Chapter – 1

Introduction

History of Software Engineering:

The term software engineering first appeared in the late 1950's and early 1960's. Programs might have always known about civil, electrical and computer engineering and debated what engineering might mean for software.

The NATO science committee sponsored two conference on software engineering in 1968 and 1969 and many believes these conferences marked the official start of the profession of software engineering. The term software engineering gets importance after software crisis in 1965 to 1985.

Before 1960's, different individual proposed different solutions to problems like: some would suggest problems in management team, some team organization while some would argue for better languages and tools. Based on these facts, it was required to apply engineering approach for management, team organization, tools, theories, methodology and techniques implemented which finally gave rise to software engineering.

Software and Software Engineering:

❖ Software:

Software are programs that execute within a computer of any size and architecture and provides desired output. Software is a data structures that enable the programs to adequately manipulate information.

Who builds software?

- Software Engineers

Importance of Software:

- Touch every aspects of lives

❖ Software Engineering:

Software engineering encompasses a process, a collection of methods (practices) and an array of tools that allows professionals to build a high quality computer software. Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software and study of these applications of engineering to software.

It is important because it enables us to build complex systems in a timely manner and with high quality. Software engineering can be viewed as a layered technology.

ROLE OF SOFTWARE ENGINEERING:

1. To deal with the increasing complex nature of software, software engineering plays an important role.
2. Software engineering researches, designs and develops software systems to meet with clients requirements. Once the system had been fully designed software engineers then test, debug and maintain the system.
3. Generally, software engineering translates vague requirements and drives into precise specification.
4. It derives model of application and understand the behavior of performance.
5. Provides methods to schedule works, operate systems at various levels and obtain the necessary details of software.
6. Promotes interpersonal skills, communication skills and management skills.

SOFTWARE DEVELOPMENTS:

Software development is the process of developing software through successive phases in an orderly way. This process incudes not only the actual writing of code but also the preparation of requirements and objectives the design of what is to be coded and confirmation that what is developed has met objectives.

The complexity of modern systems and computer programs made the need of clearly and orderly development process for software. The orderly phases are:

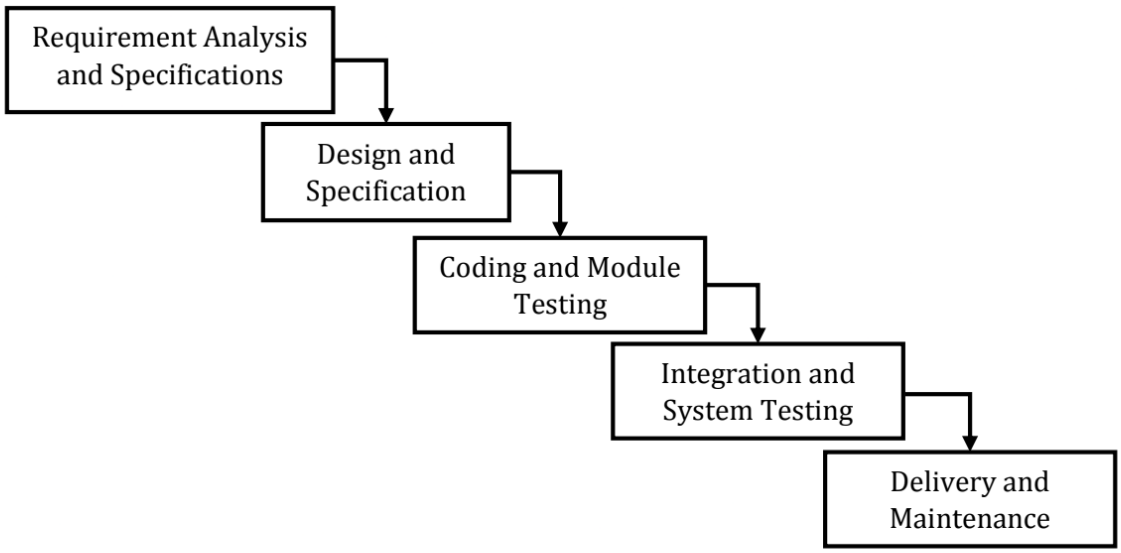


Fig: Waterfall Model

SOFTWARE DEVELOPMENT MODEL:

1. Waterfall Model:

Waterfall model is also known as the traditional or classical model and is popular in most software development and industrial practices. It is a structured process in which the activities are in linear and sequential phases. Similar to the name this model is just like a waterfall flowing downward the phases. The model uses an approach where one phase continues its operation and the previous phase is completed and the processes continuous till the last phase. The common phases of waterfall model are:

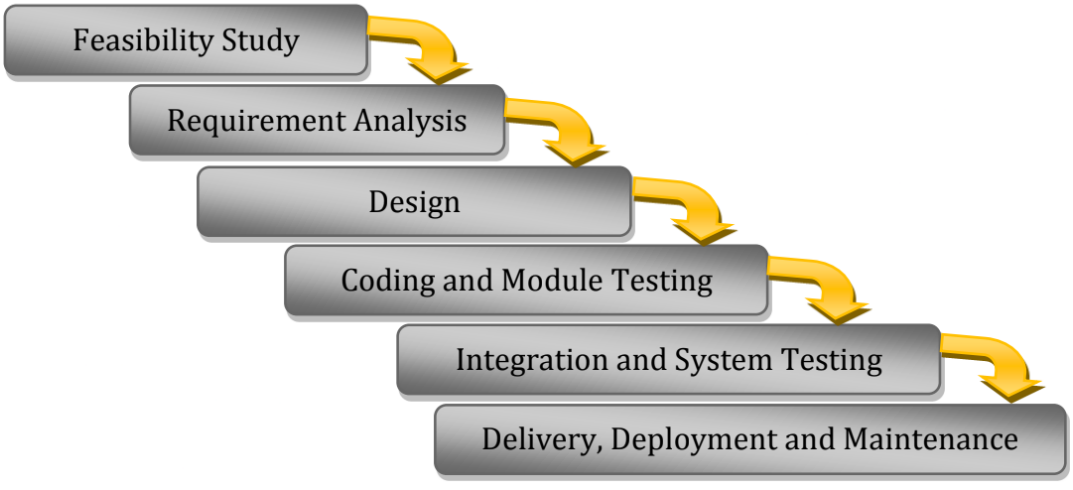


Fig: Waterfall Model

Advantages of Waterfall Model:

- This model is simple and easy to understand and use.
- It is easy to manage due to the rigidity of the model - each phase has specific deliverables and a review process.

- In this model phases are processed and completed one at a time. Phases do not overlap.
- Waterfall model works well for smaller projects where requirements are very well understood.

Disadvantages of Waterfall Model:

- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.

2. Spiral Model:

Spiral model is a risk driven process model for the development of software projects. It provides a framework for designing processes including the risk levels associated with them. A spiral model is cyclic model unlike that of the Waterfall Model which is linear programming model. It allows the rapid generation of subsequent phases during the software development phases. It also allows checking the robustness and correctness of the phases. After each cycle a prototype is developed and checked for its robustness and to meet the requirements.

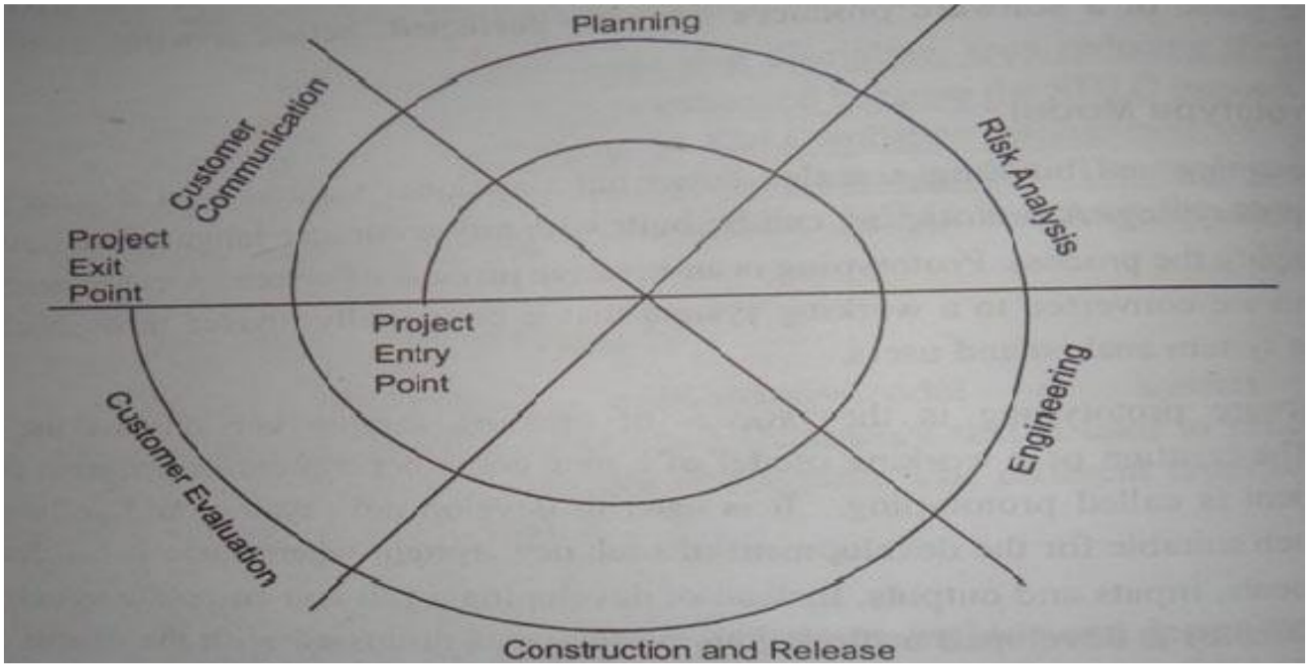


Fig: Spiral Model

Advantages of Spiral Model:

- High amount of risk analysis hence, avoidance of risk is enhanced.

- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional functionality can be added at a later date.
- Software is produced early in the software life cycle.
- Project estimates in terms of schedule, cost etc. become more and more realistic as the project moves forward and loops in spiral get completed.
- It is suitable for high risk projects, where business needs may be unstable. A highly customized product can be developed using this.

Disadvantages of Spiral Model:

- Can be costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.
- It is not suitable for low risk projects.
- May be hard to define objective, verifiable milestones.
- Spiral may continue indefinitely.

3. Prototype or Transformation Model:

A Prototype is a working model (sample) of software with some limited functionality. Prototyping approach of software development tries to implement a sample of the system that shows limited but measure functionality of the proposed system. The goal of the development of Prototyping is to counter the limitations of Waterfall Model. The Prototype displays the functionality of the product under development but may not hold the actual logic of the original software. It enables the customers to understand the system at an early stage of development and may give valuable feedback to the software developer to develop what the costumer actually expect.

Advantages of Prototype Model:

- Increase user involvement in the product even before implementation.
- Since a working model of the system is displayed, the users get a better understanding of the system being developed.
- Reduces time and cost as the defect can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily.
- Confusing or difficult functions can be identified.

Disadvantages of Prototype Model:

- Risk of insufficient requirement analysis owing to too much dependency prototype.
- Users may get confused in the prototypes and actual systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- Developers may try to reuse the existing prototypes to build the actual system, even when it is not technically feasible.
- The effort invested in building prototypes may be too much is not monitored properly.

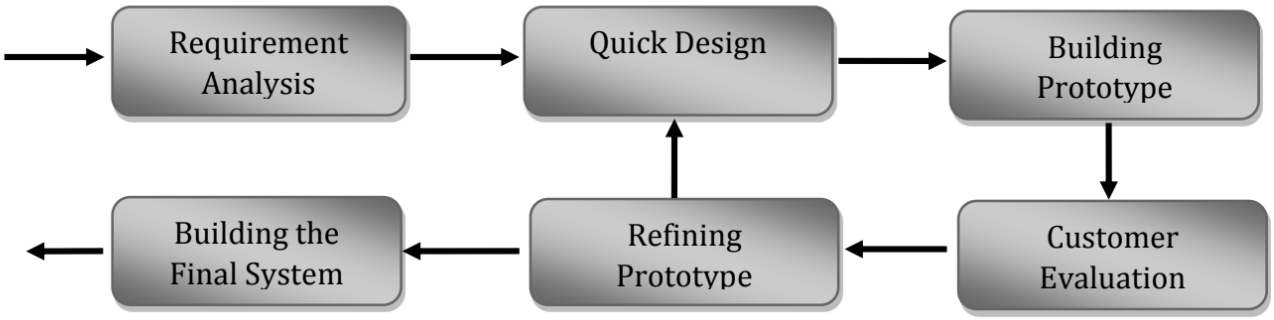


Fig: Prototype Model

4. **Iterative Model:**

A sequence of operation that can run multiple times is called iteration. In computer programming, iterative is used to describe a situation in which a sequence of instructions can be executed multiple times. One complete cycle through the sequence is called iteration. If the sequence is executed repeatedly, it is known as loop. In software development, iterative model uses a sequential planning and development process where an application is developed in small section of sequences called iterations. Each and every iteration are reviewed and analyzed by the development team and the potential end-users insight gained from. The analysis is used to improve and to determine the next step in the development. In software development, iterative model is the most popular model where the loops are used in every component of programs.

5. **The V-Model:**

The V-model represents a software development process which is an extension of waterfall model. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase to form a v-shape. This model demonstrates the relation between each phases of the development and its associated phase of testing. The horizontals access represents the time taken for completing the project and the vertical access represents definition and the implementation phase of the project.

6. **Rapid Application Development (RAD) model:**

Basic principle of RAD is to create a prototype of the working system in such a way that it reduces the time taken to build a complete package of the system. Since the emphasis of RAD is on reducing the time in development, the initial problem, requirement and decision analysis phases are combined and accelerated. The logical and physical design phases are also treated as a single phase and they are also accelerated. In each of the phases a prototype is created and tested. A

technical feedback of the prototype is analyzed and the system is revised with respect to the requirement and objectives of the system until a final system is placed into operation.

7. The Big Bang Model:

Big Bang Model of software or system development is less used model that is based on the Big Bang Theory. A huge amount of people, money, time and resources are put together, a lot of energy is used and the final system is produced. This model is very simple and there is little or no planning, scheduling or any other formal development process. All the efforts are spend on developing the software and delivering the solution on one row. This model is useful in projects where the requirements are very clear and well understood but the ratio of failure in projects are very high.

8. The Evolutionary Model:

The Evolutionary Model is based on theory of evolution in which a system gradually improves in its functionality, co-operation and the procedure over a considerable period of time. The system may get changes as per requirement in the changing environment of policies, technology and other factors. In the evolutionary model development effort is first made to establish correct requirement definition agreed by all the users in the organization. It is achieved to the applications of gradual processes to evolve a system best suited for a given situation. This process is iterative as it goes to a repetitive process of requirement analysis, designing, testing, prototyping, implementation and evaluation until the users are satisfied.

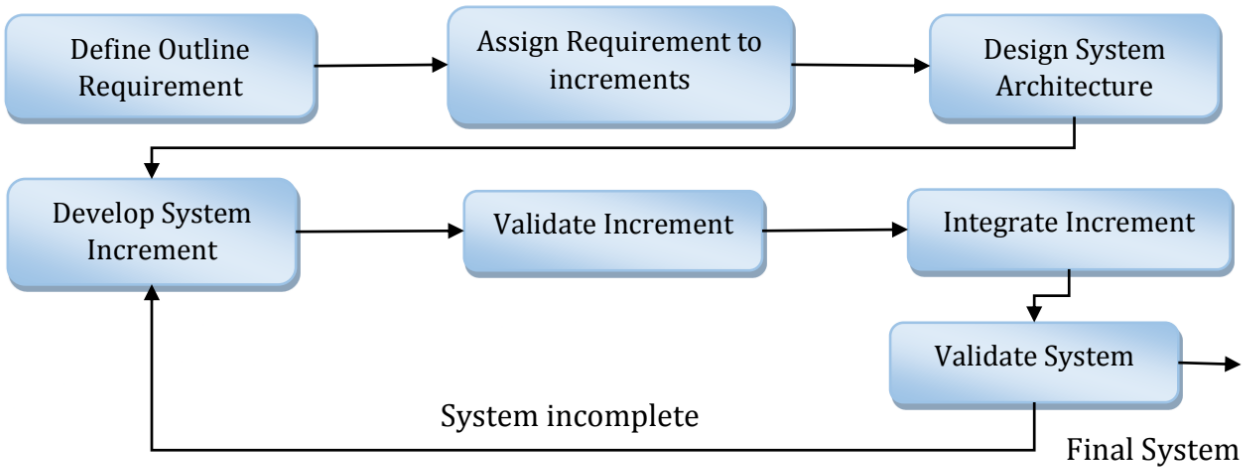


Fig: The Evolutionary Model

ATTRIBUTE OF GOOD SOFTWARE:

Maintainability is "the ease with which changes can be made to satisfy new requirements or to correct deficiencies". Well-designed software should be flexible enough to accommodate future changes that will be needed as new requirements come to light. Since maintenance accounts for nearly 70% of the cost of the software life cycle, the importance of this quality characteristic cannot be overemphasized. Quite often the programmer responsible for writing a section of code is not the one who must maintain it. For this reason, the quality of the software documentation significantly affects the maintainability of the software product.

Correctness is "the degree with which software adheres to its specified requirements". At the start of the software life cycle, the requirements for the software are determined and formalized in the requirements specification document. Well-designed software should meet all the stated requirements. While it might seem obvious that software should be correct, the reality is that this characteristic is one of the hardest to assess. Because of the tremendous complexity of software products, it is impossible to perform exhaustive execution-based testing to insure that no errors will occur when the software is run. Also, it is important to remember that some products of the software life cycle such as the design specification cannot be "executed" for testing. Instead, these products must be tested with various other techniques such as formal proofs, inspections, and walkthroughs.

Reusability is "the ease with which software can be reused in developing other software". By reusing existing software, developers can create more complex software in a shorter amount of time. Reuse is already a common technique employed in other engineering disciplines. For example, when a house is constructed, the trusses which support the roof are typically purchased preassembled. Unless a special design is needed, the architect will not bother to design a new truss for the house. Instead, he or she will simply reuse an existing design that has proven itself to be reliable. In much the same way, software can be designed to accommodate reuse in many situations. A simple example of software reuse could be the development of an efficient sorting routine that can be incorporated in many future applications.

Reliability is "the frequency and criticality of software failure, where failure is an unacceptable effect or behavior occurring under permissible operating conditions". The frequency of software failure is measured by the average time between failures. The criticality of software failure is measured by the average time required for repair. Ideally, software engineers want their products to fail as little as possible (i.e., demonstrate high correctness) and be as easy as possible to fix (i.e., demonstrate good maintainability). For some real-time systems such as air traffic control or heart monitors, reliability becomes the most important software quality characteristic. However, it would be difficult to imagine a highly reliable system that did not also demonstrate high correctness and good maintainability.

Portability is "the ease with which software can be used on computer configurations other than its current one". Porting software to other computer configurations is important for several reasons. First, "good software products can have a life of 15 years or more, whereas hardware is frequently changed at least every 4 or 5 years. Thus good software can be implemented, over its lifetime, on three or more different hardware configurations". Second, porting software to a new computer configuration may be less expensive than developing analogous software from scratch. Third, the sales of "shrink-wrapped software" can be increased because a greater market for the software is available.

DIFFERENCE BETWEEN SOFTWARE ENGINEERING AND SYSTEM ENGINEERING:

1. System Engineers focus more on users and domains, while Software Engineering focus more on implementing quality software.

2. System Engineer may deal with a substantial amount of hardware, software, people, database and other elements involved with that system which is going to be developed but typically software engineers will focus solely on software components.
3. System Engineers may have a broader education (including Engineering, Mathematics and Computer science), while Software Engineers will come from a Computer Science or Computer engineering background.
4. Software engineering is a part of system engineering.

SOME CHALLENGES OF SOFTWARE ENGINEERING:

- The methods used to develop small or medium scale projects are not suitable when it comes to the development of large scale or complex projects.
- Changes in software development are not avoidable.
- Production of high quality software
- Maintenance of software within acceptable cost.
- Informal communications which leads to waste of time, develop the completion of projects in time.
- Verification is difficult.
- Through testing consumes more resources for large projects.

SOFTWARE ENGINEERING ETHICS:

Software engineers shall commit themselves to make the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public software engineers shall adhere (follow) to the following principles:

1. **Public:**

Software engineers should act consistently with the public interest.

2. **Client and Employer:**

Software engineers shall act in a manner that is in the best interests of their client and employer.

3. **Product:**

Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

4. **Management:**

Software engineers/managers and leaders shall subscribe to and promote and ethical approach to management of software.

5. **Colleagues:**

Software engineers shall be fair to and supportive of their colleagues.

CHAPTER – 2

SOFTWARE SPECIFICATION

INTRODUCTION:

- A software specification or software requirement specification (SRS) is a requirement specification for a software project/system. It is a complete description of the behavior of a system to be developed and may include set of use-cases that describe interactions the users will have with the software.
- Enlists all necessary requirements that are required for project development.
- Contains the details of what is to be done and what is not to be done.
- Consists of functional and non-functional requirements.

Functional Requirements:

- Describes everything related to how the system works.
- Example: Functional requirement of a phone is to be able to make and receive calls.

Non-functional Requirements:

- Impose constraints on design or implementation (such as performance quality, design constraints, etc.)
- Example: For that some phone, "to be easy to use" is non-functional requirements

USES OF SPECIFICATION:

i. A statement of the user requirements or needs.
ii. A statement of the interface between the machine and the control environment.
iii. A statement of the requirement for the implementation.
iv. A reference point during product maintenance.
v. A legal contract.

SPECIFICATION QUALITIES:

NASA's Software Assurance Technology Center has identified the following as the ten important criteria that any SRS (Software Requirements Specifications) should satisfy:

1. **Complete**:

A complete requirements specification must precisely define all the real world situations that will be encountered and the capability's responses to them. It must not include situations that will not be encountered or unnecessary capability features.

2. **Consistent:**

System functions and performance level must be compatible and the required quality features (reliability, safety, security, etc.) must not contradict the utility of the system. For example, the

only aircraft that is totally safe is one that cannot be started, contains no fuel or other liquids, and is securely tied down.

3. **Correct**:

The specification must define the desired capability's real world operational environment, its interface to that environment and its interaction with that environment. It is the real world aspect of requirements that is the major source of difficulty in achieving specification correctness. The real world environment is not well known for new applications and for mature applications the real world keeps changing. The Y2K problem with the transition from the year 1999 to the year 2000 is an example of the real world moving beyond an application's specified requirements.

4. **Modifiable:**

Related concerns must be grouped together and unrelated concerns must be separated. Requirements document must have a logical structure to be modifiable.

5. **Ranked:**

Ranking specification statements according to stability and/or importance is established in the requirements document's organization and structure. The larger and more complex the problem addressed by the requirements specification, the more difficult the task is to design a document that aids rather than inhibits understanding.

6. **Testable:**

A requirement specification must be stated in such a manner that one can test it against pass/fail or quantitative assessment criteria, all derived from the specification itself and/or referenced information. Requiring that a system must be "easy" to use is subjective and therefore is not testable.

7. **Traceable:**

Each requirement stated within the SRS document must be uniquely identified to achieve traceability. Uniqueness is facilitated by the use of a consistent and logical scheme for assigning identification to each specification statement within the requirements document.

8. **Unambiguous:**

A statement of a requirement is unambiguous if it can only be interpreted one way. This perhaps, is the most difficult attribute to achieve using natural language. The use of weak phrases or poor sentence structure will open the specification statement to misunderstandings.

9. **Valid:**

To validate a requirements specification all the project participants, managers, engineers and customer representatives, must be able to understand, analyze and accept or approve it. This is the primary reason that most specifications are expressed in natural language.

10. **Verifiable:**

In order to be verifiable, requirement specifications at one level of abstraction must be consistent with those at another level of abstraction. Most, if not all, of these attributes are

subjective and a conclusive assessment of the quality of a requirements specification requires review and analysis by technical and operational experts in the domain addressed by the requirements.

CLASSIFICATION OF SPECIFICATION STYLES:

1. **Informal:**

Not formal, more flexible, leave more decision space to implementer.

2. **Formal:**

Use of formalisms make specifications precise and augmented automatic verification possibilities. They have syntax, their semantics fall in one domain and they are able to be used to take useful information.

3. **Semi-Formal:**

Often we use notation which semantics has not been defined so precisely (example: UML)

4. **Operational:**

Describe the system in terms of the expected behavior generally providing a model. Describes the operational detail. Example: DFD, FSM (Finite State Machine), etc.

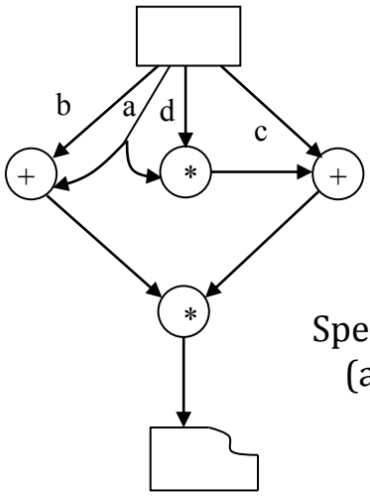
5. **Descriptive Specification:**

Describe the system in terms of desired properties for the system. Example: collaboration diagram, sequence diagram, etc.

OPERATIONAL SPECIFICATION:

1. **DFD (Data Flow Diagram):**

DFDs are well known and widely used notation for specifying the functions of an information system, and how data flow from function to function. They describes system as collection of functions that manipulate data.



Specifies evaluation of
(a + b) * (c + a * d)

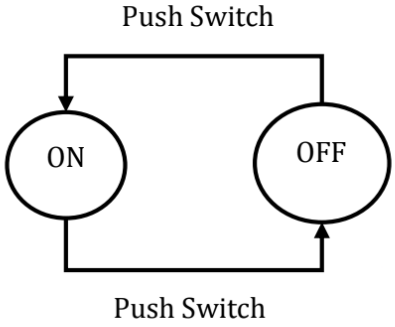
2. UML (Unified Modeling Language):

Consider an online subscription system (OSS), its purpose is to provide online learning materials for a specific field. The system has general users and registered users. General users can view and download free subscriptions and asks for membership. Registered users can view, download free subscriptions as well as can view download and search paid subscription after payment is performed. He/she can also asks for extension of validity. OSS administrator can check for validity of contents provided by the editors of the contents and upload the contents to the web. Finance administrator can keep track of payment done and about validity of users. Editors are assigned by OSS administrator in order to collect the online contents for the web.

3. FSM (Finite State Machine):

- Describe control flow
- FSM are widely known and important diagram in UML
- An FSM consists of following things:

a. Finite set of states, Q
b. Finite set of inputs, I
c. Transition function δ = Q * I -> Q
   δ can be some event or values
   Nodes represents states



Chapter – 3

Software Testing Techniques and Strategies

INTRODUCTION:

Testing is the process of evaluating the system or its components with the intent to find errors whether it satisfies the specified requirements or not. It is the process of executing a system in order to identify gaps, errors or missing requirements in contrary to the actual desire or requirements.

SOFTWARE TESTING FUNDAMENTALS:

1. **Testing Objectives:**
   - To demonstrate that faults are not present
   - To find errors
   - To ensure that all the functionalities are implemented
   - To ensure that customer will be able to get his/her work done

2. **Testing Life cycle:**
   - Never ending process
   - Start from early development of project to the late delivery of product
   - An early start of testing reduces the cost, time and rework then provides error free software that is delivered to client
   - In SDLC, software testing can be started from requirement gathering phase and last till the deployment of software.
   - However, it also depends on the development model we preferred for our project.
   - It is difficult to determine when to stop testing, as testing is never ending process and no one can say that any software is 100% tested.
   - When to stop testing:
     - ✓ If testing deadline meet
     - ✓ Completion of test case execution
     - ✓ Buy rate falls below certain level
     - ✓ Management decision, etc.

3. **Test Cases:**

A test case is a document that describes an input, action or event and an expected response, to determine if a feature of application is working correctly.

A test case should contain particulars such as test case name, objective, test conditions, input data requirements and expected results.

In order to design test cases, we use two approaches:

a. Black Box Testing:

Knowing the specified function, that a product has been designed to perform, tests can be conducted that demonstrate each function is fully operational while at the same time searching for errors in each function.

b. White Box Testing:

Knowing the internal workings of a product, tests can be conducted to ensure that all the internal operations are performed according to specifications and all the internal components have been adequately exercised.

LEVELS OF TESTING:

1. Unit Testing:

The unit testing focuses verification effort on the smallest unit of software design, the software component or module. During unit testing, the modules are tested in isolation i.e. the overall module is splitted into small units and the testing is performed over the small units.

Advantages:

- Reduces debugging effort
- If all modules were to be tested together, it may not be easy to determine which module has the error, so unit testing is necessary to find errors quickly.
- Testing can be carried out immediately
- In general, the goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionally.

Disadvantages:

- Time consuming
- Cost exceeds
- Does not removes bugs completely

2. Integration Testing:

Once all the modules have been unit tested, it needs to put them together i.e. interfacing is required. However, there may arise problems like data loss across an interface. One module can have inadvertent, adverse effect on another, sub-functions when combined may not produce the desired function, and global data structures can prevent problems and so on.

Therefore, the integration testing produces a systematic technique for constructing the problem structure while at the same time conducting tests to uncover errors associating with interfacing.

Two methods for integration testing:

- **Bottom Up Integration:** Test follows from lower to higher level modules.
- **Top Down Integration:** Test follows from higher level to lower level modules.

3. System Testing:

This testing tests the system as a whole. Once all the components are integrated, the application as a whole is tested to see that it meets quality standards. The testing is performed by the specialized testing team.

The system testing is the first testing in SDLC where the system is tested as a whole. The application is tested thoroughly to verify that it meets the functional and technical specifications. The application is tested on the environment which is very close to product environment where the application will be implemented.

4. Acceptance Testing:

Conducted by quality assurance team, who will gauge whether the application meets the intended specifications and satisfies the clients' requirements. The QA team will have a set of pre-written scenarios and test cases that will be used to test the application. More ideas will be shared about the application and more tests can be performed on it to gauge its accuracy and the reasons why the project was initiated. Not done to find errors, but it is important to point out the system crash and major error issues in the application.

5. Alpha and Beta Testing:

Most software project builders use a process called alpha and beta testing to uncover errors that only end users seem able to find.

The alpha test is conducted at the developer's site by a representative group of end users. Unit testing, integration testing and system testing when combined are known as alpha testing.

Beta test is conducted at one or more end user's site. Unlike, alpha testing the developer is not present. Therefore, beta test is a "live" application in an environment that cannot be controlled by the developer.

6. Manual V/s Automatic Testing:

Manual Testing	Automated Testing
❖ Manual testing is not accurate at all times due to human error, hence it is less reliable.	❖ Automated testing is more reliable, as it is performed by tools and/or scripts.
❖ Manual testing is time-consuming, taking up human resources.	❖ Automated testing is executed by software tools, so it is significantly faster than a manual approach.
❖ Investment is required for human resources.	❖ Investment is required for testing tools.
❖ Manual testing is only practical when the test cases are run once or twice, and frequent repetition is not required.	❖ Automated testing is a practical option when the test cases are run repeatedly over a long time period.
❖ Manual testing allows for human observation, which may be more useful if the goal is user-friendliness or improved customer experience.	❖ Automated testing does not entail human observation and cannot guarantee user-friendliness or positive customer experience.

7. Static V/s Dynamic Testing:

Static Testing	Dynamic Testing
❖ Static Testing is white box testing which is done at early stage if development life cycle. It is more cost effective than dynamic testing	❖ Dynamic Testing on the other hand is done at the later stage of development lifecycle.
❖ Static testing has more statement coverage than dynamic testing in shorter time	❖ Dynamic Testing has less statement stage because it is covers limited area of code
❖ It is done before code deployment	❖ It is done after code deployment
❖ It is performed in Verification Stage	❖ It is done in Validation Stage
❖ This type of testing is done without the execution of code.	❖ This type of execution is done with the execution of code.
❖ Static testing gives assessment of code as well as documentation.	❖ Dynamic Testing gives bottlenecks of the software system.
❖ In Static Testing techniques a checklist is prepared for testing process	❖ In Dynamic Testing technique the test cases are executed.
❖ Static Testing Methods include Walkthroughs, code review.	❖ Dynamic testing involves functional and nonfunctional testing

8. Tester Workbench:

The tester workbench is the process used to verify and validate the system structurally and functionally. To understand the testing methodology it is necessary. The tester workbench is one of the part of SDLC, which is comprised of many workbenches.

Example: The programmer's workbench for one of the step to build a system is:

- Input (programmer specification) is given to the producer (programmer)
- Work (example: coding and debugging) is performed; a procedure is followed and a product is developed (produced i.e. methodology is used)
- Work is checked to ensure the product meets the specifications and standards. If check finds no problem, the problem is released to the next workbench. If the check finds a problem, the product is sent back for rework.

A project team uses the workbench to guide them through unit test of computer code. The programmer takes the following steps:

- Given input products (example: program code) to the tester
- Perform work (example: execute unit test) follow a procedure and produce a product deliverables (example: the test result)
- Check work to ensure test results meets test specifications and standards and that test procedure was followed. If the check finds no problem, release the product (i.e. test result) to the next workbench. If the check process finds a problem, the product is sent back for rework.

9. 11 – Steps of Testing Process:

An 11 – step testing process follows the "V" concept of testing. The "V" represents both the software development process and the 11 – step software testing process. The first five steps

use verification as the primary means to evaluate the correctness of the interim development deliverables. Validation is used to test the software in an executable mode. Left side of "V" is for verification and right side of "V" shows the 11 steps of testing processes.

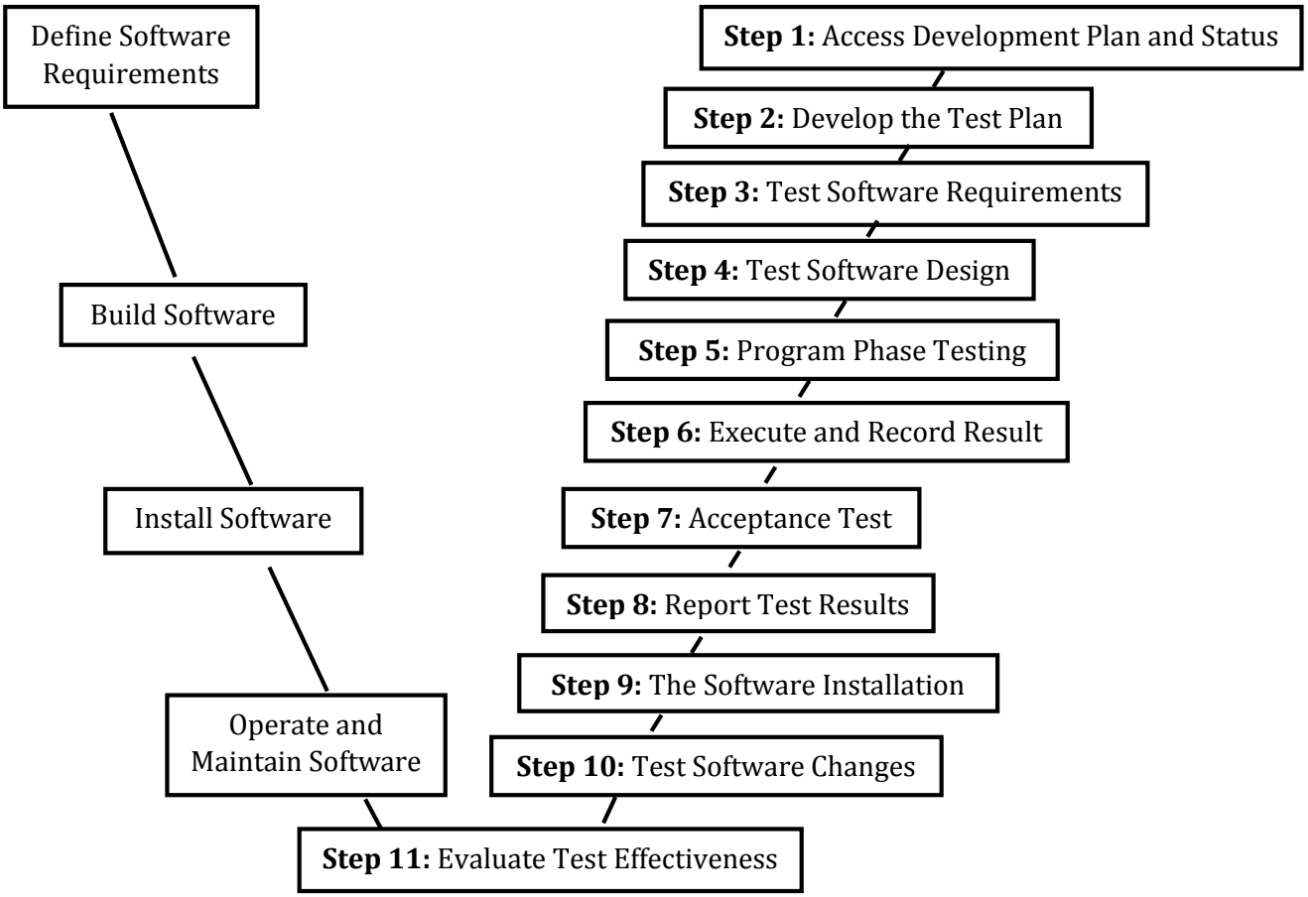


Fig: 11 – Steps of Testing Processes

DIFFERENT TYPES OF TESTING:

1. Installation Testing:

Installation testing is also known as deployment testing. In many cases, software must execute on a variety of platforms and under more than one operating system environments. This testing examines all installation procedures and specialized installation software (example: installers) that will be used by users and all documentation that will be used to introduce the software to end users. Example: A web application is necessary to test with all browsers.

2. Usability Testing:

Usability testing ensures that a good and user friendly GUI is designed and is easy to use for the end users. According to Nielson, usability can be defined in terms of 5 factors:

- Efficiency of use
- Learnability
- Memorability
- Safety

- Satisfaction

Usability testing deals with HCI (Human Computer Interaction)

3. **Regression Testing:**

Whenever a change in a software application is made, it is quite possible that other areas within the application have been affected by the change. To verify that a fixed bug hasn't result in another functionality violation is regression testing.

4. **Performance Testing:**

Performance testing is done to identify any bottlenecks or performance issues rather than finding bugs in software. The different causes that result in performance are:

- Network Delay
- Client Side Processing
- Database Transaction Processing
- Loading Balancing Between Servers

Performance testing is done in terms of following aspects:

- Speed
- Capability
- Stability/Durability
- Scalability

5. **Load Testing:**

Load testing is the process of testing the behavior of software by applying maximum load in terms of software accessing and manipulating large input data. It can be done at both normal and peak load conditions. It identifies capacity of software by using different tools such as Load Runner, Apache Jmeter, Visual Studio Load Tester, etc.

6. **Stress Testing:**

Testing of software under abnormal condition such as taking away the resources, apply load beyond the actual load limit, etc.

7. **Security Testing:**

Under security testing we can include following criteria:

- Confidentiality
- Integrity
- Authentication
- Authorization
- Vulnerabilities
- Session Management
- Input Validation

BLACK BOX AND WHITE BOX TESTING:

1. Functional Testing (Black-Box):

This type of testing is called black box testing which is based on specification of the software that is to be tested. The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for. Functional testing of the software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. The steps that are involved when testing an application for functionality are:

- The determination of the functionality that the intended application is meant to perform.
- The creation of test data based on the specifications of the application.
- The output based on the test data and the specifications of the applications.
- The writing of test scenarios and execution of test cases.
- The comparison of actual and expected results based on the executed test cases.

Black Box Approach:

The technique of testing without having any knowledge of the interior workings of the application. The tester is unclear about the system architecture and does not have access to source code. Typically when performing a black box test, the tester will interact with the system's user interface by providing inputs and examining the outputs without knowing how and where the inputs are worked upon.

2. Structural Testing (White Box):

Structural testing is also known as glass box or open box testing. It is the detailed investigation of the internal logic and structure of the code. In order to perform white box test on an application, the tester needs to have knowledge of the internal workings of the system. Inappropriateness of the code/module can be identified. It uses different path tests (cyclometer complexity). Expertise are necessary to conduct this test.

3. Domain Testing (Gray Box):

It is a technique to test the application with limited knowledge of the internal workings of an application.

In software testing, the term the more we know the better carries a lot of weight when testing and application.

4. Non – Functional Testing:

This testing is based upon the testing of the application from its non-functional attributes. Non-functional testing involves testing the software from the requirements which are non-functional in nature. Non-functional testing also give importance on fields such as performance, security, user interface, reliability, maintainability, usability, etc.

5. Validation Testing and Validation Testing Activities:

The set of activities that ensure that the software that has been built is traceable to customer requirements. When the software is completely assembled as a package and interfacing errors

have been uncovered and corrected validation is necessary. The different activities of validation testing are:

- Test Criteria
- Review
- Alpha Test and Beta Test

6. Black Box V/s White Box:

The Differences between Black Box Testing and White Box Testing are listed below.

Criteria	Black Box Testing	White Box Testing
Definition	Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is NOT known to the tester	White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester.
Levels Applicable To	Mainly applicable to higher levels of testing: Acceptance Testing System Testing	Mainly applicable to lower levels of testing: Unit Testing Integration Testing
Responsibility	Generally, independent Software Testers	Generally, Software Developers
Programming Knowledge	Not Required	Required
Implementation Knowledge	Not Required	Required
Basis for Test Cases	Requirement Specifications	Detail Design

Chapter – 4

Software Qualities and Software Quality Assurance

Software Qualities:

The term software quality describes to what extent is the software good rather best to be implemented for the purpose it has been proposed and developed. It is very important to maintain the software quality because every individuals or every companies are always willing to use the best system.

Thus, quality refers to the measurable characteristics things that we are able to compare to known standards such as length, color, electrical properties and malleability. Based upon the measurable characteristics of an item, software quality can be classified into two kinds:

1. **Quality of Design:**

It refers to the characteristics that designer specify for an item. The quality of software on the design basis is influenced by the grade of the materials, tolerances and performance specifications. Use of higher grade materials contribute to higher tolerances and thus specify greater levels of performances, finally increasing the design quality of product.

2. **Quality of Conformance:**

It is the degree to which the design specifications are followed during manufacturing. The greater the degree of conformances the higher the level of quality of conformances will be.

Software Quality Factors:

Developing methods that can produce high-quality software is another fundamental goal of software engineering. Factors that affects the quality of software are explained below:

1. **Correctness:**

Correctness is the extent to which a program satisfies its specifications.

2. **Reliability:**

Reliability is the property that defines how well the software meets its requirements.

3. **Efficiency:**

Efficiency is a factor relating to all issues in the execution of software; it includes considerations such as response time, memory requirement, and throughput.

4. **Usability:**

Usability, or the effort required locating and fixing errors in operating programs.

5. **Portability:**

Portability is the effort required to transfer the software from one configuration to another.

6. **Reusability:**

Reusability is the extent to which parts of the software can be reused in other related applications.

7. **Maintainability:**

Maintainability is the effort required to maintain the system in order to check the quality.

8. **Testability:**

Testability is the effort required to test to ensure that the system or a module performs its intended function.

9. **Flexibility:**

Flexibility is the effort required to modify an operational program.

SOFTWARE QUALITY ASSURANCE (SQA):

Software Quality Assurance is the process of providing the management with the data necessary to be informed about product quality, thus getting confidence that the product quality is meeting its goal. It provides the management with data that identify problems as a result the management will be conscious and responsive towards the problems and apply the necessary resources to solve the quality issues.

Various persons like software engineers, project managers, customers, sales persons and individuals who serve within SQA group are responsible for the software quality assurance. The SQA group serves as the customer's representative i.e. people who perform SQA must look at the software from customers point of view.

SOFTWARE QUALITY ASSURANCE ACTIVITIES:

Some of the SQA activities performed by the SQA group are as follows:

1. Prepares the SQA plan for the project.
2. Participates in the development of the project's software process description.
3. Review software engineering activities.
4. They audits designated software work products.
5. They ensure that deviation's in documented and handled according to a documented procedure.
6. Records any non-compliance reports to senior management. Non-compliance items are tracked until they are resolved.

SOFTWARE QUALITY STANDARDS:

Many organizations around the globe develop and implement different standards to improve the quality needs of their software.

1. International Standard Organization:

❖ ISO/IEC 9126:

This standard deals with the following aspects to determine the quality of a software application:

- Quality model
- External metrics
- Internal metrics
- Quality in use metrics

This standard presents some set of quality attributes for any software such as:

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

❖ ISO/IEC 9241-11:

Part 11 of this standard deals with the extent to which a product can be used by specified users to achieve specified goals with Effectiveness, Efficiency and Satisfaction in a specified context of use.

This standard proposed a framework that describes the usability components and the relationship between them. In this standard, the usability is considered in terms of user performance and satisfaction. According to ISO 9241-11, usability depends on context of use and the level of usability will change as the context changes.

❖ ISO/IEC 25000:2005:

ISO/IEC 25000:2005 is commonly known as the standard that provides the guidelines for Software Quality Requirements and Evaluation (SQuaRE). This standard helps in organizing and enhancing the process related to software quality requirements and their evaluations. In reality, ISO-25000 replaces the two old ISO standards, i.e. ISO-9126 and ISO-14598.

SQuaRE is divided into sub-parts such as:

- ISO 2500n - Quality Management Division
- ISO 2501n - Quality Model Division
- ISO 2502n - Quality Measurement Division
- ISO 2503n - Quality Requirements Division
- ISO 2504n - Quality Evaluation Division

The main contents of SQuaRE are:

- Terms and definitions
- Reference Models
- General guide
- Individual division guides

- Standard related to Requirement Engineering (i.e. specification, planning, measurement and evaluation process)

❖ **ISO/IEC 12119:**

This standard deals with software packages delivered to the client. It does not focus or deal with the clients' production process. The main contents are related to the following items:

- Set of requirements for software packages.
- Instructions for testing a delivered software package against the specified requirements.

2. Software Engineering Institute:

Software Engineering Institute at Carnegie-Mellon University; initiated by the U.S. Defense Department to help improve software development processes.

CMM = 'Capability Maturity Model', developed by the SEI. It's a model of 5 levels of organizational 'maturity' that determine effectiveness in delivering quality software. It is geared to large organizations such as large U.S. Defense Department contractors. However, many of the QA processes involved are appropriate to any organization, and if reasonably applied can be helpful. Organizations can receive CMM ratings by undergoing assessments by qualified auditors.

Level 1: characterized by chaos, periodic panics, and heroic efforts required by individuals to successfully complete projects. Few if any processes in place; successes may not be repeatable.

Level 2: software project tracking, requirements management, realistic planning, and configuration management processes are in place; successful practices can be repeated.

Level 3: standard software development and maintenance processes are integrated throughout an organization; a Software Engineering Process Group is in place to oversee software processes, and training programs are used to ensure understanding and compliance.

Level 4: metrics are used to track productivity, processes, and products. Project performance is predictable, and quality is consistently high.

Level 5: the focus is on continuous process improvement. The impact of new processes and technologies can be predicted and effectively implemented when required.

Software Review:

Software review is the meeting conducted by technical people for technical people. It can thus be defined as technical assessment of a work product created during the software engineering process. It is a basis for SQA mechanism.

In other words, software review is the way of using the diversity of a group of people to:

- Point out needed improvement in the software.
- Confirm those portions of the software in which improvement is either not desired or not needed.
- Conforms for rework.

TYPES OF SOFTWARE REVIEW:

a. Informal Review:

Conducted on an as-needed i.e. informal agenda. The date and the time of the agenda for the Informal Review will not be addressed in the Project Plan. The developer chooses a review panel and provides and/or presents the material to be reviewed. The material may be as informal as a computer listing or hand-written documentation.

b. Formal Review:

Conducted at the end of each life cycle phase i.e. Formal Agenda. The agenda for the formal review must be addressed in the Project Plan. The acquirer of the software appoints the formal review panel or board, who may make or affect a go/no-go decision to proceed to the next step of the life cycle. The material must be well prepared *Example:* Formal reviews include the Software Requirements Review, the Software Preliminary Design Review, the Software Critical Design Review, and the Software Test Readiness Review.

For conducting Formal Technical Review (FTR) we can follow the following guidelines:

1. Review the product, not the producer.
2. Set an agenda and maintain it.
3. Limit debate.
4. Enunciate problem areas, but does not attempt to solve every problem noted.
5. Take written notes.
6. Limit the number of participants and insist upon advance preparation.
7. Schedule review as project tasks.
8. Review your early reviews.
9. Record and report all review results.

COST IMPACT OF SOFTWARE DEFECTS:

Since the defect/ fault/ error imply a quality problem that is discovered by software engineers (or others) before the software is released to the end user (or to another framework activity to the software process).

The primary objective of technical review is to find errors during the process so that they do not become defected after release of the software. The clear benefits of technical review is the early discovery of errors so that they do not propagate to the next step in software process.

A number of industry studies indicate that design activities introduce between 50% and 65% of all errors during software process. However, review technique have been shown up to 75% effective in covering design flows.

DEFECT AMPLIFICATION AND REMOVAL:

A defect amplification model can be used to illustrate the generation and detection of errors during the design and code generation actions of a software.

Development step

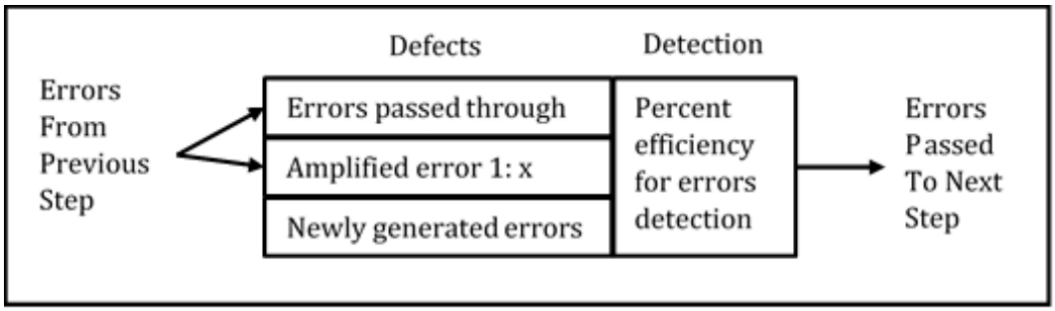


Fig: Defect Amplification Model

The box represents a software engineering action. During the action, errors may be introduced. Review may fail to uncover newly generated errors and errors from previous step resulting in some numbers of errors that are passed through. In some case, errors from previous steps are amplified (by amplification factor x) by current work.

The box sub-divisions represents each of these characteristics and the percent of efficiency for detecting errors a function of thoroughness of review.

FORMAL TECHNICAL REVIEW (FTR):

A FTR is a software quality control activity performed by software engineers and others. The objectives of FTR are:

1. To uncover errors in function, logic or implementation for any representation of the software.
2. To verify that the software under review meets its requirements.
3. To ensure that software has been represented according to predefined standards.
4. To achieve software that is developed in a uniform manner.
5. To make projects more manageable.

In addition, the FTR server as a training ground, enabling junior engineers to observe different approaches to software analysis, design and implementation. Each FTR is conducted as a meeting and will be successful only if it is properly planned, controlled and attended.

THE REVIEW MEETING:

Every review meeting should abide by the following constraints:

- Between 3 to 5 people (typically) should be involved in the review.
- Advance preparation should occur but should require no more than two hours of work for each person.
- The duration of review meeting should be less than two hours.

Given these constraints, it should be clear that an FTR focuses on a specific (and small) part of overall software, so that the FTR has a higher likelihood of uncovering errors. The focus of FTR

is on a work product (example: a portion of requirement modes, a detailed component design source code, etc.). The individual who has developed the work product (producer) informs the project leader that the product is complete and review is required.

The project leaders contacts a review leader, who evaluates the product for readiness, generates copies of product material and distribute them to reviewers for advance preparation. Each review is expected to spend 1 to 2 hours, reviewing the product, making notes, and otherwise become familiar to work. Concurrently, the review leader also reviews the product and establishes an agenda for the review meeting, which is typically scheduled for next day.

The review meeting is attended by the review leader and all the reviewers and the producer. One of the reviewer takes on the role of recorder i.e. who records (in writing) all important issues raised during review.

The FTR begins with the introduction of the agenda and a brief introduction by the producer. When valid problems or errors are discovered the recorder note each. At the end of review, all attendees of the FTR must decide whether to:

- Accept the product without further modifications.
- Reject the product due to server errors (once corrected another review must be performed).
- Accept the product provisionally (minor errors have been encountered and must be correct, but no additional review will be required)

After complete, all reviewer needs to sign off.

REVIEW REPORTING AND RECORD KEEPING:

During the FTR, a reviewer (the recorder) actively records all issues that have been raised. These are summarized at the end of the review meeting, and the review issues list is produced. In addition a FTR summary report is completed. A review summary report answers three questions:

1. What was reviewed?
2. Who reviewed it?
3. What were the findings and the conclusions?

The review summary report is single page form (with possible attachments). It becomes part of the project historical record and may be distributed to the project leader and other interested parties. The review issue list provides/serves two purposes:

i. To identify problem areas within the product and
ii. To serve as an action item check list that guides the producer as corrections are made. An issue list is normally attached to the summary report.

FORMAL APPROACHES TO SQA:

Software quality can be achieved through competent software engineering practice as well as through the application of technical review, testing strategies, better control of software work products and the changes made to them, standard accepted, etc. But software engineering community has argued that a more formal approaches to SQA is required.

It can be argued that a computer program is a mathematical object. A syntax and semantics can be defined for every programming language and approach to specification of software requirements is available. So, different approaches are necessary for Software Quality Assurance.

PROOF OF CORRECTNESS:

- Performed using verification and validation in static way.
- Using testing strategies in dynamic aspects.
- Comparing output deviations.
- Follow of standards.

STATISTICAL SOFTWARE QUALITY ASSURANCE:

Statistical SQA reflects the growing trend throughout industry to become more quantitative about quality. Statistical quality assurance for software implies the following steps:

- Information about software errors and defects is collected and categorized.
- An attempt is made to trace each error, and defect to its underlying cause.
- Using the pareto principle (80% of the defects can be trace to 20% of possible causes) isolate the 20% (vital few).
- Once the vital few causes have been identified move to correct the problems that have caused the errors and defects.

Example:

Error	**Total Number**	**%**	**Serious Number**	**%**	**Moderate Number**	**%**	**Minor Number**	**%**
IES	205	22	34	27	68	18	103	24
MCC	156	17	12	9	68	18	76	17
IDS	48	5	1	1	24	6	23	5
VPS	25	3	0	0	15	4	10	2
EDR	130	14	26	20	68	18	36	8
ICI	58	6	9	7	18	5	31	7
EDL	45	5	14	11	12	3	19	4
IET	35	10	12	9	35	9	48	11
IID	36	4	2	2	20	5	14	3
PLT	60	6	15	12	19	5	26	6
HCI	28	3	3	2	17	4	8	2
MIS	56	6	0	0	15	4	41	9

Total	**942**	**100**	**128**	**100**	**379**	**100**	**435**	**100**

- IES: Incomplete or Erroneous Specifications
- MCC: Misinterpretation of Customer Communication
- IDS: Intentional Deviation form Specs
- VPS: Violation of Programming Standard
- EDR: Error in Data Representation
- EDL: Error in Design Logic

CLEAN ROOM ENGINEERING:

- It is a software development philosophy that is based on avoiding software defects by using formal methods of development and a rigorous inspection process.
- The name "clean room" was derived by analogy with semiconductor fabrication units. In these units, defects are avoided by manufacturing in an ultra-clean atmosphere.
- The objective of this approach is zero-defect software development.
- The clean room approach to software development is based on five keys strategies:
  a. Formal Specification
  b. Incremental Development
  c. Structured Programming
  d. Static Verification
  e. Statistical Testing of System.

Chapter – 5

Software Reliability

INTRODUCTION:

Software reliability is one of the most important element of the overall quality of any software even after accomplishment of a software work. If it fails to meet its actual performance after its deployment, then the software is considered as unreliable software. We cannot expect better performance from such software.

Software reliability is defined in statistical terms as the probability of failure free operation of a computer program in a specific environment for specific time for a software to be reliable, it must performs operation based upon its analysis and design, available resources, reusability (if reusable components are involved) and so on.

If the design standards are not maintained, efficient algorithms are not implemented, necessary resources are not available or the supply is not at proper times, then there is high probability of software failure or there is degradation in quality of performance.

Therefore, in order to make a software reliable, we should take some measures or earlier stages as follows:

- Designing software project based on the available resources.
- Develop software that best fits the current environment conditions.
- Estimating costs that might be required even after its implementation.
- Estimating the actual performance upon using reusable resources.

SOFTWARE SAFETY:

Software safety is a SQA activity that focuses on the identification and assessment of potential hazards that may affect software negatively and cause an entire system to fail. If hazards can be identified early in the software process, software design features can be specified that will either eliminate or control potential hazards.

A modeling and analysis process is conducted as part of software safety. Initially, hazards are identified and categorized by critically and risk. For example: some of the hazards associated with a computer base cruise control for an automobile might be:

- Causes uncontrolled acceleration that cannot be stopped.
- Does not respond to depression of brake pedal (by turning off)
- Does not engage when switch is activated.
- Slowly loses or gain speed.

Once such hazards are identified analysis techniques are used to assign severity and probability of occurrence.

Software reliability uses statistical analysis to determine the likelihood that a software failure will cause. However, the occurrence of failure does not necessarily result in a mishap (accident).

Software safety examines the ways in which failure result in conditions that can lead to a mishap. That is, failures are not considered in vacuum but are evaluated in the context of an entire computer based system and its environment.

MEASURES OF RELIABILITY AND AVAILABILITY:

If we consider a computer based system, a simple measure of reliability is meantime between failures.

MTBF = MTTF + MTTR

Availability = $\frac{MTTF}{MTTF+MTTR} * 100\%$

- MTBF = Meantime Between Failure
- MTTF = Meantime to Failure
- MTTR = Meantime to Repair

SOFTWARE RELIABILITY MODELS:

The basic goal of software engineering is to produce high quality software at low cost. With growth in size and complexity of software, management issues began dominating. Reliability is one of the representative qualities of software development process. Reliability of software is basically defined as the probability of expected operation over specified time interval.

Software Reliability is an important attribute of software quality, together with functionality, usability, performance, serviceability, capability, install ability, maintainability, and documentation. Reliability is the property of referring 'how well software meets its requirements' & also 'the probability of failure free operation for the specified period of time in a specified environment. Software reliability defines as the failure free operation of computer program in a specified environment for a specified time. Software Reliability is hard to achieve, because the complexity of software tends to be high.

A proliferation of software reliability models have emerged as people try to understand the characteristics of how and why software fails, and try to quantify software reliability. Software Reliability Modelling techniques can be divided into two subcategories: Prediction modelling and Estimation modelling. Both kinds of modelling techniques are based on observing and accumulating failure data and analyzing with statistical inference.

1. PREDICTION MODELS:

This model uses historical data. They analyze previous data and some observations. They are usually made prior to development and regular test phases. The model follow the concept phase and the predication from the future time.

a. Musa Model:

This prediction technique is used to predict, prior to system testing, what the failure rate will be at the start of system testing. This prediction can then later be used in the reliability growth

modelling. For this prediction method, it is assumed that the only thing known about the hypothetical program is a prediction of its size and the processor speed.

This model assumes that failure rate of the software is a function of the number of faults it contains and the operational profile. The number of faults is determined by multiplying the number of developed executable source instructions by the fault density. Developed excludes re-used code that is already debugged. Executable excludes data declarations and compiler directives. For fault density at the start of system test, a value for faults per KSLOC needs to be determined. For most projects the value ranges between 1 and 10 faults per KSLOC. Some developments which use rigorous methods and highly advanced software development processes may be able to reduce this value to 0.1 fault/KSLOC.

b. <u>Putnam Model:</u>

Created by Lawrence Putnam, Sr. the Putnam model describes the time and effort required to finish a software project of specified size. SLIM (Software Lifecycle Management) is the name given by Putnam to the proprietary suite of tools his company QSM, Inc. has developed based on his model. It is one of the earliest of these types of models developed, and is among the most widely used. Closely related software parametric models are Constructive Cost Model (COCOMO), Parametric Review of Information for Costing and Evaluation – Software (PRICE-S), and Software Evaluation and Estimation of Resources – Software Estimating Model (SEER-SEM).

Putnam's observations about productivity levels to derive the software equation:

$$\frac{B^{1/3} \cdot \text{Size}}{\text{Productivity}} = \text{Effort}^{1/3} \cdot \text{Time}^{4/3}$$

Where:

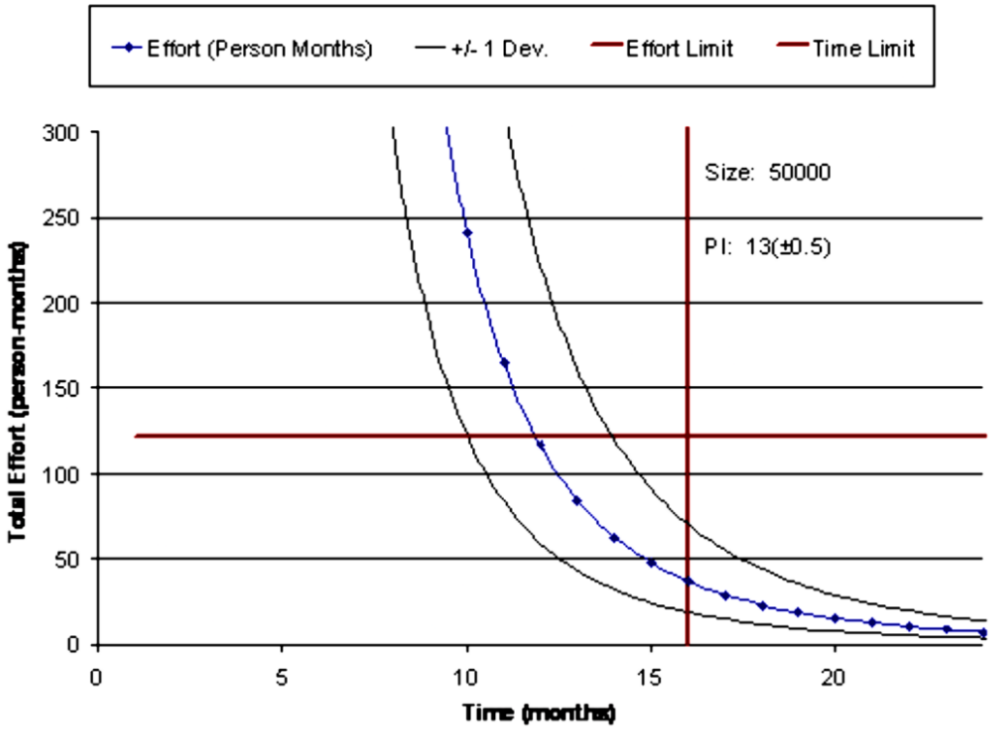
- Size is the product size (whatever size estimate is used by your organization is appropriate). Putnam uses ESLOC (Effective Source Lines of Code) throughout his books.
- B is a scaling factor and is a function of the project size Productivity is the Process Productivity, the ability of a particular software organization to produce software of a given size at a particular defect rate.
- Effort is the total effort applied to the project in person-years.
- Time is the total schedule of the project in years.

In practical use, when making an estimate for a software task the software equation is solved for *effort*:

$$\text{Effort} = \left[\frac{\text{Size}}{\text{Productivity} \cdot \text{Time}^{4/3}}\right]^3 \cdot B$$

An estimated software size at project completion and organizational process productivity is used. Plotting *effort* as a function of *time* yields the *Time-Effort Curve*. The points along the curve represent the estimated total effort to complete the project at some *time*. One of the distinguishing features of the Putnam model is that total effort decreases as the time to complete

the project is extended. This is normally represented in other parametric models with a schedule relaxation parameter.



Thus estimating method is fairly sensitive to uncertainty in both size and process productivity. Putnam advocates obtaining process productivity by calibration.

c. Rome Lab TR-92-52 Model:

Software Reliability Measurement and Test Integration Techniques Method RL-TR-92-52 contain empirical data that was collected from a variety of sources, including the Software Engineering Laboratory. The model consists of 9 factors that are used to predict the fault density of the software application. The nine factors are:

Factor	Measure	Range of values	Applicable Phase*	Tradeoff Range
A – Application	Difficulty in developing various application types	2 to 14 (defects/KSLOC)	A-T	None – fixed
D - Development organization	Development organization, methods, tools, techniques, documentation	.5 to 2.0	If known at A, D-T	The largest range
SA - Software anomaly management	Indication of fault tolerant design	.9 to 1.1	Normally, C-T	Small
ST - Software	Traceability of design and code to requirements	.9 to 1.0	Normally, C-T	Large
SQ - Software quality	Adherence to coding standards	1.0 to 1.1	Normally, C-T	Small
SL - Software	Normalizes fault density by language type	Not applicable	C-T	N/A
SX - Software	Unit complexity	.8 to 1.5	C-T	Large
SM - Software	Unit size	.9 to 2.0	C-T	Large
SR - Software standards review	Compliance with design rules	.75 to 1.5	C-T	Large

There are certain parameters in this prediction model that have tradeoff capability. This means that there is a large difference between the maximum and minimum predicted values for that particular factor. Performing a tradeoff means that the analyst determines where some changes can be made in the software engineering process or product to experience an improved fault density prediction. A tradeoff is valuable only if the analyst has the capability to impact the software development process. The output of this model is a fault density in terms of faults per KSLOC. This can be used to compute the total estimated number of inherent defects by simply multiplying by the total predicted number of KSLOC.

d. Rayleigh Model:

This model predicts fault detection over the life of the software development effort and can be used in conjunction with the other prediction techniques. Software management may use this profile to gauge the defect status. This model assumes that over the life of the project that the faults detected per month will resemble a Raleigh curve.

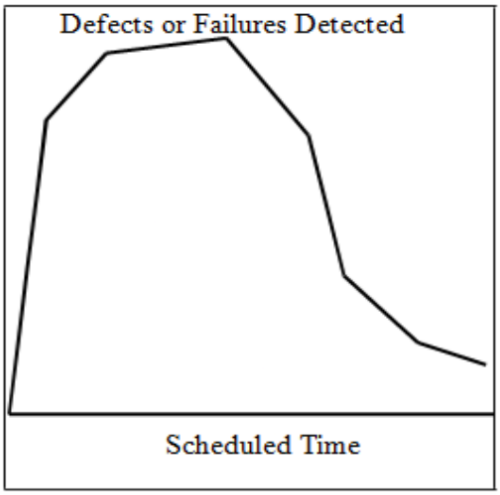
The Rayleigh model is the best suited model for estimating the number of defects logged throughout the testing process, depending on the stage when it was found (unit testing, integration testing, system testing, functional testing etc.). Defect prediction function is the following:

$$F(t) f(K, t, t) m = (1)$$

Parameter K is the cumulated defect density, t is the actual time unit and tm is the time at the peak of the curve.

For example, the estimated number of defects impacts the height of the curve while the schedule impacts the length of the curve. If the actual defect curve is significantly different from the

predicted curve then one or both of these parameters may have been estimated incorrectly and should be brought to the attention of management.



2. ESTIMATION MODEL:

This model uses the current data from the current software development effort and doesn't use the conceptual development phases and can estimate at any time.

a. Weibull Model (WM):

This model is used for software/hardware reliability. The model incorporates both increasing/decreasing and failure rate due to high flexibility. This model is a finite failure model.

$$MTTF = \int (1-F(t)) = \int \exp(-\beta t^{\alpha}) \qquad (4)$$

Where, MTTF = Mean Time to Failure

α, β = Weibull distribution parameters. t=Time of failure.

A Weibull-type testing-effort function with multiple change-points can be estimated by using the methods of least squares estimation (LSE) and maximum likelihood estimation (MLE) The LSE minimizes the sum of squares of the derivations between actual data patterns and predicted curve, while the MLE estimates parameters by solving a set of simultaneous equations.

b. Bayesian Models (BM):

The models are used by several organizations like Motorola, Siemens & Philips for predicting reliability of software. BM incorporates past and current data. Prediction is done on the bases of number of faults that have been found & the amount of failure free operations. The Bayesian SRGM considers reliability growth in the context of both the number of faults that have been detected and the failure-free operation. Further, in the absence of failure data, Bayesian models consider that the model parameters have a prior distribution, which reflects judgment on the unknown data based on history e.g. a prior version and perhaps expert opinion about the software.

c. J -M Model (JMM):

This model assumes that failure time is proportional to the remaining faults and taken as an exponential distribution. During testing phase the number of failures at first is finite. Concurrent mitigation of errors is the main strength of the model and error does not affect the remaining errors. Error removal is all human behavior which is irregular so it cannot be avoid by introducing new errors during the process of error removal.

$$(MTBF)\, t = 1/(N-(i-1)) \qquad (6)$$

Where, N= Total number of faults.

i= Number of fault occurrences. MTBF=Mean Time between failure.

t= Time between the occurrence of the $(i-1)^{st}$ and i^{th} fault occurrences.

This model assumes that N initial faults in the code prior to testing are a fixed but known value. Failures are not correlated and the times between failures are independent and exponentially distributed random variables. Fault removal on failure occurrences is instantaneous and does not introduce any new faults into the software under test. The hazard rate z(t) of each fault is time invariant and a constant. Moreover, each fault is equally likely to cause a failure.

d. Goel-Okumoto Model (GOM):

G-O model takes the number of faults per unit time as independent random variables. In this model the number of faults occurred within the time and model estimates the failure time. Delivery of software within cost estimates is also decided by this model.

The Goel-Okumoto (G-O) non-homogeneous Poisson process (NHPP) model has slightly different assumptions from the J-M model. The significant difference between the two is the assumption that the expected number of failures observed by time t follows a Poisson distribution with a bounded and non-decreasing mean value function (t). The expected value of the total number of failures observed in infinite time is a finite value N.

e. Thompson and Chelson's Model:

In this model the Number of failures detected in each interval (fi) and Length of testing time for each interval i (Ti).

$$(fi + f0 + 1)/$$

$$(Ti+T0)$$

Software is corrected at end of testing interval. Software is operational. Software is relatively fault free. The Thompson Chelson model can be used, If the failure intensity is decreasing, has the software been tested or used in an operational environment representative of its end usage, with no failures for a significant period of time.

Chapter – 6

Management of Software Engineering

Responsibilities of a software project manager:

Proper project management is essential for the successful completion of a software project and the person who is responsible for it is called project manager. To do his job effectively, the project manager must have certain set of skills.

➢ Job Responsibilities of a Software Project Manager:

1. Involves with the senior managers in 'the process of appointing team members
2. Builds the project team and assigns tasks to various team members
3. Responsible for effective project planning and scheduling, project monitoring and control activities in order to achieve the project objectives
4. Acts as a communicator between the senior management and the other persons involved in the project like the development team and internal and external stakeholders
5. Effectively resolves issues (if any) that arise between the team members by changing their roles and responsibilities
6. Modifies the project plan (if required) to deal with the situation.

➢ Skills Necessary for Software Project Management:

Although the actual skills for effective project management develop with experience, every project manager must exhibit some basic skills that are listed below.

1. Must have the knowledge of different project management techniques like risk management, configuration management, cost estimation techniques, etc.
2. Must have the ability to make judgment, since project management frequently requires making decisions.
3. Must have good grasping power to learn the latest technologies to adapt to project requirements.
4. Should be open-minded enough to accept new ideas from the project members. In addition, he should be creative enough to come up with new ideas.
5. Should have good interpersonal, communication, and leadership qualities in order to get work done from the team members.

Project Planning:

Before a project begins, the manager and the software development team must make a proper plan. The project planning involves estimation of the work to be done, the resources that will be required, the effort and money to be invested and the time to be complete (i.e. from start to the completion of the project).

The software planning begins with determining the software scope. Before the software scope could be established, the function and performance of the software should be well understood

and well defined. The software scope describes the data and control to be processed function, performance, constraints and reliability.

Once the project scope has been well defined, it is necessary to study the feasibility of the software or project. By software feasibility, it means that if we can build the software to meet the defined scope, the software feasibility can be defined for the technology used, financial support, total time required and resources needed for the development of the software.

The success of any software or project is based on proper planning. After planning, we will be able to track or monitor the project in course of time.

- **THE SPMP DOCUMENT:**

SPMP stands for Software Project Management Plan. Document used for proper project management includes:

1. **Introduction:**
   a. Project Overview
   b. Project Deliverables
      - Preliminary Project Plan
      - Requirement Specification
      - Analysis [Object Model, Dynamic Model and User Interface]
      - Architectural Specification
      - Component/object Specification
      - Source code
      - Test Plan
      - Final Product/Demo

2. **Project Organization:**
   a. Process Model [Waterfall, Spiral, Prototype, etc.]
   b. Organization Structure
      - Team Members
      - WBS [Work Breakdown Structure]
      - Deliverable per week/month

3. **Managerial Process:**
   a. Management Objectives and Priorities
   b. Assumptions, Dependencies and Constraints
   c. Risk Management
      - Risk Identification
      - Risk Mitigation
   d. Monitoring and Controlling Mechanisms

4. **Technical Process:**
   a. Methods/tools and techniques
   b. Software Documentations, etc.

5. **Work Elements, Schedule and Budget:**

METRICS FOR PROJECT SIZE ESTIMATION:

Concerned with the cost associated with the project, the software considered to constitute very small cost than overall computer based system cost. But as the system become more advance, the cost of software has contributed a lot to the overall cost of the project.

The inappropriate estimation might lead to conditions that are disastrous and the developer has to bear a loss. Since, any project has its association with human, technical, environment, political aspects, remarkable estimation can be done only considering these aspects.

A general project estimation can be done by:

1. Delaying estimation until late in the project.
2. Estimating projects based upon similar past projects.
3. Using relatively simple decomposition technique.
4. Using one or more empirical models for cost and effort estimations.

The first two techniques '1' and '2' are not efficient because estimation should be done as early as possible and that not all past experiences resemble the same criterion and might not fulfill the requirement of the current project.

The technique '3' is a conventional estimation technique because it follows the "divide and conquer".

The technique '4' is the empirical estimation model, it can be used to complement the decomposition technique.

- **APPROACHES FOR PROJECT SIZE ESTIMATION:**

1. LOC Approach:

Consider a software package to be developed for a CAD application for mechanical components. The software needs to have interface with various computer graphics peripherals including a mouse, digitizer, high-resolution color display and laser printer. It will need to accept 2 or 3 dimensional data. CAD database need to be maintain. Design and analysis module need to be developed to produce the required output, which will be displayed on display devices. The software will be designed to control and interact with peripheral devices.

- UICF = User Interface and Control Facilities
- 2DGA = Two Dimension Geometric Analysis
- 3DGA = Three Dimension Geometric Analysis
- DBM = Database Management
- CGDF = Computer Graphics Display Facilities
- PCF = Peripheral Control Function
- DAM = Design Analysis Module

Estimation Table for LOC Method

Function	Estimated LOC (Lines of Code)
UICF	2300
2DGA	5300
3DGA	6800

DBM	3350
CGDF	4950
PCF	2100
DAM	8400
Total LOC	**33200**

Total LOC for given CAD application = 33,200

A review of historical data indicates that the organizational average productivity for systems of this type is 620 LOC per month, based on the labor rate of $8000 per month and the cost per line of code is approximately (8000/620 = $12.9 = $13). So, based on LOC estimate the total estimated project cost is ($13 * 33200 = $4,33,000) and the estimated effort is (33200/620 = 53.54 = 54 persons)

2. **Function Point Metrics Approach:**

Function Point Metric is indirect measure to estimate size. This approach can be used effectively as a means for measuring the functionality delivered by a system. Using historical data, the Function Point Metrics Approach can be used to:

a. Estimate the cost or effort required to design, code and test the software.
b. Predict the number of errors that will be encountered during the testing.
c. Forecasts the number of components and/or the number of projected sources lines in the implemented system.

Function Point Metrics are derived using an empirical relationship based on countable (direct) measures of software's information domain and qualitative assessment of software complexity. Information domain values are defined in the following manner:

- Number of external inputs (EIs) from user
- Number of external outputs (EOs) by system
- Number of external inquires (EQs) for the system
- Number of internal logical files (ILFs) used in system
- Number of external interface files (EIFs) used in system

Once these data have been collected, we use this approach. To compute Function Point Metrics we use the following relationship:

- FP = Count *(sum of all FP entries)* total * (0.65+0.01*E(Fi))
- Fi = 0 – 14

Example:

Measurement Parameters	Count	Weight	Count * Weight
Number of user inputs	40	4	160
Number of user outputs	25	5	125
Number of user inquires	12	4	48
Number of logical files	4	7	28
Number of external interfaces	4	7	28
Count Total			389

Now, FP can be calculated as:

FP = count total * (0.65 + 0.01 * ∑ (Fi))

Where, Fi is the complexity adjustment factor based on response to various questions like:

- Does the system require reliable backup?
- Are data communication required?
- Is performance critical?
- Are the master's file updated online?

PROJECT ESTIMATION TECHNIQUES:

We discussed various parameters involving project estimation such as size, effort, time and cost. Project manager can estimate the listed factors using some techniques such as:

1. EMPIRICAL ESTIMATION TECHNIQUE:

It is based on making an educated guess of the project parameters. While using this technique, prior experience with development of similar products is helpful. Although empirical estimation techniques are based on common sense, different activities involved in estimation have been formalized over the years. Two popular empirical estimation techniques are: Expert judgment technique and Delphi cost estimation.

This technique uses empirically derived formulae to make estimation. These formulae are based on LOC or FPs. The general form is:

Effort = Tuning Coefficient * (size) $^{\text{Exponent}}$

Here,

- Effort = Usually derived as person month of effort required
- Tuning Coefficient = Either a constant or a number derived based on complexity of project
- Size = usually LOC or FP
- Exponent = empirically derived
- i.e. $E = A + B\ (ev)^{C}$,
  A, B, C are constant, empirically derived

Some LOC oriented Approach are:

a. Walston Felix Model
   $E = 5.2 * (KLOC)^{0.91}$

b. Bailey Basili Model
   $E = 5.5 + 0.73 + (KLOC)^{1.16}$

Some Function Point oriented Approach are:

a. Albrecht and Gattney Model
   $E = 13.39 + 0.545 * FP$

b. Matson Barnett and Mellichamp Model

E = 585.7 + 150.12 * FP

- **Expert Judgment Technique:**

This technique is widely used for cost estimation. It is top-down estimation technique i.e. it first focuses on the system level costs such as computing resources and personnel required to develop the system as well as the costs of configuration management, quality assurance, system integration, training and publications. Personnel costs are estimated by examining the cost of similar past projects.

Expert judgment relies on the experience, background and business sense of one or more key people (experts) in the organization. An expert might arrive at a cost estimate in the following measures:

Example:

Consider a system to be developed is a process control system similar to one that was developed last 10 years in 10 months of a cost of $ 1 million. The new system has similar control functions, but has 25% more activities to control, thus we will increase our time and cost estimates by 25%. The previous system was the first of its type that we develop however we will use the same computer and external sensing/controlling devices and many of the same people are available to develop the new system. So, we can reduce our estimate by 20%.

Furthermore, we can reuse much of the low-level code from the previous product, which reduce the time and cost estimate by 25%. The net effect of these consideration is a time and test reduction of 20% which result is an estimate of $ 800,000 and 8 months development time. We know that the customer has budgeted $ 1 million and 1 year delivery time for system. Therefore, we add a small margin of safety and bid the system at $ 850,000 and 9 months development time.

- **Delphi Cost Estimation Technique:**

The Delphi estimation technique can be adapted to software cost estimation in the following manner:

- A coordinator provides each estimator with the system definition document and a form for recording a cost estimate.
- Estimators study the definition and complete their estimates anonymously. They may ask questions of the coordinator but they do not discuss their estimates with one another.
- The coordinator prepares and distributes a summary of the estimator's response and includes any usual rationales (reasons and principle on which a decision is based) noted by the estimators.
- Estimator complete another estimate using the results from the previous estimate
- Estimators whose estimates differ sharply from the group may be asked to provide justification for their estimates.
- The process is iterated for as many rounds as required. No group discussion is allowed during the entire process.

It is possible that several round of estimates will not lead to a consequence estimate. In this case, the coordinator must discuss the issues involved with each estimator to determine the reasons

for the differences. The coordinator may have to gather additional information and present it to the estimators in order to resolve the difference in view point.

2. Heuristic Technique:

Heuristic techniques assume that the relationships among the different project parameters can be modeled using suitable mathematical expressions. Once the basic (independent) parameters are known, the other (dependent) parameters can be easily determined by substituting the value of the basic parameters in the mathematical expression. Different heuristic estimation models can be divided into the following two classes: single variable model and the multi variable model.

Single variable estimation models provide a means to estimate the desired characteristics of a problem, using some previously estimated basic (independent) characteristic of the software product such as its size. A single variable estimation model takes the following form:

$$\textit{Estimated Parameter} = c_1 * e^{d_1}$$

In the above expression, e is the characteristic of the software which has already been estimated (independent variable). *Estimated Parameter* is the dependent parameter to be estimated. The dependent parameter to be estimated could be effort, project duration, staff size, etc. c_1 and d_1 are constants. The values of the constants c_1 and d_1 are usually determined using data collected from past projects (historical data). The basic COCOMO model is an example of single variable cost estimation model.

A multivariable cost estimation model takes the following form:

$$\textit{Estimated Resource} = c_1 * e_1^{d_1} + c_2 * e_2^{d_2} + \ldots$$

Where e_1, e_2, … are the basic (independent) characteristics of the software already estimated, and c_1, c_2, d_1, d_2, … are constants. Multivariable estimation models are expected to give more accurate estimates compared to the single variable models, since a project parameter is typically influenced by several independent parameters. The independent parameters influence the dependent parameter to different extents. This is modeled by the constants c_1, c_2, d_1, d_2, … . Values of these constants are usually determined from historical data. The intermediate COCOMO model can be considered to be an example of a multivariable estimation model.

- **COCOMO Model:**

It stands for Constructive Cost Model and is proposed by Barry Boehm. Barry Boehm postulates that any software development can be classified into three categories based on the development complexity.

1. **Organic:**

It deals with well understood application program, size of development team is reasonably small, and team members are experienced in developing similar type of projects.

2. **Semi-Detached:**

Project team consists of a mixture of experienced and inexperienced staff. Team members have limited experience on related systems and may be unfamiliar with some aspects of system being developed.

3. Embedded:

The software is strongly coupled to complex hardware, or real-time systems. They have strict procedure and operation with more than 300 KLOC.

☑ Basic COCOMO Model:

Gives only an approximate estimation:

- Effort = $a_1(KLOC)^{a2}$
- $T_{dev}=b_1 (Effort)^{b2}$
  - ✓ KLOC is theestimated kilolines of source code,
  - ✓ $a_1a_2b_1b_2$ are constants for different categories of software products,
  - ✓ T_{dev} is the estimated time to develop the software in months,
  - ✓ Effort estimation is obtained in terms of person months (PMs).

Development Effort Estimation

Organic:

- ❖ Effort = $2.4 (KLOC)^{1.05}$ PM

Semi-detached:

- ❖ Effort = $3.0(KLOC)^{1.12}$ PM

Embedded:

- ❖ Effort = $3.6 (KLOC)^{1.20}$ PM

Development Time Estimation

Organic:

- ❖ Tdev = $2.5 (Effort)^{0.38}$ Months

Semi-detached:

- ❖ Tdev = $2.5 (Effort)^{0.35}$ Months

Embedded:

- ❖ Tdev = $2.5 (Effort)^{0.32}$ Months

3. Analytical Estimation technique:

It derives the required results starting with basic assumptions regarding the project. Thus, unlike empirical and heuristic techniques, analytical techniques do have scientific basis, Halstead's software science is an example of an analytical technique.

It can be used to derive some interesting results starting with a few simple assumptions. It is especially useful for estimating software maintenance efforts. In fact, it outperforms both empirical and heuristic techniques when used for predicting software maintenance efforts.

Halstead's Software Science – An Analytical Technique

Halstead's software science is an analytical technique to measure size, development effort, and development cost of software products. Halstead used a few primitive program parameters to develop the expressions for overall program length, potential minimum value, actual volume, effort, and development time.

For a given program, let:

- ☑ η_1 be the number of unique operators used in the program,
- ☑ η_2 be the number of unique operands used in the program,
- ☑ N_1 be the total number of operators used in the program,
- ☑ N_2 be the total number of operands used in the program.

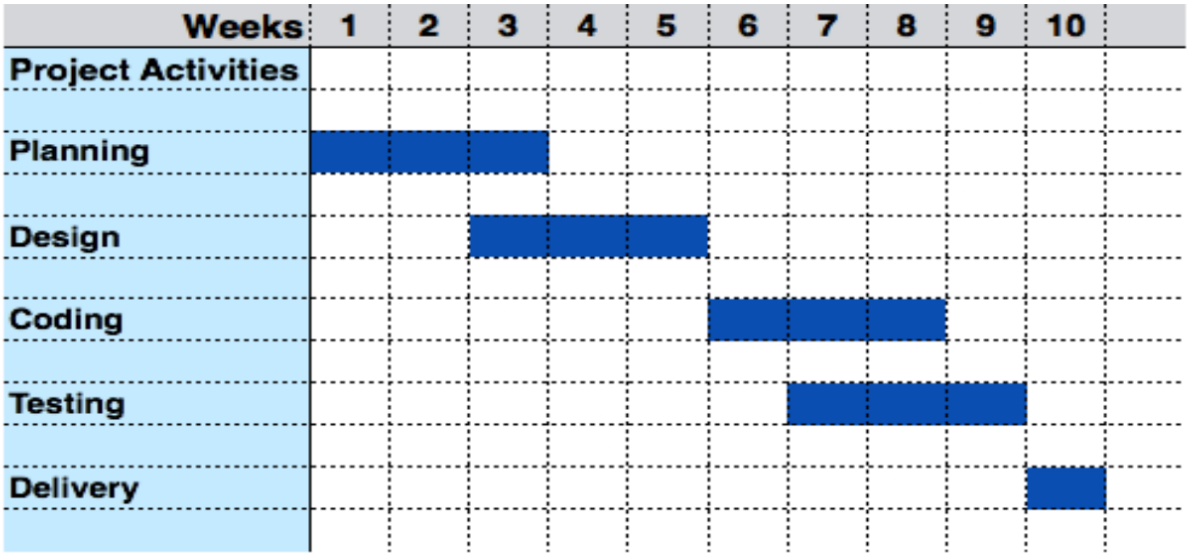
PROJECT SCHEDULING:

Project Scheduling in a project refers to roadmap of all activities to be done with specified order and within time slot allotted to each activity. Project managers tend to define various tasks, and project milestones and they arrange them keeping various factors in mind. They look for tasks lie in critical path in the schedule, which are necessary to complete in specific manner (because of task interdependency) and strictly within the time allocated. Arrangement of tasks which lies out of critical path are less likely to impact over all schedule of the project.

For scheduling a project, it is necessary to:

- ☑ Break down the project tasks into smaller, manageable form
- ☑ Find out various tasks and correlate them
- ☑ Estimate time frame required for each task
- ☑ Divide time into work-units
- ☑ Assign adequate number of work-units for each task
- ☑ Calculate total time required for the project from start to finish

For project scheduling, creating timeline chart is an efficient option.



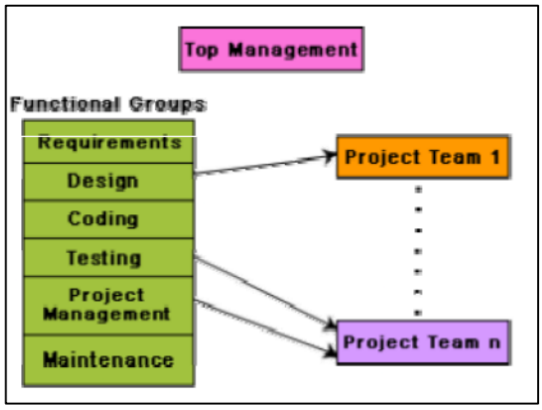
ORGANIZATION AND TEAM STRUCTURE:

1. Organization Structure:

Usually every software development organization handles several projects at any time. Software organizations assign different teams of engineers to handle different software projects. Each type of organization structure has its own advantages and disadvantages so the issue "how is the organization as a whole structured?" must be taken into consideration so that each software project can be finished before its deadline.

There are essentially two broad ways in which a software development organization can be structured:

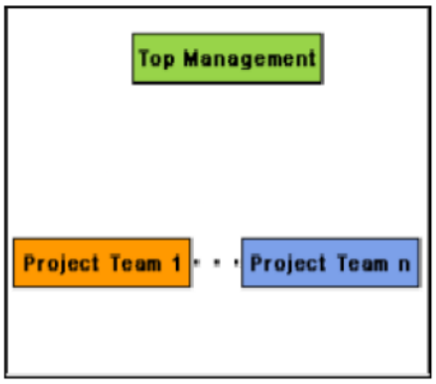
a. Functional Format



In the functional format, the development staff are divided based on the functional group to which they belong. The different projects borrow engineers from the required functional groups for specific phases to be undertaken in the project and return them to the functional group upon the completion of the phase.

In the functional format, different teams of programmers perform different phases of a project. For example, one team might do the requirements specification, another do the design, and so on. The partially completed product passes from one team to another as the project evolves. Therefore, the functional format requires considerable communication among the different teams because the work of one team must be clearly understood by the subsequent teams working on the project. This requires good quality documentation to be produced after every activity.

b. Project Format:



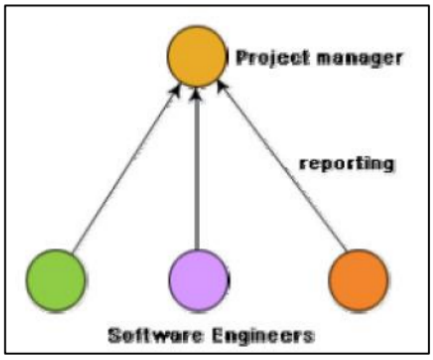
In the project format, the project development staff are divided based on the project for which they work.

In the project format, a set of engineers is assigned to the project at the start of the project and they remain with the project till the completion of the project. Thus, the same team carries out all the life cycle activities. Obviously, the functional format requires more communication among teams than the project format, because one team must understand the work done by the previous teams.

2. Team Structure:

Team structure addresses the issue of organization of the individual project teams. There are some possible ways in which the individual project teams can be organized. There are mainly three formal team structures: chief programmer, democratic, and the mixed team organizations although several other variations to these structures are possible. Problems of different complexities and sizes often require different team structures for chief solution.

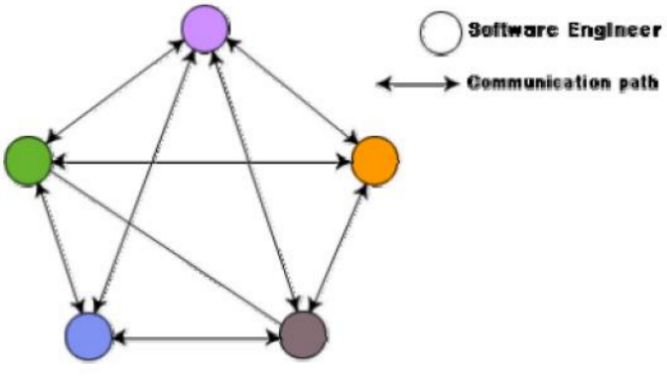
a. Chief Programmer Team:



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

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

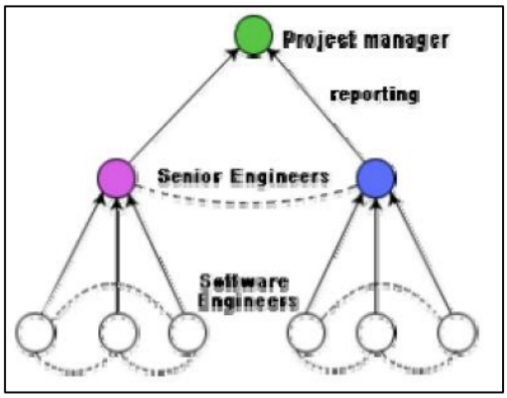
b. Democratic Team:



The democratic team structure, as the name implies, does not enforce any formal team hierarchy. Typically, a manager provides the administrative leadership. At different times, different members of the group provide technical leadership.

The democratic organization leads to higher morale and job satisfaction. Consequently, it suffers from less man-power turnover. Also, democratic team structure is appropriate for less understood problems, since a group of engineers can invent better solutions than a single individual as in a chief programmer team. A democratic team structure is suitable for projects requiring less than five or six engineers and for research-oriented projects. For large sized projects, a pure democratic organization tends to become chaotic. The democratic team organization encourages egoless programming as programmers can share and review one another's work.

c. Mixed Control Team Organization:



The mixed team organization, as the name implies, draws upon the ideas from both the democratic organization and the chief-programmer organization. This team organization incorporates both hierarchical reporting and democratic set up. In figure below, the democratic connections are shown as dashed lines and the reporting structure is shown using solid arrows. The mixed control team organization is suitable for large team sizes. The democratic arrangement at the senior engineer's level is used to decompose the problem into small parts. Each democratic setup at the programmer level attempts solution to a single part. Thus, this team organization is eminently suited to handle large and complex programs. This team structure is extremely popular and is being used in many software development companies.

SOFTWARE PROJECT STAFFING:

All the management activities that involve filling and keeping filled the positions that were established in the project organizational structure by well-qualified personnel.

MAJOR ISSUES IN STAFFING:

The major issues in staffing for a software engineering project are as follows:

- ☑ Project managers are frequently selected for their ability to program or perform engineering tasks rather than their ability to manage (few engineers make good managers).
- ☑ The productivity of programmers, analysts, and software engineers varies greatly from individual to individuals.
- ☑ There is a high turnover of staff on software projects especially those organized under a matrix organization.
- ☑ Universities are not producing a sufficient number of computer science graduates who understand the software engineering process or project management.
- ☑ Training plans for individual software developers are not developed or maintained.

QUALITY OF SOFTWARE ENGINEER:

- ❖ **Education:**

Does the candidate have the minimum level of education for the job? Does the candidate have the proper education for future growth in the company?

- ❖ **Experience:**

Does the candidate have an acceptable level of experience? Is it the right type and variety of experience?

- **Training:**

Is the candidate trained in the language, methodology, and equipment to be used, and the application area of the software system?

- **Motivation:**

Is the candidate motivated to do the job, work for the project, work for the company, and take on the assignment?

- **Commitment:**

Will the candidate demonstrate loyalty to the project, to the company, and to the decisions made?

- **Self-motivation:**

Is the candidate a self-starter, willing to carry a task through to the end without excessive direction?

- **Group affinity:**

Does the candidate fit in with the current staff? Are there potential conflicts that need to be resolved?

- **Intelligence:**

Does the candidate have the capability to learn, to take difficult assignments, and adapt to changing environments?

RISK MANAGEMENT:

Risk management is a series of steps that help a software team to understand and manage uncertainty. It's a really good idea to identify it, assess its probability of occurrence, estimate its impact, and establish a contingency plan that 'should the problem actually occur'. Risk management is a part of umbrella activities.

Simplistically, we can think of a risk as something that we prefer not to have happen. Risks may threaten the project, the software that is being developed or the organization.

CATEGORIES OF RISKS:

There are, three related categories of risk:

1. **Project Risks:**

Risks which will affect the project schedule or resources. For example, stuff turnover that is an experience team member of a project left the organization.

2. **Product Risks:**

Risks that affect the quality or performance of the software being developed. For example a component isn't performing as expected.

3. **Business Risks:**

Risks that affect the organization developing or procuring the software. For example, a competitor is developing a similar product that will challenge the product being developed.

Process of risks management:

Risks has to be listed and recovery actions has to be predetermined to avoid bigger impacts when development is underway. Following are the stages of risk management.

1. **Risk Identification:**

It is a systematic attempt to specify threats to the project plan (i.e. estimates, schedule, resource loading, etc.). Risks can be removed or avoided only if they can be identified. Basically all the risks are classified in two categories:

a. **Generic Risks:**

Generic risks are those risks which are common to all. A risk item checklist is created based upon the known and predictable risks to identify generics risks are as follows:

- **Product Size:** Risk associated with the product size of the software to be built.
- **Business Impact:** Risk associated with market place and constraints in management.
- **Customer Characteristics:** Risk associated with intensions of customer and the developer ability to communicate with the customer in timely manner.
- **Process Definition:** Risks associated with the degree to which the software process have been defined and is followed by development organization.
- **Development Environment:** Risk associated with the availability and quality of the tools to be used to build the product.
- **Technology to be built:** Risk associated with the complexity of the system to be built and due to advance of technology.

b. **Product Specific Risks:**

Product specific risks are those that are associated or relevant only to certain products. These risks occurs depending upon the nature and characteristics of product like the technology implemented, the people involved and the environment conditions. The identification of such risks requires proper knowledge of the special characteristics of the product and may vary from products to products.

Risk Projection and Risk Table:

Risk projection also known as risk estimation is the process of rating the risk based upon its likelihood or provability of occurrence and the consequence of the problems associated with the risks. It is carried out by project planner along with other managers and technical staff. The risk table is used in risk projection.

Risk Table

Risks	Category	Probability	Impact
Size estimate may be significantly low	PS	60%	2
Large numbers of inputs than planned	PS	30%	3

Less reuse than planned	PS	70%	2
End user resists system	BU	40%	3
Delivery deadline will be tight	BU	50%	2
Funding will be lost	CU	40%	1
Customer will change requirements	PS	80%	2
Technology will not meet expectations	TE	30%	1
Lack of training on tools	DE	80%	3
Staff turnover will be tight	ST	30%	2

Full Form	**Impact Value**
PS = Product Size	1 = Catastrophic
BU = Business Risk	2 = Critical
CU = Customer Characteristics	3 = Marginal
TE = Technology and Environment	4 = Negligible

2. **Risk Assessment (Refinement):**

Whenever we carryout project planning, at the previous stages, we state or specify the risk that occur in general, risks that are common, distinct and more likely to occurs. But as time passes by, we become more familiar and more involved in the project. As a result, we learn more and more about the project and the risk associated with it. Consequently, we will be able to refine the risk into more detailed and advance form, and finally we will be able to mitigate (avoid) monitor and manage (RMMM) the risk occurrence.

One way of risk assessment/ refinement is to represent the risk in condition-transition-consequence (CTC) format i.e. risk is stated in following form:

Given that <condition> then there is <consequence>

3. **Risk Containment:**

It assumes that mitigation effort have been failed and risk become a reality. Example: Number of people announces that they will be leaving. When the project is well under way if the mitigation strategy has been followed, backup is available information is documented and knowledge has been dispersed across the team.

Further, those individual who are leaving are asked to stop all work and spend their last week in "knowledge transfer mode" with the new comers who must be added to the team to get up to speed.

SOFTWARE CONFIGURATION MANAGEMENT:

Configuration management is a process of tracking and controlling the changes in software in terms of the requirements, design, functions and development of the product.

IEEE defines it as "the process of identifying and defining the items in the system, controlling the change of these items throughout their life cycle, recording and reporting the status of items and change requests, and verifying the completeness and correctness of items".

Generally, once the SRS is finalized there is less chance of requirement of changes from user. If they occur, the changes are addressed only with prior approval of higher management, as there is a possibility of cost and time overrun. One of the concept used for SCM is baseline.

Baseline:

A phase of SDLC is assumed over if it baselined, i.e. baseline is a measurement that defines completeness of a phase. A phase is baselined when all activities pertaining to it are finished and well documented. If it was not the final phase, its output would be used in next immediate phase.

Configuration management is a discipline of organization administration, which takes care of occurrence of any change (process, requirement, technological, strategical etc.) after a phase is baselined. CM keeps check on any changes done in software.

Necessarily of Software Configuration Management:

- Reduced redundant work.
- Effective management of simultaneous updates.
- Avoids configuration-related problems.
- Facilitates team coordination.
- Helps in building management; managing tools used in builds.
- Defect tracking: It ensures that every defect has traceability back to its source.

Change Control/Activities of SCM:

Change control is function of configuration management, which ensures that all changes made to software system are consistent and made as per organizational rules and regulations.

A change in the configuration of product goes through following steps:

- **Identification**: A change request arrives from either internal or external source. When change request is identified formally, it is properly documented.
- **Validation**: Validity of the change request is checked and its handling procedure is confirmed.
- **Analysis**: The impact of change request is analyzed in terms of schedule, cost and required efforts. Overall impact of the prospective change on system is analyzed.
- **Control**: If the prospective change either impacts too many entities in the system or it is unavoidable, it is mandatory to take approval of high authorities before change is incorporated into the system. It is decided if the change is worth incorporation or not. If it is not, change request is refused formally.
- **Execution**: If the previous phase determines to execute the change request, this phase take appropriate actions to execute the change, does a thorough revision if necessary.
- **Close request**: The change is verified for correct implementation and merging with the rest of the system. This newly incorporated change in the software is documented properly and the request is formally is closed.

Source Code Control System:

Source Code Control System (SCCS) is a version control system designed to track changes in source code and other text files during the development of a piece of software. This allows the user to retrieve any of the previous versions of the original source code and the changes which are stored. It was originally developed at Bell Labs in 1972 by Marc Rochkind for an IBM System/370 computer running OS/360.

Software is typically upgraded to a new version by fixing bugs, optimizing algorithms and adding extra functions. Popular software has many versions such as Java. Changing software causes problems that require version control to solve.

- ☑ Source code takes up too much space because it is repeated in every version.
- ☑ Passing optimization from one version to other versions is difficult.
- ☑ It is hard to acquire information about when and where changes occurred.
- ☑ Finding the exact version which the client has problems with is difficult.

SCCS was built to solve these problems. SCCS has nine major versions which are designed to help programmers control changes in software. Two specific implementations using SCCS are: PDP 11 under UNIX and IBM 370 under the OS.

Revision Control System (RCS):

The **Revision Control System** (**RCS**) is a software implementation of revision control that automates the storing, retrieval, logging, identification, and merging of revisions. RCS is useful for text that is revised frequently, for example programs, documentation, procedural graphics, papers, and form letters. RCS is also capable of handling binary files, though with reduced efficiency and efficacy. Revisions are stored with the aid of the diff utility.

RCS was initially developed in the 1980s by Walter F. Tichy while he was at Purdue University as a free and more evolved alternative to the then-popular Source Code Control System (SCCS). It is now part of the GNU Project but is still maintained by Purdue University.

RCS operates only on single files, has no way of working with an entire project. Although it provides branching for individual files, the version syntax is cumbersome. Instead of using branches, many teams just use the built-in locking mechanism and work on a single *head* branch.

Advantages

- ➢ Simple structure and easy to work with
- ➢ Revision saving is not dependent to the central repository

Disadvantages

- ➢ Poor security system when it comes to an intentional misconduct
- ➢ Only one user can work on a file at a time

Chapter – 7

Object Oriented Concepts and Principles

Object Oriented Paradigm:

The object oriented paradigm characterizes any problem domain as a set of objects that have specific attributes and behaviors. The objects in turn are characterized by classes and sub-classes.

The object oriented process moves through an evolutionary spiral that starts with customer communication. At this stage, the problem domain is defined and the basic problem classes are identified, planning is done to define resources, timelines and all other project related information.

The process involves:

a. Identify candidate classes
b. Lookup classes in library
c. Extract classes if available
d. Engineer/ develop classes if unavailable
   - Analysis
   - Design
   - Programming
   - Testing
e. Put new classes in library
f. Construct n^{th} iteration of system

Object Oriented Concepts:

1. Object:

An object is a real-world element in an object–oriented environment that may have a physical or a conceptual existence. Each object has:

- ☑ Identity that distinguishes it from other objects in the system.
- ☑ State that determines the characteristic properties of an object as well as the values of the properties that the object holds.
- ☑ Behavior that represents externally visible activities performed by an object in terms of changes in its state.

Objects can be modelled according to the needs of the application. An object may have a physical existence, like a customer, a car, etc.; or an intangible conceptual existence, like a project, a process, etc.

2. Class:

A class represents a collection of objects having same characteristic properties that exhibit common behavior. It gives the blueprint or description of the objects that can be created from it.

Creation of an object as a member of a class is called instantiation. Thus, object is an instance of a class.

The constituents of a class are:

- ☑ A set of attributes for the objects that are to be instantiated from the class. Generally, different objects of a class have some difference in the values of the attributes. Attributes are often referred as class data.
- ☑ A set of operations that portray the behavior of the objects of the class. Operations are also referred as functions or methods.

3. Encapsulation:

Encapsulation is the process of binding both attributes and methods together within a class. Through encapsulation, the internal details of a class can be hidden from outside. It permits the elements of the class to be accessed from outside only through the interface provided by the class.

4. Data Hiding:

Typically, a class is designed such that its data (attributes) can be accessed only by its class methods and insulated from direct outside access. This process of insulating an object's data is called data hiding or information hiding.

5. Message Passing:

Any application requires a number of objects interacting in a harmonious manner. Objects in a system may communicate with each other using message passing. Suppose a system has two objects: obj1 and obj2. The object obj1 sends a message to object obj2, if obj1 wants obj2 to execute one of its methods.

The features of message passing are:

- ☑ Message passing between two objects is generally unidirectional.
- ☑ Message passing enables all interactions between objects.
- ☑ Message passing essentially involves invoking class methods.
- ☑ Objects in different processes can be involved in message passing.

6. Inheritance:

Inheritance is the mechanism that permits new classes to be created out of existing classes by extending and refining its capabilities. The existing classes are called the base classes/parent classes/super-classes, and the new classes are called the derived classes/child classes/subclasses. The subclass can inherit or derive the attributes and methods of the super-class provided that the super-class allows so. Besides, the subclass may add its own attributes and methods and may modify any of the super-class methods. Inheritance defines an "is – a" relationship.

Types of Inheritance:

- ☑ **Single Inheritance:** A subclass derives from a single super-class.
- ☑ **Multiple Inheritance:** A subclass derives from more than one super-classes.

- ☑ Multilevel Inheritance: A subclass derives from a super-class which in turn is derived from another class and so on.
- ☑ **Hierarchical Inheritance:** A class has a number of subclasses each of which may have subsequent subclasses, continuing for a number of levels, so as to form a tree structure.
- ☑ **Hybrid Inheritance:** A combination of multiple and multilevel inheritance so as to form a lattice structure.

7. Polymorphism:

Polymorphism is originally a Greek word that means the ability to take multiple forms. In object-oriented paradigm, polymorphism implies using operations in different ways, depending upon the instance they are operating upon. Polymorphism allows objects with different internal structures to have a common external interface. Polymorphism is particularly effective while implementing inheritance.

Benefits of Object Model:

The benefits of using the object model are:

- ☑ It helps in faster development of software.
- ☑ It is easy to maintain. Suppose a module develops an error, then a programmer can fix that particular module, while the other parts of the software are still up and running.
- ☑ It supports relatively hassle-free upgrades.
- ☑ It enables reuse of objects, designs, and functions.
- ☑ It reduces development risks, particularly in integration of complex systems.

Identify Elements of an Object Model:

If we look around a room, there is a set of physical objects that can be easily identified, classified, and defined (in terms of attributes and operations). But when we "look around" the problem space of a software application, the objects may be more difficult to comprehend.

We can begin to identify objects by examining the problem statement or performing a "grammatical parse" on the processing narrative for the system to be built. Objects are determined by underlining each noun or noun clause and entering it in a simple table. Synonyms should be noted. If the object is required to implement a solution, then it is part of the solution space; otherwise, if an object is necessary only to describe a solution, it is part of the problem space. What should we look for once all of the nouns have been isolated? Objects manifest themselves in one of the ways represented below.

Objects can be:

- ☑ External entities (e.g., other systems, devices, people) that produce or consume information to be used by a computer-based system.
- ☑ Things (e.g., reports, displays, letters, signals) that are part of the information domain for the problem.
- ☑ Occurrences or events (e.g., a property transfer or the completion of a series of robot movements) that occur within the context of system operation.

- ☑ Roles (e.g., manager, engineer, salesperson) played by people who interact with the system.
- ☑ Organizational units (e.g., division, group, and team) that are relevant to an application.
- ☑ Places (e.g., manufacturing floor or loading dock) that establish the context of the problem and the overall function of the system.
- ☑ Structures (e.g., sensors, four-wheeled vehicles, or computers) that define a class of objects or in the extreme, related classes of objects.

It is also important to note what objects are not. In general, an object should never have an "imperative procedural name". For example, if the developers of software for a medical imaging system defined an object with the name image inversion, they would be making a subtle mistake. The image obtained from the software could, of course, be an object (it is a thing that is part of the information domain). Inversion of the image is an operation that is applied to the object. It is likely that inversion would be defined as an operation for the object image, but it would not be defined as a separate object to connote "image inversion." As Cashman states: "the intent of object-orientation is to encapsulate, but still keep separate, data and operations on the data."

MANAGEMENT OF OBJECT-ORIENTED SOFTWARE PROJECTS:

Project Management is the discipline of planning, organizing, securing and managing resources to bring about the successful completion of specific project goals and objectives. It is sometimes conflated with program management, however technically that is actually a higher level construction: a group of related and somehow interdependent

The primary challenge of project management is to achieve all of the engineering project goals and objectives while honoring the preconceived project constraints. Typical constraints are scope, time, and budget.

The secondary and more ambitious challenge is to optimize the allocation and integration of inputs necessary to meet pre-defined objectives.

In object oriented analysis we concentrated on identifying the objects that represented actual data within the business design. These objects are called entity objects. Entity objects usually correspond to items in real life and contain information, known as attributes, that describes the different instances of the entity. They also encapsulate those behaviors that maintain its information or attributes.

An entity object is said to be persistent, meaning the object typically lives on after the execution of a method. Two additional types of objects will be introduced during design. New objects will be introduced to represent a means through which the user will interface with the system. These objects are called interface objects. It is through the interface objects that the users communicate with the system. The use case functionality that describes the user directly interacting with the system should be placed in interface objects.

It translates the user's input into information that the system can understand and use to process the business event. Other types of objects that are introduced are objects that hold application or business rule logic. These objects are called control objects. They serve as the traffic cop

containing the application logic or business rules of the event for managing or directing the interaction between the objects.

1. Object Responsibility:

In design we focus on identifying the behaviors a system must support and, in turn, design the methods to perform those behaviors. Along with behaviors, we determine the responsibilities an object must have. An object responsibility is the obligation that an object has to provide a service when requested, thus corroborating with other objects to satisfy the request if required.

2. Object Framework:

An object framework is a set of related, interacting objects that provide a well-defined set of services for accomplishing a task. Component A component is a group of objects packaged together into one unit.

Management Spectrum:

The management spectrum describes the management of a software project or how to make a project successful. It focuses on the four P's; people, product, process and project. Here, the manager of the project has to control all these P's to have a smooth flow in the project progress and to reach the goal.

The four P's of management spectrum has been described briefly in below.

1. People:

People of a project includes from manager to developer, from customer to end user. But mainly people of a project highlight the developers. It is so important to have highly skilled and motivated developers that the Software Engineering Institute has developed a People Management Capability Maturity Model (PM-CMM), "to enhance the readiness of software organizations to undertake increasingly complex applications by helping to attract, grow, motivate, deploy, and retain the talent needed to improve their software development capability". Organizations that achieve high levels of maturity in the people management area have a higher likelihood of implementing effective software engineering practices.

Different people involves are:

a. Players
   - Senior Managers
   - Project (Technical) Managers
   - Practitioners
   - Customers
   - End user
b. Team Leader
c. Software Team

2. Product:

Product is any software that has to be developed. To develop successfully, product objectives and scope should be established, alternative solutions should be considered, and technical and management constraints should be identified. Without this information, it is impossible to define reasonable and accurate estimates of the cost, an effective assessment of risk, a realistic breakdown of project tasks or a manageable project schedule that provides a meaningful indication of progress.

3. Process:

A software process provides the framework from which a comprehensive plan for software development can be established. A number of different tasks sets tasks, milestones, work products, and quality assurance points enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team. Finally, umbrella activities overlay the process model. Umbrella activities are independent of any one framework activity and occur throughout the process.

4. Project:

Here, the manager has to do some job. The project includes all and everything of the total development process and to avoid project failure the manager has to take some steps, has to be concerned about some common warnings etc.

Chapter – 8

Emerging Trends

Client Server Software:

A client is basically a consumer of services and server is a provider of services. A client requests some services from the server and the server provides the required services to the client. Client and server are usually software components running on independent machines. Even a single machine can sometimes acts as a client and at other times a server depending on the situations. Thus, client and server are mere roles.

Example:

A man was visiting his friend's town in his car. The man had a handheld computer (client). He knew his friend's name but he didn't know his friend's address. So he sent a wireless message (request) to the nearest "address server" by his handheld computer to enquire his friend's address. The message first came to the base station. The base station forwarded that message through landline to local area network where the server is located. After some processing, LAN sent back that friend's address (service) to the man.

Advantages of client-server software development

There are many advantages of client-server software products as compared to monolithic ones. These advantages are:

- ☑ **Simplicity And Modularity:**

Client and server components are loosely coupled and therefore modular. These are easy to understand and develop.

- ☑ **Flexibility:**

Both client and server software can be easily migrated across different machines in case some machine becomes unavailable or crashes. The client can access the service anywhere. Also, clients and servers can be added incrementally.

- ☑ **Extensibility:**

More servers and clients can be effortlessly added.

- ☑ **Concurrency:**

The processing is naturally divided across several machines. Clients and servers reside in different machines which can operate in parallel and thus processing becomes faster.

- ☑ **Cost Effectiveness:**

Clients can be cheap desktop computers whereas severs can be sophisticated and expensive computers. To use a sophisticated software, one needs to own only a cheap client and invoke the server.

☑ **Specialization:**

One can have different types of computers to run different types of servers. Thus, servers can be specialized to solve some specific problems.

☑ **Current trend:**

Mobile computing implicitly uses client-server technique. Cell phones (handheld computers) are being provided with small processing power, keyboard, small memory, and LCD display. Cell phones cannot really compute much as they have very limited processing power and storage capacity but they can act as clients. The handhold computers only support the interface to place requests on some remote servers.

☑ **Application Service Providers (ASPs):**

There are many application software products which are very expensive. Thus it makes prohibitively costly to own those applications. The cost of those applications often runs into millions of dollars. For example, a Chemical Simulation Software named "Aspen" is very expensive but very powerful. For small industries it would not be practical to own that software. Application Service Providers can own ASPEN and let the small industries use it as client and charge them based on usage time. A client and simply logs in and ASP charges according to the time that the software is used.

☑ **Component-Based Development:**

It is the enabler of the client-server technology. Component-based development is radically different from traditional software development. In component-based development, a developer essentially integrates pre-built components purchased off-the-shelf. This is akin to the way hardware developers integrate ICs on a Printed Circuit Board (PCB). Components might reside on different computers which act as servers and clients.

☑ **Fault Tolerance:**

Client-server based systems are usually fault-tolerant. There can be many servers. If one server crashes then client requests can be switched to a redundant server.

Disadvantages of client-server software

There are several disadvantages of client-server software development. Those disadvantages are:

☑ **Security:**

In a monolithic application, implementation of security is very easy. But in a client-server based development a lot of flexibility is provided and a client can connect from anywhere. This makes it easy for hackers to break into the system. Therefore, ensuring security in client-server system is very challenging.

☑ **Servers Can Be Bottlenecks:**

Servers can turn out to be bottlenecks because many clients might try to connect to a server at the same time. This problem arises due to the flexibility given that any client can connect anytime required.

☑ **Compatibility:**

Clients and servers may not be compatible to each other. Since the client and server components may be manufactured by different vendors, they may not be compatible with respect to data types, language, etc.

☑ **Inconsistency:**

Replication of servers is a problem as it can make data inconsistent.

CORBA:

Common Object Request Broker Architecture (CORBA) is an architecture and specification for creating, distributing, and managing distributed program objects in a network. It allows programs at different locations and developed by different vendors to communicate in a network through an "interface broker." CORBA was developed by a consortium of vendors through the Object Management Group (OMG), which currently includes over 500 member companies. Both International Organization for Standardization (ISO) and X/Open have sanctioned CORBA as the standard architecture for distributed objects (which are also known as components). CORBA 3 is the latest level.

The essential concept in CORBA is the Object Request Broker (ORB). ORB support in a network of clients and servers on different computers means that a client program (which may itself be an object) can request services from a server program or object without having to understand where the server is in a distributed network or what the interface to the server program looks like. To make requests or return replies between the ORBs, programs use the General Inter-ORB Protocol (GIOP) and, for the Internet, it's Internet Inter-ORB Protocol (IIOP). IIOP maps GIOP requests and replies to the Internet's Transmission Control Protocol (TCP) layer in each computer.

A notable hold-out from CORBA is Microsoft, which has its own distributed object architecture, the Distributed Component Object Model (DCOM). However, CORBA and Microsoft have agreed on a gateway approach so that a client object developed with the Component Object Model will be able to communicate with a CORBA server (and vice versa).

Distributed Computing Environment (DCE), a distributed programming architecture that preceded the trend toward object-oriented programming and CORBA, is currently used by a number of large companies. DCE will perhaps continue to exist along with CORBA and there will be "bridges" between the two.

COM (COMPONENT OBJECT MODEL):

Component Object Model (COM) is a binary-interface standard for software components introduced by Microsoft in 1993. It is used to enable inter-process communication and dynamic object creation in a large range of programming languages. COM is the basis for several other Microsoft technologies and frameworks, including OLE, OLE Automation, Browser Helper

Object, ActiveX, COM+, DCOM, the Windows shell, DirectX, UMDF and Windows Runtime. The essence of COM is a language-neutral way of implementing objects that can be used in environments different from the one in which they were created, even across machine boundaries. For well-authored components, COM allows reuse of objects with no knowledge of their internal implementation, as it forces component implementers to provide well-defined interfaces that are separated from the implementation. The different allocation semantics of languages are accommodated by making objects responsible for their own creation and destruction through reference-counting. Type conversion casting between different interfaces of an object is achieved through the Query Interface method. The preferred method of "inheritance" within COM is the creation of sub-objects to which method "calls" are delegated.

DCOM (Distributed Component Object Model):

Distributed Component Object Model (DCOM) is a proprietary Microsoft technology for communication between software components on networked computers. DCOM, which originally was called "Network OLE", extends Microsoft's COM, and provides the communication substrate under Microsoft's COM+ application server infrastructure.

The addition of the "D" to COM was due to extensive use of DCE/RPC (Distributed Computing Environment/Remote Procedure Calls) more specifically Microsoft's enhanced version, known as MSRPC.

In terms of the extensions it added to COM, DCOM had to solve the problems of:

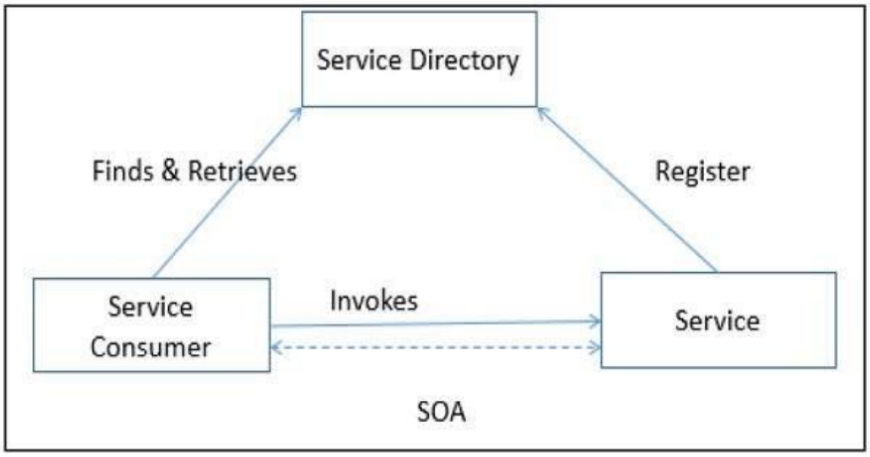
- ☑ Marshalling: serializing and deserializing the arguments and return values of method calls "over the wire".
- ☑ Distributed garbage collection: ensuring that references held by clients of interfaces are released when, for example, the client process crashed, or the network connection was lost.
- ☑ It had to combine Hundreds/Tens of Thousands of objects held in the client's browser with a single transmission in order to minimize bandwidth utilization.

DCOM was a major competitor to CORBA.

Service-Oriented Architecture (SOA):

A service is a component of business functionality that is well-defined, self-contained, independent, published, and available to be used via a standard programming interface. The connections between services are conducted by common and universal message-oriented protocols such as the SOAP Web service protocol, which can deliver requests and responses between services loosely.

Service-oriented architecture is a client/server design which support business-driven IT approach in which an application consists of software services and software service consumers (also known as clients or service requesters).



Features of SOA:

☑ **Distributed Deployment**:

Expose enterprise data and business logic as loosely, coupled, discoverable, structured, standard-based, coarse-grained, stateless units of functionality called services.

☑ **Composability**:

Assemble new processes from existing services that are exposed at a desired granularity through well defined, published, and standard complaint interfaces.

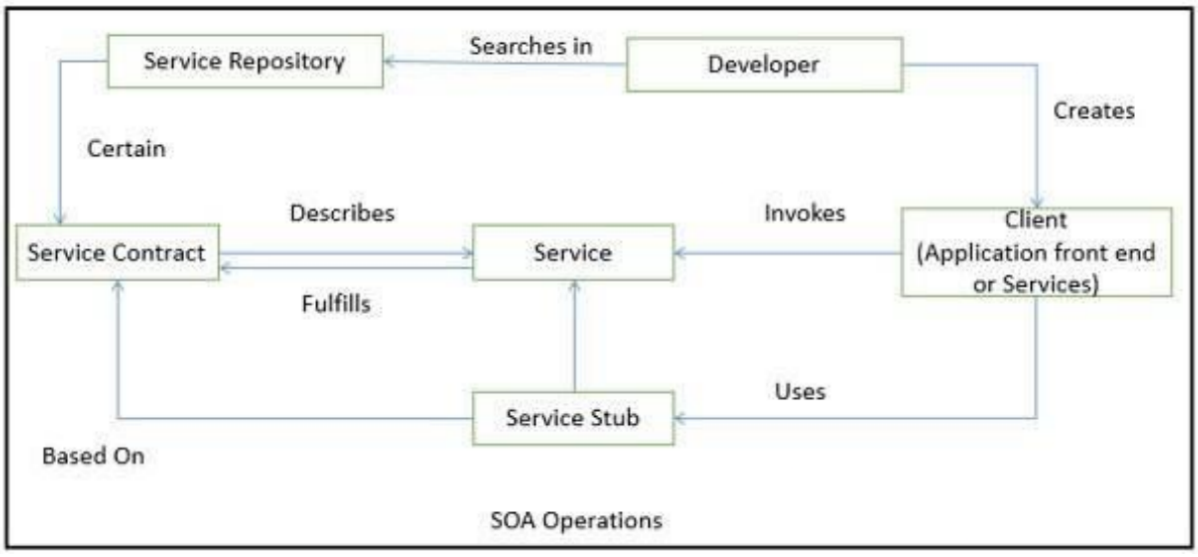
☑ **Interoperability**:

Share capabilities and reuse shared services across a network irrespective of underlying protocols or implementation technology.

☑ **Reusability**:

Choose a service provider and access to existing resources exposed as services.

SOA Operation:



SAAS (SOFTWARE AS A SERVICE):

Software as a service (SaaS) is a software distribution model in which a third-party provider hosts applications and makes them available to customers over the Internet. SaaS is one of three main categories of cloud computing, alongside infrastructure as a service (IaaS) and platform as a service (PaaS).

SaaS removes the need for organizations to install and run applications on their own computers or in their own data centers. This eliminates the expense of hardware acquisition, provisioning and maintenance, as well as software licensing, installation and support. Other benefits of the SaaS model include:

1. Flexible payments:

Rather than purchasing software to install, or additional hardware to support it, customers subscribe to a SaaS offering. Generally, they pay for this service on a monthly basis using a pay-as-you-go model. Transitioning costs to a recurring operating expense allows many businesses to exercise better and more predictable budgeting. Users can also terminate SaaS offerings at any time to stop those recurring costs.

2. Scalable usage:

Cloud services like SaaS offer high scalability, which gives customers the option to access more, or fewer, services or features on-demand.

3. Automatic updates:

Rather than purchasing new software, customers can rely on a SaaS provider to automatically perform updates and patch management. This further reduces the burden on in-house IT staff.

4. Accessibility and persistence:

Since SaaS applications are delivered over the Internet, users can access them from any Internet-enabled device and location.