**CHAPTER - 1**

**Introduction to software engineering:**

Software engineering is the establishment and used of sound engineering principles in order to obtain economical software i.e. reliable and works efficiently on real machine. Software engineering is the application of principles skills, art for the design and construction of program and system of programs.

Software engineering is the disciplined whose aim is the production of quality software i.e. delivered on time within budgets and that satisfies its requirements. Software engineering is the practical application of scientific knowledge in the design and construction of computer programs and the associated documentations require to develop, update and maintain them.

**Attributes of good software**

* Maintainability**:** It means the modifications that are made to a software system after its initial release. This includes a wide range of activities all having to do with modifying and existing piece of software to make and improvement.

It can be divided into three categories:

* + 1. Corrective maintenance**:** This includes the removal of residual error that are present in the product when it is delivered and error introduced into the software during its maintenance.
    2. Adoptivemaintenance**:** This involve the changing of software because the environment in which it is embedded changes.
    3. Perfectivemaintenance**:** This involve changing in software to improve some of its qualities.
* Portability**:** Software is portable if it can run on different environment means hardware platform or software environment such as a particular operating system.
* Productivity**:** It is a quality of the software production process referring to its efficiency and performance. an efficient process result in faster delivers of products.
* Efficiency**:** The software is termed good if it use the resources as its command in the most effective manner, i.e. software shouldn't make wasteful use of a system resources such as memory, processor and storage. Efficiency therefore includes the following:

1. Responsiveness
2. Processing time
3. Memory utilization

* Usability**:** Software becomes usable if it doesn't all for extra effort to be learnt. Usability increases with good documentation and appropriate user interface.
* Verifiability**:** A software system is Verifiable if its properties Can be Verified easily. Example it is important to be able to verify correctness or performance of a software system. Verification Can be performed by formal and informal analysis method or trough testing. Modular design, disciplined coding practice and use of an appropriate programming language all contributes to verifiability.
* Reparability**:** A software system is repairable if its defects Can be Corrected with a reasonable amount of work. As the Cost of a product decreases and the product assume the status of a Commodity, the need for reparability decreases. it is Cheaper to replace the whole thing or at least major of it tan to repair it.
* Understandability**:** Some software systems are easier to understand than others. Example, system that does weather forecasting will be harder to understand than the system that prints mailing list. there are certain guidelines which can be used to produce more understandable design and to write more understandable programs.
* Reliability**:**  Software is reliable if the user can depend on it. it can also be defined in terms of statistical behavior i.e. the probability that the software will operate as expected over a specified time in level.

**Roles of software engineering**

* Software takes a dual role: Product and at the same time, the vehicle for delivering a product.
* As a product, it delivers the computing potential embedded by computer hardware, or more broadly, a network of computer that are accessible by local hardware.
* As the vehicle use to deliver the product, software acts as the basis for the control of computer, the communication of information and the creation and control of other programs.
* The role of computer software has undergone significant change over a time span of little more than 50 years.
* Dramatic improvement in hardware performance, performed change in computing architecture, vast increase in memory and storage capacity and a wide variety of input/output options, have all participated and sophisticated and complex computer based system.

**Software engineering ethics**

The software engineering ethics are given below:

1. Public: Act consistently in the public interest or requirements about public.
2. Client and employer: Act in the manner that while protecting public interest, the best interest of the client or customer and employer are not ignored and are also consistent with the public interests. There has to be rational balance between the two.
3. Product: Ensure that products, modifications and there versions meet the highest professional standard.
4. Management: Subscribe and prompt ethical approach to the management of software development and maintenance.
5. Self: Participate in live long learning regarding software engineering practice and prompt control ethical approach to the professions.

**Some challenges of software engineering**

1. Legacy system: Old, valuable system which told that majority of larger software system in use must be maintained and updated.
2. Heterogeneity: System are distributed and include a mix of hardware and software. The heterogeneity challenge in the challenge of developing technique to built dependable software which is flexible enough to cope with heterogeneity.
3. Delivery: There is a increasing process for faster delivery of software. Unlike time consuming traditional software engineering concept technique. The delivery challenge is the challenge of shoring times for larger and complex system without compromising system quality.

**Difference between software engineering and system engineering**

System engineering is concerned with all aspects of a computer based system development including hardware, software, process engineering. Software engineering is a part of this process.

System engineers are involved in system specification architectural design, integration and development.

**Software development**

Software development (also known as application development, software design, design software, software application development or platform development) is the development of a software product.

The term software development of a software product maybe used to refer to the activity of computer programming, which is the process of writing and maintaining the source code but in a broader sense of the term it includes all that is involved between the conceptions of the desired software, ideally is the planned and structured process.

Therefore, software development may include research, new development, prototyping, modification, reuse, reengineering, maintenance or any other activities that result in software products.

Most methodology share some combination of the following stages of software development:

* Analyzing the problem.
* Market research.
* Gathering requirement for the proposed business solution.
* Design plan for the software based solution.
* Implementation (coding of the software)
* Testing the software.
* Deployment.
* Maintenance and bug fixing.

**CHAPTER - 2**

**Software specification**

The uses of specification**:** software specification represent the technical design of business process to be automated or supported by computer programs to be written by system builders.

A specification can be a written document, a graphical model, a formal mathematical model, a collection of uses scenarios, a prototype or any combination of these.

Example of software specification includes state transition diagram, flowchart, structure chart or unified modeling language diagram.

the designers intend is to prepare software specifications that:

* fulfill the business process requirements of system users and,
* provides sufficient details and consistency for communicating the software design to system builders.

It serves as a foundation for hardware engineering , software engineering and database engine. it describes the function and performance of computer based system and the constraints that will govern its development. the specification bounds each allocated system element. It also describes the information that is input and output from the system.

**Verification of specification**

* The software requirement specification document is verifiable if and only if every requirement stated there in is verifiable.
* An SRS document is said to be verifiable if for each requirement returning SRS, there exists some cost effective process using which a person or tool can ensure that the software meets the requirements.
* verifiability depends on the ways a requirement is stated in the documented ex: a requirement "user interface software should be user friendly is ambiguous and will have multiple interpretation by multiple user. hence it is not possible to verify these requirement. therefore the requirement must be documented carefully and the use of ambiguous statement or non-measureable should be avoided.
* A review of the software requirement specification conducted by both software developer and customer. because the specification form the foundation of developing phase.
* requirement validation examines the specification to ensure that all system requirements having unambiguous, that inconsistency, omissions and errors that have been detected and corrected and that the work product confirm to the standards established for the process, the project and the product.

**Specification qualities**

* a high quality software requirement specification is a prerequisites to high quality software if SRS is wrong, even though the correct and efficient implementation gives the wrong system. a high quality SRS contains few errors, show the implementation result in high quality software.
* a good quality SRS reduces the errors in systems, therefore helps in a tremendous reducing cost and schedule overruns and also helps in increasing the reputation of providing high quality software in time with less cost.

**Classification of specification style**

There are 3 basic approach to problem analysis. they are:

1. Informal approach: in this approach no defined methodologies is used. the analyst gathers information using questionnaires, brain storming, interviews, study of the existing system etc. it is particularly useful for the area of the problem for which there is no formal model is possible.
2. conceptual modeling: a formal model is built to represent the specific part of the system. such model are using the principles of particular problem
3. prototyping: in prototyping, a throwaway prototype is developed for the use of users. new requirements are elicited by getting the feedback of users.

**Types of specification**

1. Operational specification: these specification should specify the given inputs, validation conditions on inputs, operations to be performed on the inputs and corresponding outputs of operations, equations or logical operations that must be used to transform inputs into outputs. in addition, some abnormal input system behavior for invalid input must be specified.
2. Descriptive specifications: there are two types of descriptive specifications. they are:
3. static: static requirements include number of terminals to be supported, no of simultaneous user to be supported, the no. of file system has to process and their sizes.
4. dynamic: dynamic requirements include execution time, behavior of the system such as response time, expected time for completion of the operation, acceptable ranges of different performance parameters should be specified.

**CHAPTER - 3**

**Software testing techniques and strategies**

**Software testing fundamentals**

Testing objectives: Software testing is performed to fulfill the following objectives:-

1. Software quality improvements: software quality mean that conformance to the specified software design requirements are met. it includes as being correct, minimum requirement of quality and performing as required under specified circumstances.

The effects of bugs are very severe specially in critical operation. bugs and errors cause high losses and disaster. Example of bug causing critical issues are plane crashes, rocket launching, etc. the quality of software is vital aspect in the computerized world. Debugging is conducted to find out defects in design, which is developed by the programmers.

1. Verification and validation: testing is heavily used as a tool in the verification and validation process. tester can make claim but an interpretation of the testing results which either the products work under certain situation or it doesn't work.

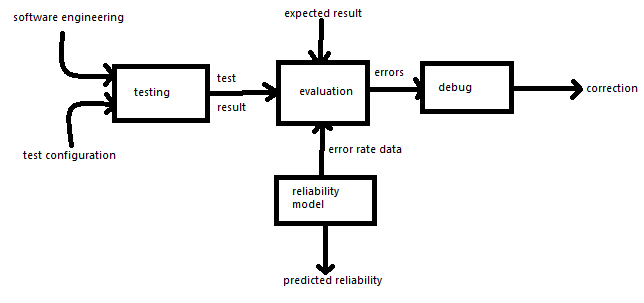
Software verification is the process of evaluating a system or component to determine whether the product of a given development phase satisfies the conditions ignored at the start of the phase.

Software validation is the process of evaluating a system or component during at the end of the development process to determine whether it satisfies specified requirement.

1. Software reliability estimation: the objective of testing is to discover, FZ, Honda, R15, avenger :D LOL. the residual design errors before delivering to the customer. the failure data during the testing process are taken in town to estimate the software reliability. the testing process may function with regular feedback from the reliability analysis to the testers and designers.

**How test information flow?**

Test information flow is an arrangement to check and maintain the right flow of information during software testing. Figure below is the representation of test information flow.



In figure above, we feed the software and test configuration specification to the testing block which generate test result. then this test results are fed into the evaluation block along with the expected results and error rate data. the expected results and the test results are compared and errors are generated. these errors are then fed into the debug block. the debug block corrects the error and produces corrected results.

**Test cases**

Test cases are required to find out the presence of fault in the system. Test cases are the input to the testing process. In order to reveal the correct behavior of the system it is necessary to have a large set of valid test cases. while selecting the test eases the primary objective is to ensure that if there is an error or fault in the program. An ideal test case set is one of that succeed only if there are no errors in programs. One possible ideal set of test of test cases is one that includes all the possible inputs to the program. This is often called as exhaustive (complete) testing. However exhaustive testing is impractical and inflexible as even for small program the no. of elements in the input domain can be extremely large. Hence realistic goal for testing is to select a set of test cases that is close to ideal.

The range and type of cases to be prepared in order to perform testing depends upon test criterion. a test criterion is a condition that must be satisfied by a set of test cases.

**Levels of testing**

The basic goal of the software development process is to produce software that has no errors or very few errors. verification is performed on the output of each phase, but some faults are likely to remain undetected by this method.

Testing is usually relied onto detect these faults, in addition o the faults introduced during code phase itself. hence, instead of testing the software as a whole, testing of software is done at various levels or stages, each level of testing aims to test different aspects of the software. Following are the various levels of testing:

1. Unit testing: The first level in the testing process is called unit testing. Unit testing concerns testing smallest component of the software that is modules. Different modules are tested against the specification produced during for the modules. Test cases are therefore designed to test the following:

* Internal logic of the module
* Functionality
* Interfaces
* Boundary conditions
* Data structure
* All paths in the program module

The focus in unit testing is therefore on the performance of single module by finding defects in that module. as the focus of this testing level is on testing the code.

1. Integration testing: The second level of testing is called integration testing. It is the process of combining multiples modules systematically for conducting test in order to find errors in the interface between modules is called integration testing. The goal here, is to see if the modules can be integrated problems. Hence, the emphasis is on testing the interface between modules.
2. System Testing: Here, at these level, the entire software system is tested. The reference and document for the purpose is the requirement documents, as the objective is to see if the software meets its requirement.

These technique involves testing of a product by verifying its working with the original requirement specification documents. A number of different types of testing are used at these level. they are as follows:

* + 1. Functional Testing: Functional testing is functioned with the functionality rather than implementation of the products. In functional testing, the structure of the program is not considered test cases are described solely on the basics of requirement of specification of the program or modules.
    2. Performance testing : Performance testing deals with related issues like security test, reliability test, etc. System testing is done at the developers and before the product is given to customer for use.
    3. Acceptance testing: After the system testing is completed successfully by the developers, acceptance testing is done at the customer ends. it is the customer or end users who design the test cases.

Acceptance testing is performed with realistic data, client to demonstrate that the software is working satisfactory. In these type of testing, emphasis is on usability testing of the product.

**Static program analysis tools (testing)**

Static analysis tool is also program analysis tools. it access and computes various of a software product without executing it. Typically, static analysis tools analyze some structural representation of a program to arrived at certain analytical conclusion. The structural properties that are usually analyze are:

* whether the coding standard have been at heard?
* certain programming errors such as uninitialized variables and mismatch between actual and formal parameter, variables that are declared but never used are also checked.

Code walk through and code inspections might be consider at static analyses method. but the term static program analyses is used to denote automated analyses tools. so, a compiler can be considered to be a static program.

**Dynamic program analysis tools (testing)**

Dynamic program analysis technique require the program to be executed and its actual behavior recorded. Dynamic analyzer usually instruments the code i.e. add-additional statement in the source code to collect program execution traces.

The instrumented code when executed allow us to record the behavior of the software for different test cases. After the software has been tested with its full test suite (pieces) and its behavior recorded, the dynamic analysis tool carries out a post execution analysis and produces reports which describes the structural coverage that has been achieved by a complete test suite for the program.

Normally, the dynamic analysis result are reported in the form of histogram for a pie chart to describe the structural coverage achieved for different modules of a program. The output of a dynamic analysis dynamic can be stored easily and provides evidence that thorough testing has been done.

1. Usability testing: It was introduced as a formal procedure in the late 1980s and began to achieve wide spread use in the mid-1990s. Usability testing is typically conducted in the laboratory where users perform normal task and usability specialist record observations.

It is a technology driven approach in which screen images of the application under evaluation of videoed, as are the user's body language and hand movements. Users also provide feedback in the form of satisfaction ratings and comments/suggestions on an evaluation form.

1. Regression testing: This levels of testing are performed when a system is being built from the components that has been coded. There is another level of testing, called Regression testing that is performed when some changes are made to an existing system.

We know that changes are fundamental to software. Any software must undergo changes. Frequently a change is made to upgrade the software by adding new features and functionality.

Clearly, the modified software needs to be tested to make sure that the new features to be added to perform intended work. however, as modification have been made to an existing system, testing also now to be done to make sure that the modification has not had only undesired side effects of making some of the earlier services faulty i.e. testing have to ensure that the desired behavior of the old services is maintained. This is the task of regression testing.

1. Stress testing: They are black box test which are designed to impose range of abnormal and even illegal input conditions so as to stress the capabilities of the software.

Input data volume, input data rate, processing time, utilization of memory etc. are tested beyond the design capacity. Example: suppose an operating system is designed to support 15 multi-program job; the system is designed by attempting to run 15 or more jobs simultaneously arrival of several high priority interrupts.

Stress testing usually involves an element of time or size such as the number of records transferred per unit time, the maximum number of users active at any time, input data size, etc.

1. Security testing: Computer based systems that handles sensitive data or causes actions which can penetrate into the system. Penetration includes activities such as hackers who try to get into the system, employees get into the system for revenge, dishonest people get into the system for illegal personal gain. Security testing is conducted to check the protection mechanism built into a system, will protect the system from illegal penetration or not.

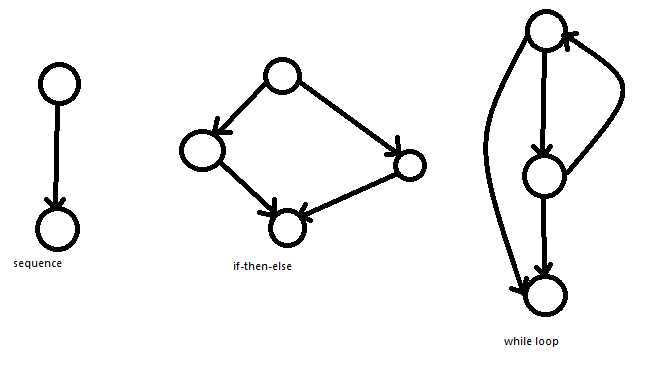
In Security testing, tester will act as an attacker who attacks to get into the system. the tester tries to get the password through external clerical ways might attack the system with customer software, hence denying service to others, might cause system errors get into the system during recovering browse through insecure data and hoping to get key to system entry.

1. Structural testing: It is also known as white box testing and it examines the internal structure of the program. In this strategy, we derive test cases from an examination of the program logic.

It is necessary because there might be a part of code which are not fully exercised by the functional test. there might be section of codes, which are surplus to requirement. This can be regarded as an error since it is deviation from requirement.

1. Path testing: Path testing is a name given to a group of test techniques based on selecting a set of test paths through the program. Path testing is used for module testing or unit testing. It requires complete knowledge of the programs structure. This type of testing involves:
2. Generating a set of path that will cover every branch in the program.
3. Finding a set of test cases that will execute every path in this set of program paths.
4. Flow graph: The control flow of a program can be analyzed using a graphical representation known as flow graph. The flow graph is a directed graph in which nodes are either statement or fragments of a statement and edges represent flow of control.

A flow graph can be easily generated from the code of any problem. The basic construct of flow graph are given below:



1. Independent path: An independent path is any path that introduces at least one new set of processing statements or conditions. Therefore, an independent path must move along at least one as that has not been traversed before the path is defined.

Independent path is used to ensure that:

* every statement in the program has been executed at least once.
* every branch has been exercised for true and false conditions.

1. Alpha testing: It is a testing in which a version of complete software is tested by the customer under the supervision of developer. this testing is performed at developers site. these test are conducted in a controlled environment.
2. Beta testing: It is testing in which the version of the software is tested by the customer without the developer being present . This testing is performed at customers sites. Online alpha testing, the developer is generally not present. Therefore, the beta test is a live application of the software in any environment that cannot be controlled by the developer. the customer records all problems (real or imagined) that are encountered during beta testing and reports these to the developer at regular intervals.

As a result of problems reported during beta test, software engineers make modifications and then prepare for release of the software product to the entire customer base.

1. Performance testing: It is designed to test the runtime performance of software within the context of an integrated system. performance testing occurs throughout all steps in the testing process. even at the unit level, the performance of an individual module maybe assessed as white box test are conducted. However, it is not until all system elements are fully integrated that the two performance of the system can be ascertained.

**Non-functional testing**

Non-functional testing refers to aspects of the software that may not be related to a specific function or user action, such as scalability or other performance behavior under certain constraints or security.

Testing will determine the breaking point, the point at which extremes of scalability or performance leads to unstable execution. Non-functional requirements tends to be those that reflect the quality of the product, particularly in the context of the suitability perspective of its users.

1. Domain testing: It is a type of function testing and tests the application by feeding interesting inputs and evaluating its outputs.

* It requires three or four tester to be derived for a relational expression. For a relational expression of the form.

E1 <relational operator> E2 when E1 and E2 are arithmetic expression.

* These testes are required to make the value of E1 greater than, equal to, or less than of E2.
* If <relational operator> is incorrect E1 and E2, are corrected then these three test guarantee the decision of the relational operators errors.
* To detect errors in E1 and E2, a test that makes the values of E1 greater or less than that of E2 should make the difference between these two values as small as possible.

1. Installation testing: An installation tests assures that the system is installed correctly and working at actual customers hardware.

* It is kind of quality assurance work in the software industry that focuses on what customer will need to look, install and setup the new software successfully.
* These testing is typically dune by the software testing engineer in conjunction with the configuration manager.

1. Load testing: A load testing is a type of non-functional testing.

* A load test is a type of software testing which is conducted to understand the behavior of the application under a specific expected load.
* Load testing is performed to determine a system behavior under both normal and at peak condition.
* It helps to identify the maximum operating capacity of an application as well as any bottleneck and determine which elements is carrying degradation. e.g. if the no. of users are increased then how much CPU memory will be consumed, what is the network and bandwidth response time.
* Load testing differs from stress testing, which evaluate the extent to which a system keep working when subjected to stream workloads or where some of its hardware and software has been compromised.
* The primary goal of load testing is to define maximum amount of work a system can handle without significant performance degradation. e.g. downloading a series of large files from the internet, running multiple application or computer or server simultaneously.

1. Manual testing: It includes the testing of the software i.e. without using automated.

* In these type of testing the tester takes over the role of an end users, test the software to identify an unexpected behavior or bug.
* There are different stages of manual testing like unit testing, integration testing, system testing.
* To ensure completeness of testing, the tester often follow a return test plan i.e. leads them through a set of important test cases.

1. Automated testing: It is the automating of test cases so that test may run multiple times with minimal efforts. To do these the test cases themselves are coded.

* However, automated test can be time consuming and expensive. The test code based can be as large as the project code based and test code must be maintained.
* test automation can automate same repetitive but necessary task in a formalize testing process can be difficult to perform manual.

**Testing lifecycle**

Software testing lifecycle is the testing process which is executed in systematic and planned manner. In SDLC process, different activities are carried out to improve the quality of the product. Following are the steps involved in software testing life cycle. Each step has its own entry criteria and deliverable.

1. Requirement analysis: It is the very first step in SDLC. In this step quality assurance (QA) team understands the requirements in terms of what we will testing and figure out the testable requirement.
2. Test planning: It is the most important phase of SDLC where all testing strategy is defined. This phase also called test strategy phase. In this phase typically test manager (or test level based on company to company) involved to determine the effort and cost estimates for entire project. The result of test planning phase will be test plan or test strategy and testing effort estimation documents.
3. Test case development: Test case development activity is started once the test planning activity is finished. This phase of SDLC where testing team write downs the detailed test cases. Once the test cases are ready then test cases are received by peer member or QA lead.
4. Environmental setup: setting up the test environment is vital part of the SDLC. Basically test environment decides on which condition software is tested. This is independent activity and can be started parallel with test case development. In process of setting up testing environmental test team is not involved in it.
5. Test execution: Once the preparation of test case development and test environment is completed then test execution phase can be kicked off. In this phase testing team start executing test case based on prepared test planning and prepared test cases in the prior step.
6. Test cycle closure: call out the testing team member meeting and evaluate cycle completion criteria based on test coverage, quality, cost, time, critical business objective, and software.

**Difference between Alpha testing and Beta testing**

|  |  |
| --- | --- |
| Alpha testing | Beta testing |
| Alpha testing performed by testers who are usually internal employers of the organization. | Beta testing is performed by clients or end user who are not employees of the organization. |
| Alpha testing performed at developer's site. | Beta testing is performed at client location or end user of the product. |
| Reliability & security testing are not performed in-depth alpha testing. | Reliability, security, robustness are checked during beta testing. |
| Alpha testing involved both the white box and black box technique. | Beta testing typically uses black box testing. |
| Alpha testing requires lab environment or testing environment. | It doesn't require any lab environment or testing environment. Software is made available to public & is said to be real time environment. |
| Long execution cycle may be required for alpha testing. | Only few weeks of execution are required for beta testing. |
| Critical issue or fixes can be addressed by developer immediately in alpha testing. | Most of the issues or feedback is collected from beta testing will be implemented in future versions of the product. |
| Alpha testing is to ensure the quality of the product before moving to beta testing. | Beta testing also concentrates on quality of the product but gathers users input on the product and ensures that the product is ready for real time users. |

**Advantages of Beta testing**

* Reduces product failure risk via customer validation.
* Beta testing allows a company to test post launch infrastructure.
* Improves product quality via customer feedback.
* Cost effective compared to similar data gathering methods.
* Creates goodwill with customers & increases the customer satisfaction.

**Disadvantages of Beta testing**

* Test management is an issue.
* Finding the right beta users and maintaining their participation would be a challenge.

**Advantages of Alpha testing**

* Provides better view about the reliability of the software at an early stage.
* Help simulate real time user behavior and environment.
* Detect many serious errors.
* Ability to provide early detection of errors with respect to design and functionality.

**Disadvantages of Alpha testing**

* In depth functionality cannot be tested as software is still under development stage sometimes developers and testers are dissatisfied with the result of alpha testing.

**Difference between Manual and Automatic testing**

|  |  |
| --- | --- |
| Manual testing | Automatic testing |
| In manual testing case the name suggested, test cases are executed manually (by a human) without any support from tools or scripts. | In automatic testing test cases are executed with the assistance of tools, scripts and software. |
| 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 lacking 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 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 has human observation and cannot guarantee user-friendliness and positive customer experience. |

**Difference between black box testing and white box testing**

|  |  |
| --- | --- |
| Black Box Testing | White Box Testing |
| Also referred as functional testing as it test the functionality of system. | Also referred as structural testing as it checks for all independent logic path |
| Also known as behavioral testing as system behavior is determined by studying its input and related output. | Also known as glass box testing as its internal coding part should be tested and verified. |
| Knowledge of internal program logic is not important. | Knowledge of internal program logic is essential. |
| Objective is to check to meet the user requirement as per the specification documents. | Objective is to check all program statement as per the design specification. |
| Mostly performed by the user's side. | Mostly performed by the programmer side. |
| Testing include equivalent partitioning boundary value analysis, comparison etc. | Testing includes basic path and control structural testing. |

**11-Steps of testing process (Only steps should be covered) :-**

11-steps of testing process are given below:-

1. Assess development plan and status.
2. Develop the test plan.
3. Test software requirement.
4. Test software design.
5. Conduct program phase testing.
6. execute and record result.
7. Conduct acceptance test.
8. Report test result.
9. Test software installation.

10) Test software changes

11) Evaluate test effectiveness.

**Tester's workbench**

* A workbench is a method of documenting how a particular activity must be fulfilled. A workbench is referred to as a stages, steps and assignment.
* A tester workbench is one part of the software development life cycle, which is comprised of many workbench. Two examples of the workbench concept are given below:

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

* input (program specification) is given to the producers (programmers).
* work (e.g. coding and debugging) is perform a procedure is followed, and a product deliverable is produced.
* A project team uses the workbench to guide them through a unit test of computer code. The programmer takes the following steps:
* give input products (e.g. program code) to the tester.
* Perform work (e.g. execute unit test) follow a procedure and produce a product deliverable (e.g. the test result)
* The objective of the workbench is to produce the defined output products is defect face manner. The procedures and standard are designed to assist in this objective.
* The worker performs defined procedure on the input products in order to produce the output product. The procedures are step by step instructions that the workers follow in performing these for tasks.

**CHAPTER - 6**

**Responsibilities of a software project manager**

Software project managers take the overall responsibility of steering a project to success. The job responsibility of a project manager ranges from invisible activities like building up team morale to highly visible customer presentations. Most managers take responsibility for project proposal writing, project cost estimation, scheduling, project staffing, software process tailoring, project monitoring and control, software configuration management, risk management, interfacing with clients, managerial report writing and presentations, etc. These activities are certainly numerous, varied and difficult to enumerate, but these activities can be broadly classified into project planning, and project monitoring and control activities. The project planning activity is undertaken before the development starts to plan the activities to be undertaken during development. The project monitoring and control activities are undertaken once the development activities start with the aim of ensuring that the development proceeds as per plan and changing the plan whenever required to cope up with the situation.

**Skills necessary for software project management**

A theoretical knowledge of different project management techniques is certainly

necessary to become a successful project manager. However, effective software

project management frequently calls for good qualitative judgment and decision

taking capabilities. In addition to having a good grasp of the latest software

project management techniques such as cost estimation, risk management,

configuration management, project managers need good communication skills

and the ability get work done. However, some skills such as tracking and controlling the progress of the project, customer interaction, managerial

presentations, and team building are largely acquired through experience.

Project planning

Once a project is found to be feasible, software project managers undertake

project planning. Project planning is undertaken and completed even before any

development activity starts. Project planning consists of the following essential

activities:

• Estimating the following attributes of the project:

Project size: What will be problem complexity in terms of the effort and time required to develop the product?

Cost: How much is it going to cost to develop the project?

Duration: How long is it going to take to complete development?

Effort: How much effort would be required?

Software Project Management Plan (SPMP)

Once project planning is complete, project managers document their plans in a

Software Project Management Plan (SPMP) document. The SPMP document

should discuss a list of different items that have been discussed below. This list

can be used as a possible organization of the SPMP document.

Organization of the Software Project Management Plan (SPMP) Document

1. Introduction

(a) Objectives

(b) Major Functions

(c) Performance Issues

(d) Management and Technical Constraints

2. Project Estimates

(a) Historical Data Used

(b) Estimation Techniques Used

(c) Effort, Resource, Cost, and Project Duration Estimates

3. Schedule

(a) Work Breakdown Structure

(b) Task Network Representation

(c) Gantt Chart Representation

(d) PERT Chart Representation

4. Project Resources

(a) People

(b) Hardware and Software

(c) Special Resources

5. Staff Organization

(a) Team Structure

(b) Management Reporting

6. Risk Management Plan

(a) Risk Analysis

(b) Risk Identification

(c) Risk Estimation

(d) Risk Abatement (to become less intense) Procedures

7. Project Tracking and Control Plan

8. Miscellaneous Plans

(a) Process Tailoring

(b) Quality Assurance Plan

(c) Configuration Management Plan

(d) Validation and Verification

(e) System Testing Plan

(f) Delivery, Installation, and Maintenance Plan

Metrics for software project size estimation

Accurate estimation of the problem size is fundamental to satisfactory estimation

of effort, time duration and cost of a software project. In order to be able to

accurately estimate the project size, some important metrics should be defined in

terms of which the project size can be expressed. The size of a problem is

obviously not the number of bytes that the source code occupies. It is neither the

byte size of the executable code.

The project size is a measure of the problem complexity in terms of the effort and time required to develop the product. Currently two metrics are popularly being used widely to estimate size: lines of code (LOC) and function point (FP). The usage of each of these metrics in project size estimation has its own advantages and disadvantages.

Lines of Code (LOC)

LOC is the simplest among all metrics available to estimate project size. This

metric is very popular because it is the simplest to use. Using this metric, the

project size is estimated by counting the number of source instructions in the

developed program. Obviously, while counting the number of source instructions,

lines used for commenting the code and the header lines should be ignored.

Determining the LOC count at the end of a project is a very simple job.

However, accurate estimation of the LOC count at the beginning of a project is

very difficult. In order to estimate the LOC count at the beginning of a project,

project managers usually divide the problem into modules, and each module into

sub modules and so on, until the sizes of the different leaf-level modules can be

approximately predicted. By using the estimation of the lowest level modules,

project managers arrive at the total size estimation.

Function point (FP)

Function point metric was proposed by Albrecht [1983]. This metric overcomes

many of the shortcomings of the LOC metric. Since its inception in late 1970s,

function point metric has been slowly gaining popularity. One of the important

advantages of using the function point metric is that it can be used to easily

estimate the size of a software product directly from the problem specification.

This is in contrast to the LOC metric, where the size can be accurately

determined only after the product has fully been developed.

The conceptual idea behind the function point metric is that the size of a

software product is directly dependent on the number of different functions or

features it supports. A software product supporting many features would certainly

be of larger size than a product with less number of features. Each function when

invoked reads some input data and transforms it to the corresponding output

data. For example, the issue book feature (as shown in fig. 11.2) of a Library

Automation Software takes the name of the book as input and displays its

location and the number of copies available. Thus, a computation of the number

of input and the output data values to a system gives some indication of the

number of functions supported by the system. Albrecht postulated that in addition

to the number of basic functions that a software performs, the size is also

dependent on the number of files and the number of interfaces.



Fig. System function as a map of input data to output data

Project Estimation techniques

Estimation of various project parameters is a basic project planning activity. The

important project parameters that are estimated include: project size, effort

required to develop the software, project duration, and cost. These estimates not

only help in quoting the project cost to the customer, but are also useful in

resource planning and scheduling. There are three broad categories of

estimation techniques:

• Empirical estimation techniques

• Heuristic techniques

• Analytical estimation techniques

Empirical Estimation Techniques

Empirical (on observation or experiment, not on theory) estimation techniques are 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.

Expert Judgment Technique

Expert judgment is one of the most widely used estimation

techniques. In this approach, an expert makes an educated guess of the

problem size after analyzing the problem thoroughly. Usually, the expert

estimates the cost of the different components (i.e. modules or

subsystems) of the system and then combines them to arrive at the overall

estimate. However, this technique is subject to human errors and

individual bias. Also, it is possible that the expert may overlook some

factors inadvertently (not done intentionally). Further, an expert making an estimate may not have experience and knowledge of all aspects of a project. For example, he may be conversant (having knowledge of sth) with the database and user interface parts but may not be very knowledgeable about the computer communication part.

A more refined form of expert judgment is the estimation made by group of experts. Estimation by a group of experts minimizes factors such as individual oversight, lack of familiarity with a particular aspect of a

project, personal bias, and the desire to win contract through overly

optimistic estimates. However, the estimate made by a group of experts

may still exhibit bias on issues where the entire group of experts may be

biased due to reasons such as political considerations. Also, the decision

made by the group may be dominated by overly assertive (showing a strong personality) members.

Delphi cost estimation

Delphi cost estimation approach tries to overcome some of the shortcomings of the expert judgment approach. Delphi estimation is carried out by a team comprising of a group of experts and a coordinator.

In this approach, the coordinator provides each estimator with a copy of the software requirements specification (SRS) document and a form for recording his cost estimate. Estimators complete their individual estimates anonymously and submit to the coordinator. In their estimates, the estimators mention any unusual characteristic of the product which has influenced his estimation. The coordinator prepares and distributes the summary of the responses of all the estimators, and includes any unusual rationale (the principles or reasons on which sth is based) noted by any of the estimators. Based on this summary, the estimators re-estimate. This process is iterated for several rounds.

However, no discussion among the estimators is allowed during the entire estimation process. The idea behind this is that if any discussion is allowed among the estimators, then many estimators may easily get influenced by the rationale of an estimator who may be more experienced or senior. After the completion of several iterations of estimations, the

coordinator takes the responsibility of compiling the results and preparing the final estimate.

Heuristic Techniques

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:

Estimated Parameter = c1 \* ed1

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. c1 and d1 are constants. The values of the constants c1 and d1 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:

Estimated Resource = c1\*e1d1 + c2\*e2d2 + ...

Where e1, e2, … are the basic (independent) characteristics of the software already estimated, and c1, c2, d1, d2, … 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 c1, c2, d1, d2, … . 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.

Analytical Estimation Techniques

Analytical estimation techniques derive 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. Halstead’s software science can be used to derive some interesting results starting with a few simple assumptions. Halstead’s software science 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 forover all program length, potential minimum value, actual volume, effort,

and development time.

For a given program, let:

η1 be the number of unique operators used in the program,

η2 be the number of unique operands used in the program,

N1 be the total number of operators used in the program,

N2 be the total number of operands used in the program.

Length and Vocabulary

The length of a program as defined by Halstead, quantifies total usage of all operators and operands in the program. Thus, length N = N1 +N2. Halstead’s definition of the length of the program as the total number of operators and operands roughly agrees with the intuitive notation of the program length as the total number of tokens used in the program. The program vocabulary is the number of unique operators and operands used in the program. Thus, program vocabulary η = η1 + η2.

Project scheduling

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

1. Identify all the tasks needed to complete the project.

2. Break down large tasks into small activities.

3. Determine the dependency among different activities.

4. Establish the most likely estimates for the time durations necessary to complete the activities.

5. Allocate resources to activities.

6. Plan the starting and ending dates for various activities.

7. Determine the critical path. A critical path is the chain of activities that determines the duration of the project.

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

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

Once the activity network representation has been worked out, resources are allocated to each activity. Resource allocation is typically done using a Gantt chart. After resource allocation is done, a PERT chart representation is developed. The PERT chart representation is suitable for program monitoring and control. For task scheduling, the project manager needs to decompose the project tasks into a set of activities. The time frame when each activity is to be

performed is to be determined. The end of each activity is called milestone (a very important stage in the development so sth). The project manager tracks the progress of a project by monitoring the timely completion of the milestones. If he observes that the milestones start getting delayed, then he has to carefully control the activities, so that the overall deadline can still be met.

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.

Functional format vs. project format

There are essentially two broad ways in which a software development organization can be structured: functional format and project format. In the project format, the project development staff are divided based on the project for which they work (as shown in fig. 12.1). 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.



(a) Project Organization

(b) Functional Organization

Fig. 12.1: Schematic representation of the functional and project organization

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.

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

Team structures

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.

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 structure of the chief programmer team is shown in fig. 12.2. 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 (the amount of confidence a group has at a particular time), since team-members work under the constant supervision of the chief programmer. This also inhibits (to make sb nervous from doing sth) 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.



Fig. 12.2: Chief programmer team structure

Democratic Team

The democratic team structure, as the name implies, does not enforce any formal team hierarchy (as shown in fig. 12.3). 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 (complete confusion). The democratic team organization encourages egoless programming as programmers can share and review one another’s work.

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. The mixed control team organization is shown pictorially in fig. 12.4. This team organization incorporates both hierarchical reporting and democratic set up. In fig. 12.4, 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 engineers level is used to decompose the problem into small parts. Each democratic setup at the programmer level attempts solution to a single part. Thus, this team organization is eminently suited to handle large and complex programs. This team structure is extremely popular and is being used in many software development companies.



Fig. 12.4: Mixed team structure

Qualities of a good software engineer

The attributes that good software engineers should posses are as follows:

• Exposure to systematic techniques, i.e. familiarity with software engineering principles.

• Good technical knowledge of the project areas (Domain knowledge).

• Good programming abilities.

• Good communication skills. These skills comprise of oral, written, and interpersonal skills.

• High motivation.

• Sound knowledge of fundamentals of computer science.

• Intelligence.

• Ability to work in a team.

• Discipline, etc.

Risk management

A software project can be affected by a large variety of risks. In order to be able to systematically identify the important risks which might affect a software project, it is necessary to categorize risks into different classes. The project manager can then examine which risks from each class are relevant to the project. There are three main categories of risks which can affect a software project:

Project risks

Project risks concern varies forms of budgetary, schedule, personnel, resource, and customer-related problems. An important project risk is schedule slippage. Since, software is intangible, it is very difficult to monitor and control a software project. It is very difficult to control something which cannot be seen. For any manufacturing project, such as manufacturing of cars, the project manager can see the product taking shape. He can for instance, see that the engine is fitted, after that the doors are fitted, the car is getting painted, etc. Thus he can easily assess the progress of the work and control it. The invisibility of the product being developed is an important reason why many software projects suffer from the risk of schedule slippage.

Technical risks

Technical risks concern potential design, implementation, interfacing, testing, and maintenance problems. Technical risks also include ambiguous specification, incomplete specification, changing specification, technical uncertainty, and technical obsolescence (going out of use). Most technical risks occur due to the development team’s insufficient knowledge about the project.

Business risks

This type of risks include risks of building an excellent product that no one wants, losing budgetary or personnel commitments, etc.

Risk assessment

The objective of risk assessment is to rank the risks in terms of their damage causing potential. For risk assessment, first each risk should be rated in two ways:

• The likelihood of a risk coming true (denoted as r).

• The consequence of the problems associated with that risk (denoted as s).

Based on these two factors, the priority of each risk can be computed:

p = r \* s

Where, p is the priority with which the risk must be handled, r is the probability of the risk becoming true, and s is the severity of damage caused due to the risk becoming true. If all identified risks are prioritized, then the most likely and damaging risks can be handled first and more comprehensive risk abatement (to become less intense) procedures can be designed for these risks.

Risk containment (the keeping of sth within limits, so that it cannot spread in a harmful way)

After all the identified risks of a project are assessed, plans must be made to contain the most damaging and the most likely risks. Different risks require different containment procedures. In fact, most risks require ingenuity (for solving problems in a cleaver original way) on the part of the project manager in tackling the risk. There are three main strategies to plan for risk containment:

Avoid the risk: This may take several forms such as discussing with the customer to change the requirements to reduce the scope of the work, giving incentives (a thing that encourages sb to do sth) to the engineers to avoid the risk of manpower turnover, etc.

Transfer the risk: This strategy involves getting the risky component developed by a third party, buying insurance cover, etc.

Risk reduction: This involves planning ways to contain the damage due to a risk. For example, if there is risk that some key personnel might leave, new recruitment may be planned.

Software configuration management

The results (also called as the deliverables) of a large software development effort typically consist of a large number of objects, e.g. source code, design document, SRS document, test document, user’s manual, etc. These objects are usually referred to and modified by a number of software engineers throughout the life cycle of the software. The state of all these objects at any point of time is called the configuration of the software product. The state of each deliverable object changes as development progresses and also as bugs are detected and fixed.

Release vs. Version vs. Revision

A new version of a software is created when there is a significant change in functionality, technology, or the hardware it runs on, etc. On the other hand a new revision of a software refers to minor bug fix in that software. A new release is created if there is only a bug fix, minor enhancements to the functionality, usability, etc.

Necessity of software configuration management

There are several reasons for putting an object under configuration management. But, possibly the most important reason for configuration management is to control the access to the different deliverable objects. Unless strict discipline is enforced regarding updation and storage of different objects, several problems appear. The following are some of the important problems that appear if configuration management is not used.

Inconsistency problem when the objects are replicated. A scenario can be considered where every software engineer has a personal copy of an object (e.g. source code). As each engineer makes changes to his local copy, he is expected to intimate (to make sth known to sb) them to other engineers, so that the changes in interfaces are uniformly changed across all modules. However, many times an engineer makes changes to the interfaces in his local copies and forgets to intimate other teammates about the changes. This makes the different copies of the object inconsistent. Finally, when the product is integrated, it does not work. Therefore, when several team members work on developing an object, it is necessary for them to work on a single copy of the object, otherwise inconsistency may arise.

Problems associated with concurrent access. Suppose there is a single copy of a problem module, and several engineers are working on it. Two engineers may simultaneously carry out changes to different portions of the same module, and while saving overwrite each other. Though the problem associated with concurrent access to program code has been explained, similar problems occur for any other deliverable object.

Providing a stable development environment. When a project is underway, the team members need a stable environment to make progress. Suppose somebody is trying to integrate module A, with the modules B and C, he cannot

make progress if developer of module C keeps changing C; this can be especially frustrating if a change to module C forces him to recompile A. When an effective configuration management is in place, the manager freezes the objects to form a base line. When anyone needs any of the objects under configuration control, he is provided with a copy of the base line item. The requester makes changes to his private copy. Only after the requester is through with all modifications to his private copy, the configuration is updated and a new base line gets formed instantly. This establishes a baseline for others to use and depend on. Also, configuration may be frozen periodically. Freezing a configuration may involve archiving everything needed to rebuild it. (Archiving means copying to a safe place such as a magnetic tape).

System accounting and maintaining status information. System accounting keeps track of who made a particular change and when the change was made.

Handling variants. Existence of variants of a software product causes some peculiar problems. Suppose somebody has several variants of the same module, and finds a bug in one of them. Then, it has to be fixed in all versions and revisions. To do it efficiently, he should not have to fix it in each and every version and revision of the software separately.

Software configuration management activities

Normally, a project manager performs the configuration management activity by using an automated configuration management tool. A configuration management tool provides automated support for overcoming all the problems

mentioned above. In addition, a configuration management tool helps to keep track of various deliverable objects, so that the project manager can quickly and unambiguously determine the current state of the project. The configuration management tool enables the engineers to change the various components in a controlled manner.

Configuration management is carried out through two principal activities:

• Configuration identification involves deciding which parts of the system should be kept track of.

• Configuration control ensures that changes to a system happen smoothly.

Configuration identification

The project manager normally classifies the objects associated with a software development effort into three main categories: controlled, pre controlled, and uncontrolled. Controlled objects are those which are already put under configuration control. One must follow some formal procedures to change them. Pre controlled objects are not yet under configuration control, but will eventually be under configuration control. Uncontrolled objects are not and will not be subjected to configuration control. Controllable objects include both controlled and pre controlled objects.

Typical controllable objects include:

• Requirements specification document

• Design documents

• Tools used to build the system, such as compilers, linkers, lexical (words of a language) analyzers, parsers (breaker of inputs in chunks), etc.

• Source code for each module

• Test cases

• Problem reports

The configuration management plan is written during the project planning phase and it lists all controlled objects. The managers who develop the plan must strike a balance between controlling too much, and controlling too little. If too much is controlled, overheads (high expenses) due to configuration management increase to unreasonably high levels. On the other hand, controlling too little might lead to confusion when something changes.

Configuration control

Configuration control is the process of managing changes to controlled objects. Configuration control is the part of a configuration management system that most directly affects the day-to-day operations of developers. The configuration control system prevents unauthorized changes to any controlled objects. In order to change a controlled object such as a module, a developer can get a private copy of the module by a reserve operation as shown in fig. 12.5. Configuration management tools allow only one person to reserve a module at a time. Once an object is reserved, it does not allow anyone else to reserve this module until the reserved module is restored as shown in fig. 12.5. Thus, by preventing more than one engineer to simultaneously reserve a module, the problems associated with concurrent access are solved.



Fig. 12.5: Reserve and restore operation in configuration control

It can be shown how the changes to any object that is under configuration control can be achieved. The engineer needing to change a module first obtains a private copy of the module through a reserve operation. Then, he carries out all

necessary changes on this private copy. However, restoring the changed module to the system configuration requires the permission of a change control board (CCB). The CCB is usually constituted from among the development team members. For every change that needs to be carried out, the CCB reviews the changes made to the controlled object and certifies several things about the change:

1. Change is well-motivated.

2. Developer has considered and documented the effects of the change.

3. Changes interact well with the changes made by other developers.

4. Appropriate people (CCB) have validated the change, e.g. someone has tested the changed code, and has verified that the change is consistent with the requirement.

Configuration management tools

SCCS and RCS are two popular configuration management tools available on most UNIX systems. SCCS or RCS can be used for controlling and managing different versions of text files. SCCS and RCS do not handle binary files (i.e. executable files, documents, files containing diagrams, etc.) SCCS and RCS provide an efficient way of storing versions that minimizes the amount of occupied disk space. Suppose, a module MOD is present in three versions MOD1.1, MOD1.2, and MOD1.3. Then, SCCS and RCS stores the original module MOD1.1 together with changes needed to transform MOD1.1 into MOD1.2 and MOD1.2 to MOD1.3. The changes needed to transform each base lined file to the next version are stored and are called deltas. The main reason behind storing the deltas rather than storing the full version files is to save disk space.

The change control facilities provided by SCCS and RCS include the ability to incorporate restrictions on the set of individuals who can create new versions, and facilities for checking components in and out (i.e. reserve and restore operations). Individual developers check out components and modify them. After they have made all necessary changes to a module and after the changes have been reviewed, they check in the changed module into SCCS or RCS. Revisions are denoted by numbers in ascending order, e.g., 1.1, 1.2, 1.3 etc. It is also possible to create variants or revisions of a component by creating a fork in the development history.

8.Emerging Trends

Client-server software

A client is basically a consumer of services and server is a provider of services as shown in figure. 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 (nothing more than) 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

The client-server software architecture is a versatile, message-based and modular infrastructure that is intended to improve usability, flexibility, interoperability and scalability as compared to centralized, mainframe, time sharing computing.Popular middleware standards

Two popular middleware standards are:

• CORBA (Common Object Request Broker Architecture)

• COM/DCOM

Common Object Request Broker Architecture (CORBA)

The Common Object Request Broker Architecture (CORBA) is a specification of a standard architecture for middleware.

Using a CORBA implementation, a client can transparently invoke a method on a server object, which can be on the same machine or across a network. The middleware takes the call, and is responsible for finding an object that can implement the request, passing it the parameters, invoking its method, and returning the results of the invocation. The client does not have to be aware of where the object is located, its programming language, its operating system or any other aspects that are not part of an object’s interface.

Component Object Model (COM)

The main idea in the Component Object Model (COM) is that:

• Different vendors can sell binary components.

• Application can be developed by integrating off-the-shelf (ready to use, packaged) and proprietary (sth that is privately owned) components.

COM is a specification developed by Microsoft for building software components that can be assembled into programs or adds functionality to existing programs running on Microsoft Windows platforms.

COM runs on a single computer. The concepts used are very similar to CORBA. The components are known as binary objects. These can be generated using languages such as Visual Basic, Visual C++ etc. These languages have the necessary features to create COM components. COM components are binary objects and they exist in the form of .exe or .dll (dynamic link library).

Distributed Component Object Model (DCOM)

Distributed Component Object Model (DCOM) is the extension of the Component Object Model (COM). The restriction that clients and servers reside in the same computer is released here. So, DCOM and CORBA both operate on networked computers.

Here development is much easier as compared to CORBA development. Many of the things are transparent to the programmer such as proxy generation, service invocation etc.

Service-Oriented Architecture (SOA)

Service-oriented architecture (SOA) is a [software design](http://en.wikipedia.org/wiki/Software_design) and software architecture [design pattern](http://en.wikipedia.org/wiki/Design_pattern) based on distinct pieces of software providing application functionality as services to other applications. This is known as [service-orientation](http://en.wikipedia.org/wiki/Service-orientation). It is independent of any vendor, product or technology.

A service is a self-contained unit of functionality, such as retrieving an online bank statement. Services can be combined by other software applications to provide the complete functionality of a large software application. SOA makes it easy for computers connected over a network to cooperate. Every computer can run an arbitrary number of services, and each service is built in a way that ensures that the service can exchange information with any other service in the network without human interaction and without the need to make changes to the underlying program itself.

A service-oriented architecture is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. Some means of connecting services to each other is needed.

Software as a Service (SaaS):

Software as a Service (SaaS) is a software distribution model in which applications are hosted by a vendor or service provider and made available to customers over a network, typically the Internet.

Short for S*oftware* a*s* a S*ervice*, SaaS is a [software](http://www.webopedia.com/TERM/S/software.html) delivery method that provides access to software and its functions remotely as a Web-based service. Software as a Service allows organizations to access business functionality at a cost typically less than paying for licensed [applications](http://www.webopedia.com/TERM/A/application_software.html) since SaaS pricing is based on a monthly fee. Also, because the software is hosted remotely, users don't need to invest in additional [hardware](http://www.webopedia.com/TERM/H/hardware.html). Software as a Service removes the need for organizations to handle the installation, set-up and often daily upkeep and maintenance. Software as a Service may also be referred to as simply *hosted applications*.

Short for S*torage* a*s* a S*ervice*, SaaS is a term used to describe a [storage](http://www.webopedia.com/TERM/S/storage.html) model where a business or organization (the client) rents or leases storage space from a third-party provider. [Data](http://www.webopedia.com/TERM/D/data.html) is transferred from the client to the service provider via the [Internet](http://www.webopedia.com/TERM/I/Internet.html) and the client would then access their stored data using [software](http://www.webopedia.com/TERM/S/software.html) provided by the storage provider. The software is used to perform common tasks related to storage, such as data [backups](http://www.webopedia.com/TERM/B/backup.html) and data transfers.

Benefits of the SaaS model include:

* easier administration
* automatic updates and [patch management](http://searchenterprisedesktop.techtarget.com/definition/patch-management)
* compatibility: All users will have the same version of software.
* easier [collaboration](http://whatis.techtarget.com/definition/collaboration), for the same reason
* global accessibility.

The traditional model of software distribution, in which software is purchased for and installed on personal computers, is sometimes referred to as software as a product.