more detaillater.

2.Interviewing:

1. It is a widely used technique. Conduct a series of interviews.
2. Ask about specific details such as maximum and minimums, whether there areany exceptions to rules and what possible changes might be anticipated.
3. Ask about the stakeholders vision for the future. This question may elicit innovative ideas and suggest what flexibility should be built into the system.
4. If a customer or user presents concrete ideas for their view of the system, ask if they have any alternative ideas.
5. Ask what would be a minimally acceptable solution to the problem.
6. Ask for other sources of information. The stakeholder you are interviewing may have interesting documents or may know someone with usefulknowledge.
7. Have the interviewee draw diagrams. The diagram could show such things as
8. The flow of information, the chain of command, how some technology works.

3.Brainstorming

1. Brainstorming is an effective way to gather information from a group of people.

2. The general idea is that the group sits around a table and discusses some topic with the goal of generating ideas.

3. The following is a suggested approach to organizing and running an effective.

Brainstorming session:

1. Call a meeting with representation from all stakeholders. Effective brainstorming session can be run with five to 20 people.

   Appoint an experienced moderator (also known as a facilitator)- that is, someone who knows how to run brainstorming meeting and will lead the process. The moderator may participate in the discussion if he or she wishes.

2. Arrange the attendees around the periphery of a table and give them plenty of paper to work with.

i. Decide on a trigger question. This is a key step in the process. A trigger question is one for which the participants can provide simple one line answers that are more than just numbers or yes or no responses.

ii. Ask each participant to follow these instructions:

iii. Think of an answer to the trigger question, no matter how trivial orquestionable the answer is

Write the answer down in one or two lines on a sheet of paper, one idea per sheet.

1. Pass the paper to the neighbor on your left to simulate his or herthoughts
2. Look at the answers passed from your neighbor to the right and pass these on to your left as well. Use the ideas you have read to simulate your own ideas.
3. Continue step 5 until ideas stop flowing or a fixed time(5-15 minutes) passes.
4. Moving around the table, ask everybody to read out one of the ideas on the sheets that happen to be In front of them.
5. After a fixed time period, or after all ideas have been recorded on the flip-Chart, the group may take a series of votes to prioritize them.
6. The concept of passing ideas clockwise around the table.

Prototyping:

1. The simplest kind: paper prototype.
2. A set of pictures of the system that are shown to users in sequence to explainwhat would happen.
3. The most common: a mock- up of the system' s UI
4. Written in a rapid prototyping language
5. Does not normally perform any computations, access any databases or interactwith any other systems.
6. May prototype a particular aspect of the system.

****

TYPES OF REQUIREMENTS DOCUMENTS

To perform good software engineering, it is always appropriate to write down requirements. The level of detail of the requirements can, however, vary significantly from project to project. At one extreme, there are documents that informally outline the requirements using a few paragraphs or simple diagrams. At the other extreme, there are specifications that contain thousands of pages of intricate detail.

Requirements documents for large systems are normally arranged in a hierarchy:

Level of detail required in a requirements document

a. **The size of the system**: A large system will need more detailed requirements for several reasons. First, there

is simply more to say. Second, the system will need to be divided into subsystems so that different teams can work on each part.

b. **The need to interface to other systems**: Even a small system will need to have well- described requirements if other systems or subsystems are going to use its services or communicate with it.

c. **The target audience**: The requirements must be written at a high enough level so that the potential users can read them.

d. **The contractual arrangements for development**: If you are arranging a contract by which a third party will develop software for you, then you will have to specify the requirements with considerable precision.

e. **The stage in requirements gathering**: At an early stage in requirements gathering, it is important not to write large volumes of precise and detailed requirements.

f. **The level of experience with the domain and the technology:** If you are developing software in a well known domain and using well known technology, then you should be able to procedure a complete requirements document before starting to design the system.

g.**The cost incurred if the requirements are faulty.** Any system that, if itfails, the environment must be precisely specified.

****

REVIEWING REQUIREMENTS:

Each individual requirement should be carefully reviewed. In order to be acceptable,a requirement should:

1. **Have benefits that outweigh the costs of development**: Cost- benefit analysis is an important skill in software engineering. You sum, in financial terms, the benefits of the requirement and compare this to the sum of the costs.
2. **Be important for the solution of the current problem**. Many ideas might be useful to implement, and might have benefits that outweigh their costs. One of the most important rules in software engineering is the 80-20 rule, which says that 80 % of the usersproblem can often be solved with 20 % of the work. You should initially consider producing only that first 20 % of the system. The 80-20 rule is also called the Pareto principle.
3. **Be expressed using a clear and consistent notation**. Each requirement should be expressed using language that the customers can understand and should be consistent with the other requirements. Requirements are normally expressed in a natural language such as English.
4. **Be unambiguous.** It is typical to find that an English sentence can have more than one interpretation.
5. **Be logically consistent**. You should check consistency with any standards, with other requirements in the document, with higher level requirements and with the requirements for other subsystems.
6. **Lead to a system of sufficient quality.** A requirement should contribute to a system that is sufficiently usable, safe, efficient, reliable and maintainable.
7. **Be realistic with a variable resources**. A requirement is realistic if the development team has the expertise and technology to implement it on the required platform within the budget and time available.
8. **Be verifiable**. The requirements document will not only be the basis for a systems design, but also for testing the system. There must be some way that the system can be tested so as to clearly conclude whether or not the requirement has been correctly implemented.
9. **Be interested uniquely identifiable**. It is important to be able to refer to each individual requirement. This is necessary in requirements review meetings so that people can indicate which requirement they want to discuss. It is also necessary in design documents to be able to say which requirement is being implemented by a given aspect of the design, a quality called traceability. In some documents, each requirement is given a unique number.
10. **The document should be sufficiently complete.** Requirements document should have sections covering the following types ofinformation.

a. **Problem**: provide a succinct description of the problem the system is solving.

b. **Background information**: Give information that will help readers understand the requirements.

c. **Environment and system models**: provide the context in which the system runs and a global overview of the system or subsystem.

d. **Functional requirements**: Describe the services provided to the user and to other systems.

e. **Non functional requirements**: describe any constraints that must be imposed on the design of the system.

****

Case Studies:

Requirements specifications for high assurance secure systems are rare in the open literature. This paper presents a case study in the development of a requirements document for a multilevel secure system. The system is secure, yet combines popular commercial components with specialized high assurance ones. Functional and non-functional requirements pertinent to security are discussed. A multi-dimensional threat model is presented. The threat model accounts for the developmental and operational phases of system evolution and for each phase accounts for both physical and non-physicalthreats.

*****

GPS based Automobile Navigation System:

The following example requirements document is for an embedded system that will be

installed in special purpose hardware in cars.

Requirements for GANA software

1. **Problem**. GANA software will help drivers navigate by giving them directions totheir destination.
2. **Background information**. See domain analysis document.
3. **Environment and system models**. GANA software is to run on special GANAhardware, the hardware provides the following to the software:
   a) GPS position information
   b) a wireless internet connection to a map database
   c) position of a trackball
   d) a colour 10 cm by 10 cm LCD screen
   e) six buttons at the bottom of the screen and
   f) Input from the cars other systems containing data about speed and Turing ofthe steering wheel. This requirements document describes the software only.
4. **Functional requirements.**

1. The system uses GPS information to calculate which map to display. The system also integrates information about the cars speed and history of turns made in order to refine it's accuracy about the vehicles location
2. The system has two main interaction modes: in setup mode, the user consults maps and specifies the destination; in navigation mode, the system assists the user to navigate to the destination.
3. Setup mode: When the system is on, and the vehicle is stationary, it enters setup mode. If the vehicle is moving, the system enters navigation mode. In setup mode, the system displays a map. The default map is in 1: 25000 scale and is centered on the users current position. At this scale, the map covers a square with km side.

   When the users current position is within the visible part of the map, the system always indicates it with a red arrow. The arrow points in the direction the user is heading.

   The system also displays in orange the shortest route from the current position to the center of the map. It will not be possible to display the entire route if the current position is not displayed.

   When the user manipulates the trackball, the screen scrolls the map in the direction of rotation of the trackball, as if the user were grabbing the map.

   The LCD screen displays the labels Zoom out, Zoom In , Go current, Go destination, Set Destination and Navigate above the six buttons. The button works as follows:

- Zoom in and zoom out displays new maps. The scale of the map appears at the top right of the screen. There may be a delay retrieving a map, in which case the system displays the message.
- Retrieving map. If the map or network is unavailable for any reason, the system displays: sorry map not available near the top of the screen and continues to display the previous map.
- When the user presses Zoom in , the map scale is doubled so that a smaller region is displayed, with more in detail.
- When the user presses Zoom out the map scale is divided by 2 so that a larger region is displayed, with less local detail..
- When the user presses set destination, the location at the center of the screen is set as the destination. The shortest route from the current position to the destination is highlighted in red and is adjusted as the car moves.
- The shortest route to the set destination is shown on the top of the shortest route to the center of the screen (orange), and hence has precedence.
- When the user presses Go current, the map jumps so that it is centered of the current location.
- When the user presses Go destination, the map jumps so that it is centered over the destination. If no destination has been set, the destination defaults to the current location.
- When the user presses navigate or the vehicle starts moving, the system enters navigation mode described below.

4. **Navigation mode:**

A detailed map is never displayed in navigation mode since the userwould not be able to concentrate on driving while looking at the map.

If no destination has been set, the system just displays the name ofthe current highway or street and municipality in large type in addition, if a destination has been set, the system displays the following in as large a size as possible:

1. An arrow pointing up if the driver should drive starlight ahead, a left arrow if the driver should turn left, a right arrow if the driver should turn right and a U-turn symbol of the driver should turn around.

   The system displays the labels speak now, volume up, volume down, guide on, guide off and setup above the six buttons. The buttons works as follows:
2. **'Speak now'** produces a computer generated voice, reading the instructionsthat are on display. Every time the user presses the button, any reading in progress is canceled and the instructions are immediately read again startingfrom the beginning.
3. '**Volume up** 'and '**volume down** 'adjust sound output.
4. '**Guide on'** causes a computer generated voice to automatically readthe instructions one minute

in advance of any required driver action, such as exiting the highway, being needed.

5. '**Guide off**' cancels this function; the user would have to read the screen or press speak now. In situations where navigational action is required more frequently than once a minute, the voice reads the next instruction as soon asthe system detects that the driver has responded to the previous instruction.
6. **Setup**' switches to setup mode if the car is stationary. If the car is not stationary, the setup button is heated out and is inactive.
   If the driver does not respond as expected to the instructions, and takes a different route, the system immediately calculates a new route.

5. Quality requirements

1. The system will be robust in the case of failure of the internet connection or failureto receive the GPS signal, maintaining whatever service it can.
2. The system will be designed in a flexible way such that changes in wireless internetor GPS technology can be incorporated in future releases.
3. The system will be designed anticipating incorporation of input from an inertialnavigation unit that would take over in cases where GPS signals fail.