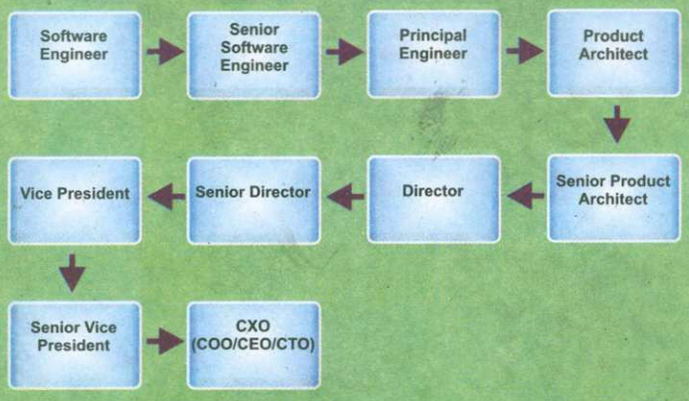
ignou
THE PEOPLE'S
UNIVERSITY

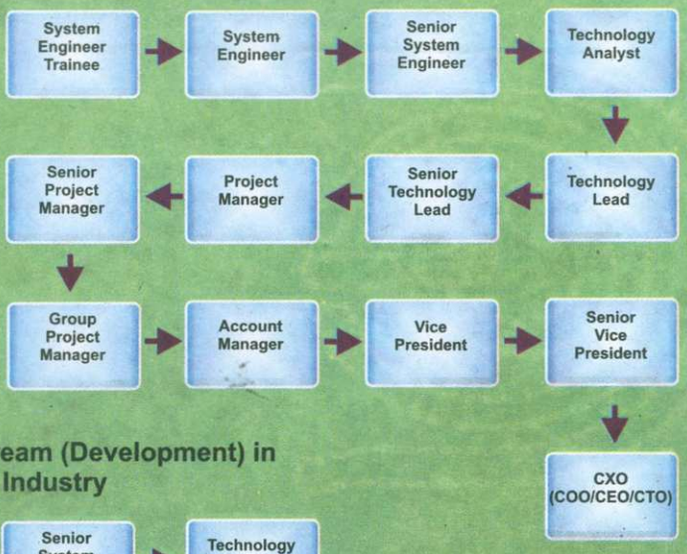
Indira Gandhi National Open University
School of Computer and Information Sciences

BCS-051
Introduction to Software Engineering

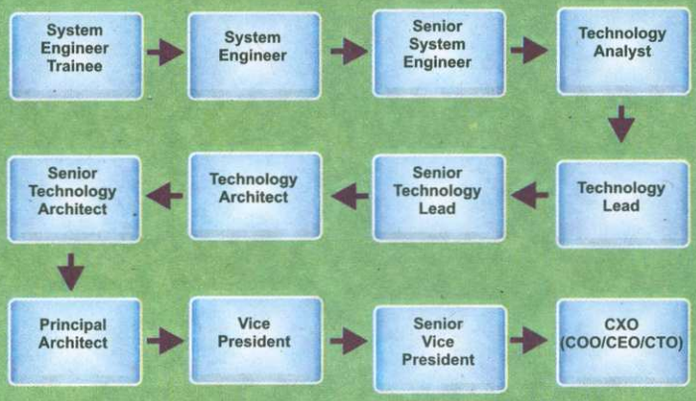
Roles for Technology Stream (Development) in IT Product Development Industry



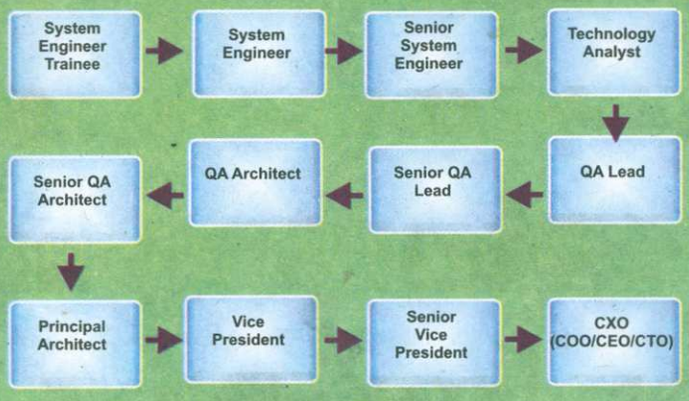
Roles for Project Management Stream (Development) in IT Services Industry



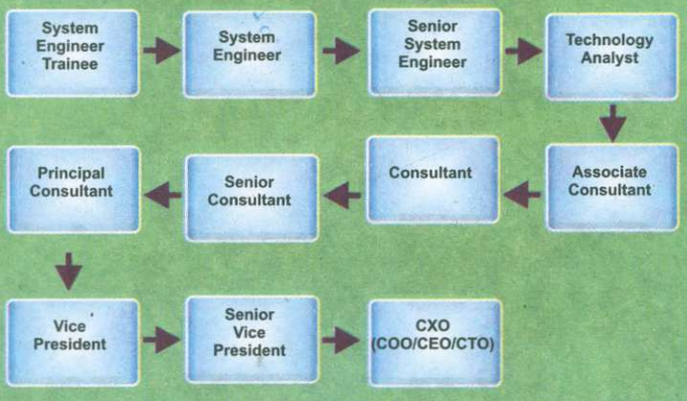
Roles for Technology Stream (Development) in IT Services Industry



Roles for QA Stream (Development) in IT Services Industry



Roles for Consulting Stream in IT Services Industry



Software Engineering Concepts

3

"शिक्षा मानव को बन्धनों से मुक्त करती है और आज के युग में तो यह लोकतंत्र की भावना का आधार भी है। जन्म तथा अन्य कारणों से उत्पन्न जाति एंव वर्तमान विषमताओं को दूर करते हुए मनुष्य को इन सबसे उपर उठाती है।"

– इन्दिरा गांधी

"Education is a liberating force, and in our age it is also a democratising force, cutting across the barriers of caste and class, smoothing out inequalities imposed by birth and other circumstances."

- Indira Gandhi

Indira Gandhi National Open University
School of Computer and Information Sciences

BCS-051
INTRODUCTION TO SOFTWARE ENGINEERING

Block

3

SOFTWARE ENGINEERING CONCEPTS

PROGRAMME DESIGN COMMITTEE

Prof. Manohar Lal
SOCIS, IGNOU, New Delhi

Prof. H. M. Gupta
Dept. of Elect. Engg.
IIT, Delhi

Prof. M. N. Doja, Dept. of CE
Jamia Millia, New Delhi

Prof. C. Pandurangan
Dept. of CSE, IIT, Chennai

Prof. I. Ramesh Babu
Dept. of CSE
Acharya Nagarjuna University
Nagarjuna Nagar (AP)

Prof. N. S. Gill
Dept. of CS, MDU, Rohtak

Prof. Arvind Kalia, Dept. of CS
HP University, Shimla

Prof. Anju Sahgal Gupta
SOH, IGNOU, New Delhi

Prof. Sujatha Verma
SOS, IGNOU, New Delhi

Prof. V. Sundarapandian
IITMK, Trivandrum

Prof. Dharamendra Kumar
Dept. of CS, GJU, Hissar

Prof. Vikram Singh
Dept. of CS, CDLU, Sirsa

Sh. Shashi Bhushan
Associate. Prof.
SOCIS, IGNOU, New Delhi

Sh. Akshay Kumar
Associate Prof.
SOCIS, IGNOU, New Delhi

Dr. P. K. Mishra, Associate Prof.
Dept. of CS, BHU, Varanasi

Dr. P. V. Suresh, Associate Prof.
SOCIS, IGNOU, New Delhi

Sh. V. V. Subrahmanyam
Associate Prof.
SOCIS, IGNOU, New Delhi

Sh. M. P. Mishra, Asst. Prof.
SOCIS, IGNOU, New Delhi

Dr. Naveen Kumar, Reader
SOCIS, IGNOU, New Delhi

Dr. Subodh Kesharwani, Asst. Prof.
SOMS, IGNOU, New Delhi

COURSE CURRICULUM DESIGN COMMITTEE

Sh. Shashi Bhushan
SOCIS, IGNOU, New Delhi

Dr. P. V. Suresh
SOCIS, IGNOU, New Delhi

Sh. V. V. Subrahmanyam
SOCIS, IGNOU, New Delhi

Dr. Naveen Kumar
SOCIS, IGNOU, New Delhi

Prof. A. K. Tripathi
Dept. of CSE, IIT(BHU),
Varanasi

Dr. S. R. N. Reddy
Dept. of CSE
IGDTUW
New Delhi

Sh. Akshay Kumar
SOCIS, IGNOU, New Delhi

Prof. Manu Sood
Dept. of CS
HP University
Shimla

Sh. M. P. Mishra
SOCIS, IGNOU, New Delhi

SOCIS FACULTY

Sh. Shashi Bhushan, Director

Sh. Akshay Kumar
Associate Professor

Sh. M. P. Mishra
Asst. Professor

Prof. Manohar Lal

Sh. V. V. Subrahmanyam
Associate Professor

Dr. Sudhansh Sharma
Asst. Professor

Dr. P. V. Suresh
Associate Professor

Dr. Naveen Kumar
Reader

PREPARATION TEAM

Content Editor:

Dr. S. R. N. Reddy
Dept. of CSE
IGDTUW
New Delhi

Language Editor:

Sh. P. Lakshmi Kantham
(Retd.), Tenali

Block Writers:

Unit 1 (adopted Unit-1 of Block-1 of MCS-034)

Dr.Ela Kumar, School of Information and Communication Technology, Gautam Buddha University

Unit 2

Sh. T. Lakshmana Kumar, Project Manager, Sri Parimala Software Services

Unit 3

Ms. Rinku Dixit, Faculty (Manav Rachna College of Engineering, Faridabad)

Course Coordinator: Dr. P. V. Suresh, SOCIS, IGNOU, New Delhi

Print Production

Mr. S. Burman	Mr. Tilak Raj	Mr. Yash Pal
Dy. Registrar (Publication)	Asstt. Registrar (Publication)	Section Officer (Publication)
MPDD, IGNOU, New Delhi	MPPD, IGNOU, New Delhi	MPDD, IGNOU New Delhi

September, 2018 (Reprint) **ISBN-978-81-266-6612-6**

Further information about the Indira Gandhi National Open University courses may be obtained from the University's office at Maidan Garhi, New Delhi-110 068.

Printed and published on behalf of the Indira Gandhi National Open University, by Registrar MPDD, IGNOU, New Delhi

Printed at: Ankur Offset Pvt. Ltd., A-54, Sector-63, Noida-201307 (U.P.)

BLOCK INTRODUCTION

This block introduces learner to Software Development Life Cycle (SDLC) and some of the SDLC models, Software Project Management (SPM) and fundamentals of Software Engineering.

Every software development project is based on some SDLC model. It is not essential that the model being used is among those available in books or elsewhere. A company may develop its own SDLC model depending on the nature of software project at hand or the domain they deal with. Every SDLC model is having its own advantages and disadvantages. It is essential to exercise utmost care while choosing a model as any error in the judgement will take you through a wrong route and the project team may not achieve the set objectives. The block introduces a SDLC model wherein it is not possible to revisit any of the earlier phases. It also introduces a SDLC model wherein it is possible to develop a model of the software to be built quickly. Then, there are other SDLC models that are introduced too. After studying the block, the learner is encouraged to go through the web trying to find any new SDLC models that are coined. It is advised that the learner study such SDLC models also to enhance the knowledge and always be in tune with the latest scientific and industry trends.

Everything is in the management. It all depends on how you manage. If things are not managed properly, then it is possible to expect a mess. Degree of mismanagement determines the point from where you need to restart. Software development is no exception and proper management of the Software projects is essential for the success of the project. There are several topics associated with Software Project Management. They include topics related to scheduling, planning, execution, risk management, etc. The concerned unit in the block introduces them. Its a field where rapid advances are in progress.

Managing the change during a Software project, maintaining the software developed and ensuring that the software developed meets the set quality standards are some of the critical topics in software engineering. Software Configuration Management (SCM) is an umbrella activity that is applied throughout the software process. Maintenance is a set of software engineering activities that occur after software has been delivered to the customer and put into operation. Quality assurance consists of the auditing and reporting functions of management. The goal of quality assurance is to provide management with the data necessary to be informed about product quality, thereby gaining insight and confidence that product quality is meeting its goals.

This block consists of three units and is organized as follows:

Unit 1 deals with Software Engineering and its models. It covers Software Development Models, Capability Maturity Models, etc.

Unit 2 deals with Software Project Management. It covers planning, monitoring, execution and control of Software projects as well as scheduling techniques.

Unit 3 deals with fundamentals of Software Engineering. It covers topics related to Software Configuration Management, Software Maintenance and Software Quality Assurance.

UNIT 1 SOFTWARE ENGINEERING AND ITS MODELS

Structure

1.0 INTRODUCTION

The field of software engineering is related to the development of software. Large software needs systematic development unlike simple programs which can be developed in isolation and there may not be any systematic approach being followed.

In the last few decades, the computer industry has undergone revolutionary changes in hardware. That is, processor technology, memory technology, and integration of devices have changed very rapidly. As the software is required to maintain compatibility with hardware, the complexity of software also has changed much in the recent past. In 1970s, the programs were small, simple and executed on a simple uniprocessor system. The development of software for such systems was much easier. In the present situation, high speed multiprocessor systems are available and the software is required to be developed for the whole organisation. Naturally, the complexity of software has increased many folds. Thus, the need for the application of engineering techniques in their development is realised. The application of engineering approach to software development lead to the evolution of the area of Software Engineering. The IEEE glossary of software engineering terminology defines the Software Engineering as:

"(a) The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software, that is, the application of engineering to software. (b) The study of approaches in (a)."

There is a difference between programming and Software Engineering. Software Engineering includes activities like cost estimation, time estimation, designing, coding, documentation, maintenance, quality assurance, testing of software etc. whereas programming includes only the coding part. Thus, it can be said that programming activity is only a subset of software development activities. The above mentioned features are essential features of software. Besides these essential features, additional features like reliability, future expansion, software reuse etc. are also considered. Reliability is of utmost importance in real time systems like flight control, medical applications, etc.

1.1 OBJECTIVES

After going through this unit, you should be able to

- define software engineering,
- understand the evolution of software engineering,
- understand the characteristics of software,
- learn about phases of software development life cycle, and
- understand software development models.

1.2 EVOLUTION OF SOFTWARE ENGINEERING

Any application on computer runs through software. As computer technologies have changed tremendously in the last five decades, accordingly, the software development has undergone significant changes in the last few decades of 20th century. In the early years, the software size used to be small and those were developed either by a single programmer or by a small programming team. The program development was dependent on the programmer's skills and no strategic software practices were present. In the early 1980s, the size of software and the application domain of software increased. Consequently, its complexity has also increased. Bigger teams were engaged in the development of Software. The software development became more bit organised and software development management practices came into existence.

In this period, higher order programming languages like PASCAL and COBOL came into existence. The use of these made programming much easier. In this decade, some structural design practices like top down approach were introduced. The concept of quality assurance was also introduced. However, the business aspects like cost estimation, time estimation etc. of software were in their elementary stages.

In the late 1980s and 1990s, software development underwent revolutionary changes. Instead of a programming team in an organisation, full-fledged software companies evolved (called software houses). A software houses primary business is to produce software. As software house may offer a range of services, including hiring out of suitably qualified personnel to work within client's team, consultancy and a complete system design and development service. The output of these companies was 'Software'. Thus, they viewed the software as a *product* and its functionality as a *process*. The concept of software engineering was introduced and Software became more strategic, disciplined and commercial. As the developer of Software and user of Software became separate organisation, business concepts like software costing, Software quality, laying of well-defined requirements, Software reliability, etc., came into existence. In this phase an entirely new computing environments based on a knowledge-based systems get created. Moreover, a powerful new concept of object-oriented programming was also introduced.

The production of software became much commercial. The software development tools were devised. The concept of Computer Aided Software Engineering (CASE) tools came into existence. The software development became faster with the help of CASE tools.

The latest trend in software engineering includes the concepts of software reliability, reusability, scalability etc. More and more importance is now given to the quality of the software product. Just as automobile companies try to develop good quality automobiles, software companies try to develop good quality Software. The software creates the most valuable product of the present era, i.e., information.

The following *Table* summarises the evolution of software:

1960s	Infancy	Machine Code
1970s	Project Years	Higher Order Languages
1980s	Project Years	Project Development
1990s	Process and Production Era	Software Reuse

The problems arising in the development of software is termed as crisis. It includes the problems arising in the process of development of software rather than software functioning. Besides development, the problems may be present in the maintenance and handling of large volumes of software. Some of the common misunderstandings regarding software development are given below:

(1) Correcting errors is easy. Though the changes in the software are possible, but, making changes in large software is extremely difficult task.

(2) By proper development of software, it can function perfectly at first time. Though, theoretically, it seems correct, but practically software undergoes many development/coding/testing passes before becoming perfect for working.

(3) Loose objective definition can be used at starting point. Once a software is developed using loose objective, changing it for specific objectives may require complete change.

(4) More manpower can be added to speed up the development. Software is developed by well coordinated teams. Any person joining it at a later stage may require extra efforts to understand the code.

Software Standards

Various terms related to software engineering are regularly standardised by organisations like IEEE (Institute of Electrical and Electronics Engineers), ANSI (American National Standards Institute), OMG (Object Management Group), CORBA (Common Object Request Broker Architecture).

IEEE regularly publishes software development standards.

OMG is international trade organisation (http://www.omg.org) and is one of the largest consortiums in the software industry. CORBA defines the standard capabilities that allow objects to interact with each other.

1.3 SOFTWARE DEVELOPMENT MODELS

Software Engineering deals with the development of software. Hence, understanding the basic characteristics of software is essential. Software is different from other engineering products in the following ways:

(1) Engineering products once developed cannot be changed. To modifications the product, redesigning and remanufacturing is required. In the case of software, ultimately changes are to be done in code for any changes to take effect.

(2) The Other Engineering products are visible but the software as such is not visible. That's why, it is said that software is developed, but not manufactured. Though, like other products, it is first designed, then produced, it cannot be manufactured automatically on an assembly line like other engineering products. Nowadays, CASE (Computer Aided Software Engineering) tools are available for software development. Still it depends on the programmer's skill and creativity. The creative skills of the programmer is difficult to quantify and standardise. Hence, the same software developed by different programmers may take varying amount of time, resources and may have variable cost.

(3) Software does not *fail* in the traditional sense. The engineering products has wear and tear in the operation. Software can be run any number of times without wear and tear. The software is considered as *failed* if:

(a) It does not operate correctly.

(b) Does not provide the required number of features.

(4) Engineering products can be perfectly designed, but in the case of software, however good the design, it can never be 100% error free. Even the best quality software is not completely error free. A software is called good quality software if it performs the required operation, even if it has a few errors.

(5) The testing of normal engineering products and software engineering products are on different parameters. In the former, it can be full load testing, etc., whereas in the case of software, testing means identification of test cases in which software may fail. Thus, testing of software means running of software for different inputs. By testing, the presence of errors is identified.

(6) Unlike most of the other engineering products, software can be reused. Once a piece of code is written for some application, it can be reused.

(7) The management of software development projects is a highly demanding task, since it involves the assessment of the developers creative skills. The estimation regarding the time and cost of software needs standardisation of developers creativity, which can be a variable quantity. It means that software projects cannot be managed like engineering products. The correction of a bug in the case of software may take hours, but, it may not be the case with normal engineering products.

(8) The Software is not vulnerable to external factors like environmental effects. But the same external factors may harm hardware. The hardware component may be replaced with spare parts in the case of failure, whereas the failure of a software component may indicate the errors in design.

Thus, the characteristics of software are quite different from other engineering products. Hence, the software industry is quite different from other industries.

1.3.1 Importance of Software Engineering

As the application domains of software are becoming complicated and design of big software without a systematic approach is virtually impossible, the field of software engineering is increasingly gaining importance. It is now developing like an industry. Thus, the industry has to answer following or similar queries of clients:

(1) What is the best approach to design of software?

(2) Why the cost of software is too high?

(3) Why can't we find all errors?

(4) Why is there always some gap between claimed performance and actual performance?

To answer all such queries, software development has adopted a systematic approach.

Software development should not remain an art. Scientific basis for cost, duration, risks, defects etc. are required. For quality assurance, product qualities and process qualities and must be made measurable as far as possible by developing metrics for them.

1.3.2 Various Development Models

The following are some of the models adopted to develop software:

(i) Build and Fix Model

It is a simple two phase model. In one phase, code is developed and in another, code is fixed.

Figure 1.1 depicts the Build and Fix model.

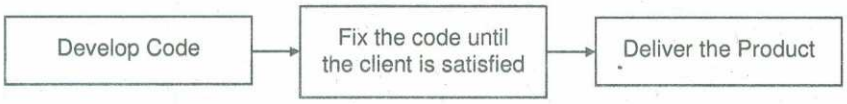


Figure 1.1 : Build and Fix Model

(ii) Waterfall Model

It is the simplest, oldest and most widely used process model. In this model, each phase of the life cycle is completed before the start of a new phase. It is actually the first engineering approach of software development.

Figure 1.2 depicts Water Fall Model.

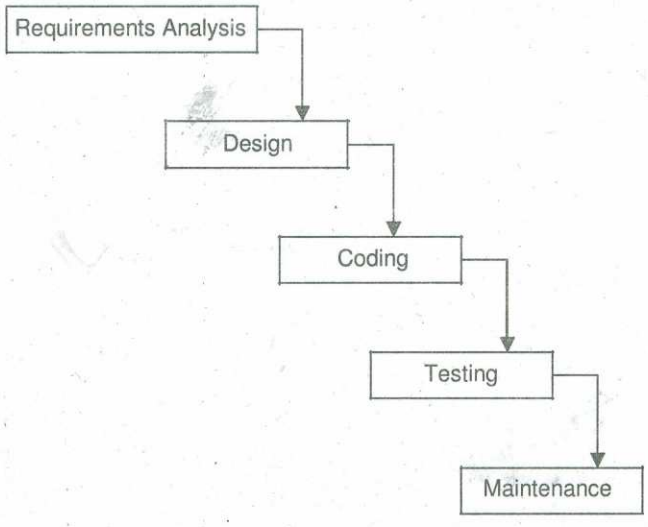


Figure 1.2 : Water Fall Model

The functions of various phases are discussed in software process technology.

The waterfall model provides a systematic and sequential approach to software development and is better than the build and fix approach. But, in this model, complete requirements should be available at the time of commencement of the project, but in actual practice, the requirements keep on originating during different phases. The water fall model can accommodate the new requirements only in maintenance phase. Moreover, it does not incorporate any kind of risk assessment. In waterfall model, a working model of software is not available. Thus, there is no methods to judge the problems of software in between different phases.

A slight modification of the waterfall model is a model with feedback. Once software is developed and is operational, then the feedback to various phases may be given.

Figure 1.3 depicts the Water Fall Model with feedback.

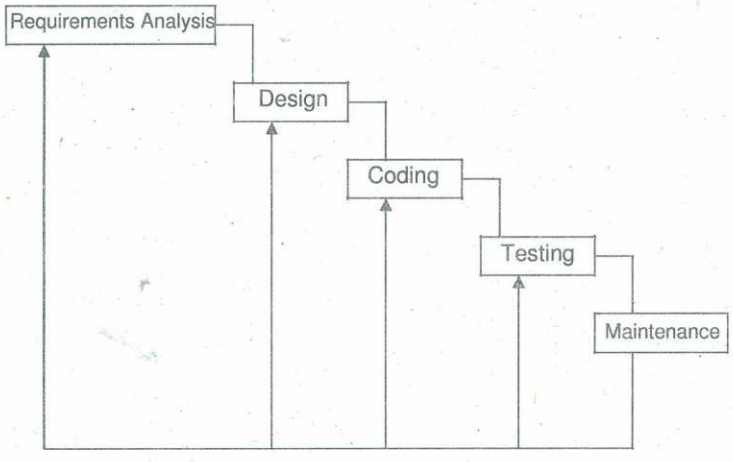


Figure 1.3 : Water Fall Model with Feedback

(iii) Iterative Enhancement Model

This model was developed to remove the shortcomings of waterfall model. In this model, the phases of software development remain the same, but the construction and delivery is done in the iterative mode. In the first iteration, a less capable product is developed and delivered for use. This product satisfies only a subset of the requirements. In the next iteration, a product with incremental features is developed. Every iteration consists of all phases of the waterfall model. The complete product is divided into releases and the developer delivers the product release by release.

Figure 1.4 depicts the Iterative Enhancement Model.

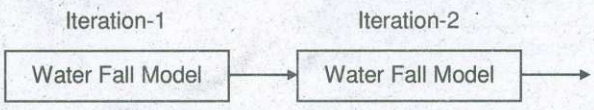


Figure 1.4 : Iterative Enhancement Model

This model is useful when less manpower is available for software development and the release deadlines are tight. It is best suited for in-house product development, where it is ensured that the user has something to start with. The main disadvantage of this model is that iteration may never end, and the user may have to endlessly wait for the final product. The cost estimation is also tedious because it is difficult to relate the software development cost with the number of requirements.

(iv) Prototyping Model

In this model, a working model of actual software is developed initially. The prototype is just like a sample software having lesser functional capabilities and low reliability and it does not undergo through the rigorous testing phase. Developing a working prototype in the first phase overcomes the disadvantage of the waterfall model where the reporting about serious errors is possible only after completion of software development.

The working prototype is given to the customer for operation. The customer, after its use, gives the feedback. Analysing the feedback given by the customer, the developer refines, adds the requirements and prepares the final specification document. Once the prototype becomes operational, the actual product is developed using the normal waterfall model. Figure 1.5 depicts the prototyping model.

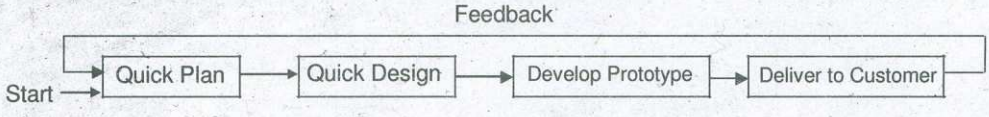


Figure 1.5 : Prototyping Model

The prototype model has the following features:

(i) It helps in determining user requirements more deeply.

(ii) At the time of actual product development, the customer feedback is available.

(iii) It does consider any types of risks at the initial level.

(v) Spiral Model

This model can be considered as the model, which combines the strengths of various other models. Conventional software development processes do not take uncertainties into account. Important software projects have failed because of unforeseen risks. The other models view the software process as a linear activity whereas this model considers it as a spiral process. This is made by representing the iterative development cycle as an expanding spiral.

The following are the primary activities in this model:

- **Finalising Objective:** The objectives are set for the particular phase of the project.
- **Risk Analysis:** The risks are identified to the extent possible. They are analysed and necessary steps are taken.
- **Development:** Based on the risks that are identified, an SDLC model is selected and is followed.
- **Planning:** At this point, the work done till this time is reviewed. Based on the review, a decision regarding whether to go through the loop of spiral again or not will be decided. If there is need to go, then planning is done accordingly.

In the spiral model, these phases are followed iteratively. Figure 1.6 depicts the Boehm's Spiral Model (IEEE, 1988).

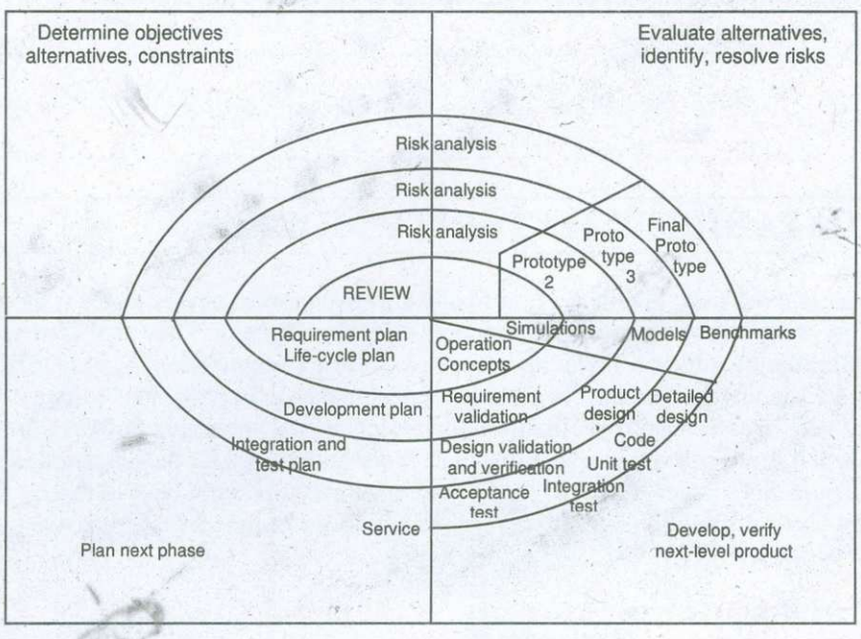


Figure 1.6: Spiral Model

In this model Software development starts with lesser requirements specification, lesser risk analysis, etc. The radical dimension this model represents cumulative cost. The angular dimension represents progress made in completing the cycle.

The inner cycles of the spiral model represent early phases of requirements analysis and after prototyping of software, the requirements are refined.

In the spiral model, after each phase a review is performed regarding all products developed upto that point and plans are devised for the next cycle. This model is a realistic approach to the development of large scale software. It suggests a systematic approach according to classical life cycle, but incorporates it into iterative framework. It gives a direct consideration to technical risks. Thus, for high risk projects, this model is very useful. The risk analysis and validation steps eliminate errors in early phases of development.

(vi) RAD Approach

As the name suggests, this model gives a quick approach for software development and is based on a linear sequential flow of various development processes. The software is constructed on a component basis. Thus multiple teams are given the task of different component development. It increases the overall speed of software development. It gives a fully functional system within very short time. This approach emphasises the development of reusable program components. It follows a modular approach for development. The problem with this model is that it may not work when technical risks are high.

☞ **Check Your Progress 1**

(1) Indicate various problems related with software development.

.....

.....

.....

(2) Give a comparative analysis of various types of software process models.

.....

.....

.....

(3) What are various phases of software development life cycle?

.....

.....

.....

1.4 CAPABILITY MATURITY MODELS

The process models are based on various software development phases whereas the capability models have an entirely different basis of development. They are based upon the capabilities of software. It was developed by Software Engineering Institute (SEI). In this model, significant emphasis is given to the techniques to improve the "software quality" and "process maturity". In this model a strategy for improving Software process is devised. It is not concerned which life cycle mode is followed for development. SEI has laid guidelines regarding the capabilities an organisation should have to reach different levels of process maturity. This approach evaluates the global effectiveness of a software company.

1.4.1 Maturity Level

It defines five maturity levels as described below. Different organisations are certified for different levels based on the processes they follow.

Level 1 (Initial): At this maturity level, software is developed an ad hoc basis and no strategic approach is used for its development. The success of developed software entirely depend upon the skills of the team members. As no sound engineering approach is followed, the time and cost of the project are not critical issues. In Maturity Level 1 organisations, the software process is unpredictable, because if the developing team changes, the process will change. The testing of software is also very simple and accurate predictions regarding software quality are not possible. SEI's assessment indicates that the vast majority of software organisations are Level 1 organisations.

Level 2 (Repeatable): The organisation satisfies all the requirements of Level 1. At this level, basic project management policies and related procedures are established. The institutions achieving this maturity level learn with experience of earlier projects and reutilise the successful practices in on- going projects. The effective process can be characterised as practised, documented, implemented and trained. In this maturity level, the manager provides quick solutions to the problem encountered in software development and corrective action is immediately taken. Hence, the process of development is much disciplined in this maturity level. Thus, without measurement, sufficiently realistic estimates regarding cost, schedules and functionality are performed. The organisations of this maturity level have installed basic management controls.

Level 3 (Defined): The organisation satisfies all the requirements of Level 2. At this maturity level, the software development processes are well defined, managed and documented. Training is imparted to staff to gain the required knowledge. The standard practices are simply tailored to create new projects.

Level 4 (Managed): The organisation satisfies all the requirements of Level 3. At this level quantitative standards are set for software products and processes. The project analysis is done at integrated organisational level and collective database is created. The performance is measured at integrated organisation level. The Software development is performed with well defined instruments. The organisation's capability at Level 4 is "predictable" because projects control their products and processes to ensure their performance within quantitatively specified limits. The quality of software is high.

Level 5 (Optimising): The organisation satisfies all the requirements of Level 4. This is last level. The organisation at this maturity level is considered almost perfect. At this level, the entire organisation continuously works for process improvement with the help of quantitative feedback obtained from lower level. The organisation analyses its weakness and takes required corrective steps proactively to prevent the errors. Based on the cost benefit analysis of new technologies, the organisation changes their Software development processes.

1.4.2 Key Process Area

The SEI has associated key process areas (KPAs) with each maturity level. The KPA is an indicative measurement of goodness of software engineering functions like project planning, requirements management, etc. The KPA consists of the following parameters:

Goals: Objectives to be achieved.

Commitments: The requirements that the organisation should meet to ensure the claimed quality of product.

Abilities: The capabilities an organisation has.

Activities: The specific tasks required to achieve KPA function.

Methods for varying implementation: It explains how the KPAs can be verified.

18 KPAs are defined by SEI and associated with different maturity levels. These are described below:

Level 1 KPAs : There is no key process area at Level 1.

Level 2 KPAs:

(1) **Software Project Planning:** Gives concrete plans for software management.

(2) **Software Project Tracing and Oversight:** Establish adequate visibility into actual process to enable the organisation to take immediate corrective steps if Software performance deviates from plans.

(3) **Requirements Management:** The requirements are well specified to develop a contract between developer and customer.

(4) **Software Subcontract Management:** Select qualified software subcontractors and manage them effectively.

(5) **Software Quality Assurance (SQA):** To Assure the quality of developed product.

(6) **Software Configuration Management (SCM):** Establish and maintain integrity through out the lifecycle of project.

Level 3 KPAs:

(1) **Organisation Process Focus (OPF):** The organisations responsibility is fixed for software process activities that improve the ultimate software process capability.

(2) **Training Program (TP):** It imparts training to develop the skills and knowledge of organisation staff.

(3) **Organisation Process Definition (OPD):** It develops a workable set of software process to enhance cumulative long term benefit of organisation by improving process performance.

(4) **Integrated Software Management (ISM):** The software management and software engineering activities are defined and a tailor made standard and software process suiting to organisations requirements is developed.

(5) **Software Product Engineering (SPE):** Well defined software engineering activities are integrated to produce correct, consistent software products effectively and efficiently.

(6) **Inter group co-ordination (IC):** To satisfy the customer's needs effectively and efficiently, Software engineering groups are created. These groups participate actively with other groups.

(7) **Peer reviews (PR):** They remove defects from software engineering work products.

Level 4 KPAs:

(1) **Quantitative Process Management (QP):** It defines quantitative standards for software process.

(2) **Software Quality Management (SQM):** It develops quantitative understanding of the quality of Software products and achieves specific quality goals.

Level 5 KPAs:

(1) **Defect Prevention (DP):** It discovers the causes of defects and devises the techniques which prevent them from recurring.

(2) **Technology Change Management (TCM):** It continuously upgrades itself according to new tools, methods and processes.

(3) **Process Change Management (PCM):** It continually improves the software processes used in organisation to improve software quality, increase productivity and decrease cycle time for product development.

☞ Check Your Progress 2

(1) What are different levels of capability maturity model?

.....

.....

.....

(2) What is the level of CMM with which no KPA is associated?

.....

.....

.....

1.5 SOFTWARE PROCESS TECHNOLOGY

The software industry considers software development as a process. According to Booch and Rumbaugh, "A process defines who is doing what, when and how to reach a certain goal?" Software engineering is a field which combines process, methods and tools for the development of software. The concept of process is the main step in the software engineering approach. Thus, a software process is a set of activities. When those activities are performed in specific sequence in accordance with ordering constraints, the desired results are produced. A software development project requires two types of activities viz., development and project management activities. These activities together comprise of a software process. As various activities are being performed in software process, these activities are categorized into groups called phases. Each phase performs well defined activities.

The various steps (called phases) which are adopted in the development of this process are collectively termed as Software Development Life Cycle (SDLC). The various phases of SDLC are discussed below. Normally, these phases are performed lineally or circularly, but it can be changed according to project as well. The software is also considered as a product and its development as a process. Thus, these phases can be termed as Software Process Technology. In general, different phases of SDLC are defined as following:

- Requirements Analysis
- Design
- Coding
- Software Testing
- Maintenance.

Let us discuss these steps in detail.

Requirements Analysis

Requirements describe the "What" of a system. The objectives which are to be achieved in Software process development are the requirements. In the requirements analysis phase, the requirements are properly defined and noted down. The output of this phase is SRS (Software Requirements Specification) document written in natural language. According to IEEE, requirements analysis may be defined as (1) the process of studying user's needs to arrive at a definition of system hardware and software requirements (2) the process of studying and refining system hardware or software requirements.

Design

In this phase, a logical system is built which fulfils the given requirements. Design phase of software development deals with transforming the customer's requirements into a logically working system. Normally, design is performed in the following two steps:

(i) **Primary Design Phase:** In this phase, the system is designed at block level. The blocks are created on the basis of analysis done in the problem identification phase. Different blocks are created for different functions emphasis is put on minimising the information flow between blocks. Thus, all activities which require more interaction are kept in one block.

(ii) **Secondary Design Phase:** In the secondary design phase the detailed design of every block is performed.

The input to the design phase is the Software Requirements Specification (SRS) document and the output is the Software Design Document (SDD). The general tasks involved in the design process are the following:

(i) Design various blocks for overall system processes.

(ii) Design smaller, compact, and workable modules in each block.

(iii) Design various database structures.

(iv) Specify details of programs to achieve desired functionality.

(v) Design the form of inputs, and outputs of the system.

(vi) Perform documentation of the design.

(vii) System reviews.

The Software design is the core of the software engineering process and the first of three important technical activities, viz., design, coding, and testing that are required to build software. The design should be done keeping the following points in mind.

(i) It should completely and correctly describe the system.

(ii) It should precisely describe the system. It should be understandable to the software developer.

(iii) It should be done at the right level.

(iv) It should be maintainable.

The following points should be kept in mind while performing the design:

(i) **Practicality:** This ensures that the system is stable and can be operated by a person of average intelligence.

(ii) **Efficiency:** This involves accuracy, timeliness and comprehensiveness of system output.

(iii) **Flexibility:** The system could be modifiable depending upon changing needs of the user. Such amendments should be possible with minimum changes.

(iv) **Security:** This is an important aspect of design and should cover areas of hardware reliability, fall back procedures, security of data and provision for detection of fraud.

Coding

The input to the coding phase is the SDD document. In this phase, the design document is coded according to the module specification. This phase transforms the SDD document into a high level language code. At present major software companies adhere to some well specified and standard style of coding called coding standards. Good coding standards improve the understanding of code. Once a module is developed, a check is carried out to ensure that coding standards are followed. Coding standards generally give the guidelines about the following:

(i) Name of the module

(ii) Internal and External documentation of source code

(iii) Modification history

(iv) Uniform appearance of codes.

Testing is the process of running the software on manually created inputs with the intention to find errors. In the process of testing, an attempt is made to detect errors, to correct the errors in order to develop error free software. The testing is performed keeping the user's requirements in mind and before the software is actually launched on a real system, it is tested. Testing is the process of executing a program with the intention of finding errors.

Normally, while developing the code, the software developer also carries out some testing. This is known as debugging. This unearths the defects that must be removed from the program. Testing and debugging are different processes. Testing is meant for finding the existence of defects while debugging stands for locating the place of errors and correcting the errors during the process of testing. The following are some guidelines for testing:

(i) Test the modules thoroughly, cover all the access paths, generate enough data to cover all the access paths arising from conditions.

(ii) Test the modules by deliberately passing wrong data.

(iii) Specifically create data for conditional statements. Enter data in test file which would satisfy the condition and again test the script.

(iv) Test for locking by invoking multiple concurrent processes.

The following objectives are to be kept in mind while performing testing:

(i) It should be done with the intention of finding the errors.

(ii) Good test cases should be designed which have a probability of finding, as yet undiscovered error.

(iii) A success test is one that uncovers yet undiscovered error(s).

The following are some of the principles of testing:

(i) All tests should be performed according to user requirements.

(ii) Planning of tests should be done long before testing.

(iii) Starting with a small test, it should proceed towards large tests. The following are different levels of testing:

Large systems are built out of sub-systems, subsystems are made up of modules, modules of procedures and functions. Thus in large systems, the testing is performed at various levels, like unit level testing, module level testing, subsystem level, and system level testing.

Thus, testing is performed at the following levels. In all levels, the testing are performed to check interface integrity, information content, performance.

The following are some of the strategies of testing: This involves design of test cases. Test case is set of designed data for which the system is tested. Two testing strategies are present.

(i) **Code Testing:** The code testing strategy examines the logic of the system. In this, the analyst develops test cases for every instruction in the code. All the paths in the program are tested. This test does not guarantee against software failures. Also, it does not indicate whether the code is according to requirements or not.

(ii) **Specification Testing:** In this, testing with specific cases is performed. The test cases are developed for each condition or combination of conditions and submitted for processing.

The objective of testing is to design test cases that systematically uncover different classes of errors and do so with the minimum amount of time and effort. Testing cannot show the absence of errors. It can only find the presence of errors. The test case design is as challenging as software development. Still, however effective the design is, it cannot remove 100% errors. Even, the best quality software are not 100 % error free. The reliability of software is closely dependent on testing.

Some testing techniques are the black box and the white box methods.

White Box Testing: This method, also known as glass box testing, is performed early in the testing process. Using this, the software engineer can derive a tests that guarantees that all independent paths within the module have been exercised at least once. It has the following features:

(i) Exercise all logical decisions on their true and false sides.

(ii) Execute all loops at their boundaries and within their operational bounds.

(iii) Exercise internal data structures to assure their validity.

Black Box Testing: This is applied during the later stage of testing. It enables the software developer to derive a set of input conditions that will fully exercise the functional requirements of a program. It enables him to find errors like incorrect or missing functions, interface errors, data structures or external data base access errors and performance errors etc.

Maintenance

Maintenance in the normal sense means correcting the problems caused by wear and tear, but software maintenance is different. Software is either wrong in the beginning or later as some additional requirements have been added. Software maintenance is done because of the following factors.

(i) To rectify the errors which are encountered during the operation of software.

(ii) To change the program function to interface with new hardware or software.

(iii) To change the program according to increased requirements.

There are three categories of maintenance:

(i) Corrective Maintenance

(ii) Adaptive Maintenance

(iii) Perfective Maintenance

Software maintenance is a very broad activity that includes error correction, enhancement of capabilities, deletion of obsolete capabilities, and optimisation [STEP 78]. Hence, once the software becomes operational, whatever changes are done are termed as maintenance. As the software requirement change continuously, maintenance becomes a continuous process. In practice, the cost of software maintenance is far more than the cost of software development. It accounts for 50% to 80% of the total system development costs. The Software maintenance has the following problems:

(i) It is very cumbersome to analyse and understand the code written by somebody.

(ii) No standards for maintenance have been developed and the area is relatively unexplored area.

(iii) Few tools and techniques are available for maintenance.

(iv) It is viewed as a necessary evil and delegated to junior programmers.

The various phases of the software development life cycle are tightly coupled, and the output of one phase governs the activity of the subsequent phase. Thus, all the phases need to be carefully planned and managed and their interaction requires close monitoring. The project management becomes critical in larger systems.

1.6 SUMMARY

Software engineering covers the entire range of activities used to develop software. The activities include requirements analysis, program development using some recognised approach like structured programming, testing techniques, quality assurance, management and implementation and maintenance. Further, software engineering expects to address problems which are encountered during software development.

1.7 ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress 1

(1) The following are major problems related with software development:

- There may be multiple models suitable to do project. How to decide the best one?
- The understanding of user requirements may be incomplete.
- The problem of less documentation or minimal structured techniques may be there.
- The last minute changes in implementation issues give rise to problems elated with hardware.

(2) Out of all SDLC models, the most popular one is waterfall model. But, it insists on having a complete set of requirements before commencement of design. It is often difficult for the customer to state all requirements easily. The iterative enhancement model, though better than waterfall model, in customised software development where the client has to provide and approve the specification, it may lead to time delays for software development. Prototype model provides better understanding of customer's needs and is useful for large systems, but, in this, the use of inefficient inaccurate or dummy functions may produce undesired results. The spiral model accommodates good features of other models. In this, risk analysis and validation steps eliminate errors in early phases of development. But, in this model, there is a lack of explicit process guidance in determining objectives.

(3) Various phases of software development life cycle are:

- Requirement analysis
- Design
- Coding
- Testing
- Maintenance

Check Your Progress 2

(1) Initial, Repeatable, Defined, Managed, Optimizable.

(2) Level-1.

1.8 FURTHER READINGS

Software Engineering, Sixth Edition, 2001, Ian Sommerville; Pearson Education.

Software Engineering – A Practitioner's Approach, Roger S. Pressman; McGraw-Hill International Edition.

Reference websites

http://www.rspa.com

http://www.ieee.org

http://standards.ieee.org

UNIT 2 SOFTWARE PROJECT MANAGEMENT

Structure

2.0 INTRODUCTION

In today's world, Software Project Management is an Art. It is "the process of planning, organizing, staffing, monitoring, controlling and leading a software project". Another definition for SPM (Software Project Management) is "the application of knowledge, skills, tools, and techniques to project activities to meet project requirements". Every software project must have a manager who leads the development team and is the interface with the initiators, suppliers and senior management. Software development is a complex process involving such activities as domain analysis, requirements specification, communication with the customers and end-users, designing and producing different artifacts, adopting new paradigms and technologies, evaluating and testing software products, installing and maintaining the application at the end-user's site, providing customer support, organizing end-user training, visualizing potential upgrades and negotiating about them with the customers, and many more.

In order to keep everything under control, eliminate delays, always stay within the budget, and prevent project runaways, that is, situations in which cost and time exceed what was planned, software project manager must exercise control and guidance over the development team throughout the project's lifecycle. In doing so, they apply a number of

tools of both economic and managerial nature. The first category of tools include budgeting, periodic budget monitoring, user chargeback mechanism, continuous cost/benefit analysis, and budget deviation analysis. The managerial toolbox includes both long-range and short-term planning, schedule monitoring, feasibility analysis, software quality assurance, organizing project steering committees, and the like. Software Project Management has to address many challenges such as resources, security, defects, etc. as given in Figure 2.1.

The skillful integration of software technology, economics and human relations in the specific context of a software project is not an easy task. The software project is a highly people-intensive effort that spans a very lengthy period, with fundamental implications on the work and performance of many different classes of people. The problems related to software project, if not handled properly, will result into poor quality, loss of reputation, failure to deliver, etc. as shown in Figure 2.2.

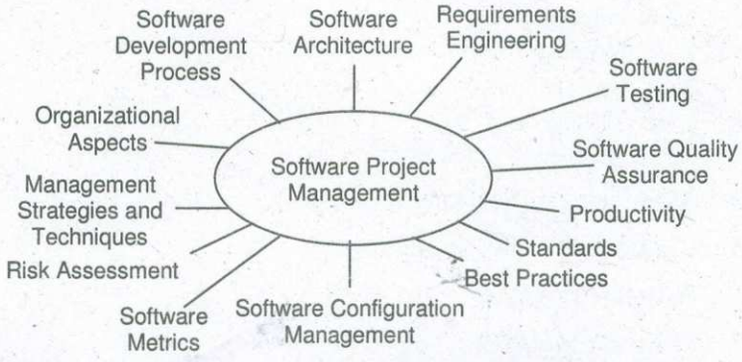


Figure 2.1: Set of Problems to be addressed by Software Project Management

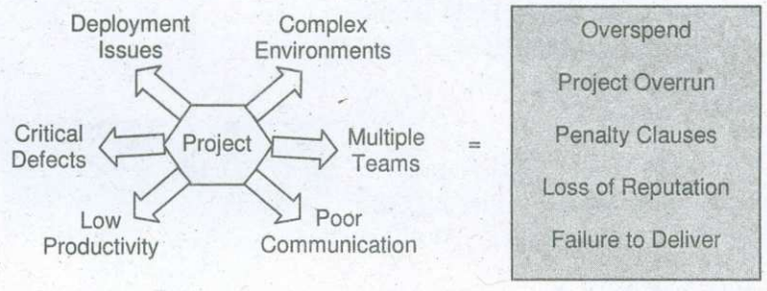


Figure 2.2 : Problems of Software Project

The project triangle (Figure 2.3) is also known as the "iron triangle" and, less poetically, the "triple constraints." Whatever you call it, it amounts to the same thing: You can't change a project's budget, schedule, or scope without affecting at least one of the other two parts. Balancing the top three project management areas (project scope, project time and project cost) is one of the biggest tasks of the project manager. Basically, if you pull one end of the triangle, one of the other ends needs to be pulled/extended to restore balance. On the other hand, if you "push" on one end, another end will need to be pushed/reduced or pulled/extended in order to bring the triangle back into balance. Decisions affecting time, scope and/or cost must be carefully considered as they are likely to have implications on overall project quality, and customer satisfaction.

Figure 2.3: The Project Triangle

2.1 OBJECTIVES

After going through this unit, you should be able to

- know the various aspects of project management like planning, monitoring, controlling and execution,
- know the reasons for major software project failure,
- know the importance of Software Metrics and also various metrics for project monitoring and control, and
- understand the use of PERT and Gantt charts in project management.

2.2 PLANNING OF SOFTWARE PROJECTS

A project plan should be based on the project requirements and developed estimates, and is a formal, consistent and approved document that provides for the essential project guidelines. Generally, the purpose of the document is to support management and control of the project execution. The plan should cover all phases of the project and should ensure that all involved plans are consistent with each other.

There is no single reason for a software engineering project to have a project plan. A project plan usually contains several parts produced in order to help the project team with their project. The main objectives with a project plan are the following:

- Guide project execution.
- Document project planning assumption.
- Document project planning decisions regarding alternatives chosen.
- Facilitate communication among stakeholders,
- Define key management reviews as to content, extent, and timing, and
- Provide a baseline for progress measurement and project control.

The project plan is generally developed in the initial phase of the project and needs to be reviewed and agreed upon by all concerned persons. However, the plan is expected to change over time and is updated each time the actual progress differs from the plan or when project conditions change which may require new approaches. A carefully prepared project plan, if properly followed and committed to, should lead to a successful project and eliminate many of the pitfalls inherent in the project management process. It provides leadership vision and facilitates for management to utilize available resources efficiently.

2.2.1 Different Types of Project Plans

In addition to the main software project plan, different types of specific plans may be developed to support the main plan in different areas. Examples of such plans may be the following:

(1) **Quality plan** – includes the quality procedures and standards that concern the project.

(2) **Validation plan** – covers approaches, resources and schedule involved in the system validation.

(3) **Configuration management plan** – consists of the configuration management procedures and structures to support the project.

(4) **Maintenance plan** – predicts the maintenance requirements, maintenance costs and the effort required.

(5) **Staff development plan** – includes how available skills and experience will be developed.

2.2.2 An Overview of the 'Step Wise' Framework

The step wise framework is shown in Figure 2.4.

Step 0: Select project

Step 1: Establish project scope and objectives

- Identify objectives and measures of effectiveness in meeting them.

  What are we trying to achieve? How do we know if we have succeeded?
- Establishment of a project authority.

  Where is the overall authority for the project?
- Identify all stakeholders in the project and their interests.

  Who will be affected by the project and who will need to be involved?
- Modify objectives in the light of stakeholder analysis.

  Do we need to do things to win the commitment of stakeholders?
- Establishment of methods of communications with all parties.

  How do we keep in touch?

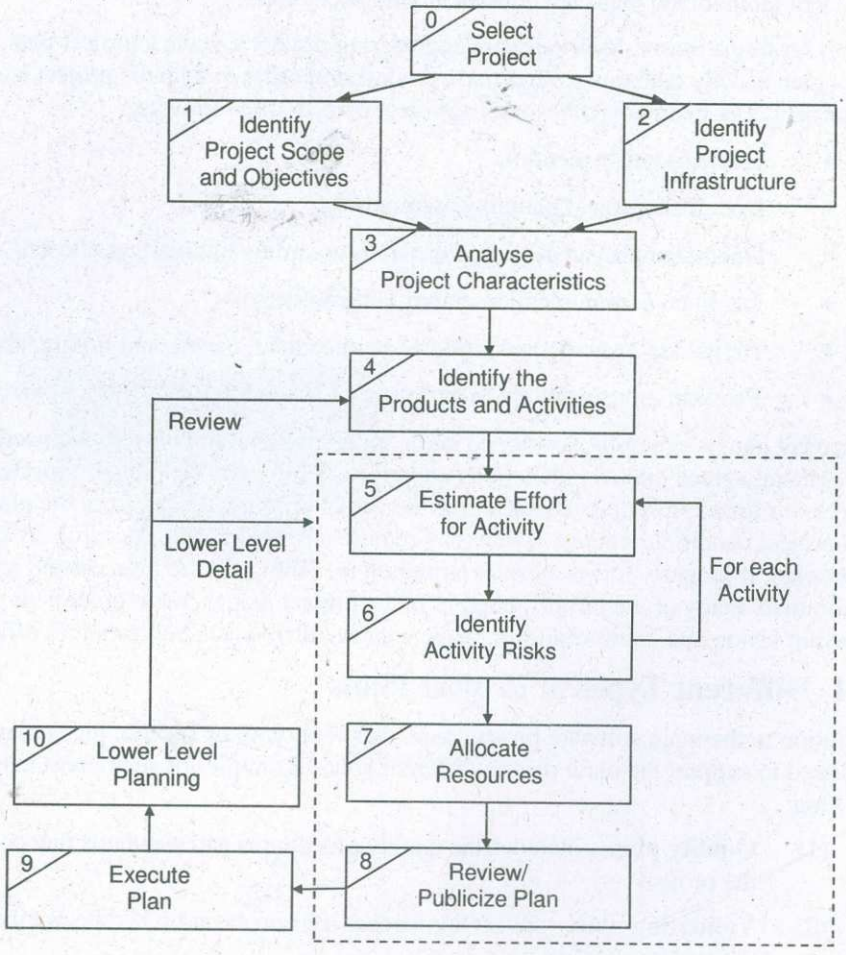


Figure 2.4: The Stepwise Framework

Step 2: Establish project infrastructure

- Establish relationship between project and strategic planning.

  Why do they want this project?'
- Identify Installation standards and procedures.

  What standards do we have to follow?

- Identify project team organization

  What is the organizational framework?

Step 3: Analysis of project characteristics

- Distinguish the project as either objective or product based.

  Is there more than one way of achieving success?

- Analyze other project characteristics (including quality based ones).

  What is different about this project?

- Identify high level project risks.

  What could go wrong?

- Take into account user requirements concerning implementation.
- Select general life cycle approach.

  What is the best approach? Increment? Prototype? Waterfall?

- Review overall resource estimates.

  Does all this make us change our mind about probable costs?

Step 4: Identify project products and activities

- Identify and describe project products (including quality criteria).

  What sort of things do we have to create?

- Document generic product flows.

  In what order do we produce them?

- Recognize product instances.

  Can we identify particular individual cases of each type of product? What modules do we have to code?

- Produce ideal activity network.
- Modify ideal activity network to take into account need for stages and checkpoints.

Step 5: Estimate effort for each activity

- Estimate effort for activity.

  How long will this activity take?

- Revise plan to create controllable activities.

  Do we need to combine activities or split them to get to a convenient size?

Step 6: Identify activity risks

- Identify and quantify activity-based risks.

  What could go wrong with this activity?

- Plan risk reduction and contingency measures where appropriate.

  How can we stop it going wrong? What do we do if it goes wrong anyway?

- Adjust plans and estimates to take into account risks.

Step 7: Allocate resources

- Identify and allocate resources.

  What resources do we need for this activity?

- Revise plans and estimates to take into account of the resource constraints. When will the resources be available?

Step 8: Review/publicize plan

- Review quality aspects of project plan.
- Complete/review documentation of plan.
- Publicize plan and obtain agreement of parties to project.

Step 9/10: Execute plan

Step 10: Plan at lower levels

This may require the reiteration of the planning process at a lower level.

2.3 EXECUTION OF SOFTWARE PROJECTS

A project development methodology is more or less the same for every software development project. Each part or point of the methodology is a unique approach to the overall development. The technique comprises of a few vital steps which are defined and detailed below (Figure 2.5).

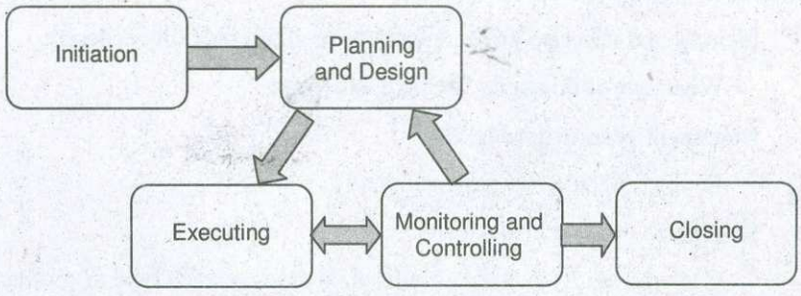


Figure 2.5: Software Project Management Life Cycle

2.3.1 Initiation

Project analysis is a most critical phase of web or software development project execution. The better a project is analyzed, the better it is executed. The analysis report serves as the base of the entire execution. The client's requirements, the project goals, the future scope of the project, the available resources, the required skills, the required tools, the available technology, the investment, the pricing and the time span are the subjects of analysis in this phase. Project analysts along with project managers perform this task ensuring attention to details. Risks and challenges associated with the project are analyzed too.

2.3.2 Planning

Once the delivery deadline is fixed up based on the project analysis report, the project plan is chalked out. Planning is a critical requirement to decide on the mode and methodology of executing the project. The better is the planning; the more efficient is the project execution. Of the several development models such as Agile Model, Spiral Model, Iterative Model and Waterfall Model, the most suitable is selected in keeping with the complexity level of the project. According to the plan, the tasks at different phases of the execution are allocated to the development team and divided into different units. If needed, a special team is built of professionals having the technological skills required for the project.

2.3.3 Execution

Project execution is the most crucial phase which demands for attention and dedication on part of the developers. The selected project development model is deployed in this process. With continuous flow of the project execution phases, the methodology is

followed to the point. Once a unit of the team is done with the development part assigned to it, the project is passed to the next unit for the next part of the development. The project manager leading the team acts as a project coordinator too. He ensures precise coordination among the units of the team to get the project executed timely as was planned. This phase is also referred to as "Construction". Quality project management plays instrumental in it.

2.3.4 Testing

This is the pre-delivery phase of the project execution cycle. The developed project is tested against a set of quality parameters before delivery. Testing is conducted through the application of a cutting-edge technology. The functionality of each component of the developed project is tested on multiple various levels. Then, the inter-functionality of the integrated components of the software is checked through integration testing. If any functional or technical issue is noticed, it is resolved to ensure the precision and accuracy of the software.

2.3.5 Acceptance and Delivery

If the outcome is positive on completion of the testing process, the developed software is deployed as a beta version to be used by key users. The client is the chief user. Based on the user experience, it is determined if the software needs further development. Then, the user documentation is prepared defining the functionality of each of the components integrated to the framework. The documentation helps with the seamless installation, use and maintenance of the software. The project cycle comes to an end, and the end-product is delivered.

2.4 MONITORING OF SOFTWARE PROJECTS

Monitoring means – collecting, recording, and reporting information concerning project performance that project manager and others wish to know.

Controlling means – uses data from monitoring activity to bring actual performance to planned performance.

We monitor the project simply because we know that things don't always go according to plan (no matter how much we prepare) and to detect and react appropriately to deviations and changes to plans. We monitor the following:

Men (human resources), Machines, Materials, Money, Space, Time, Tasks and Quality/Technical Performance.

When do we monitor? The answer is "At end of the project or continuously or logically or While there is still time to react or As soon as possible or At task completion or At pre-planned decision points (milestones)".

How do we monitor? The answer to this question is

- Through meetings with clients, parties involved in project (Contractor, supplier, etc.).
- For schedule – Update PERT Charts and Gantt Charts.
- Using Earned Value Analysis.
- Calculate Critical Ratios.
- Milestones.
- Reports.
- Tests and inspections.
- Delivery or staggered delivery.
- Update PMIS (Project Management Info System).

The following issues related to monitoring are discussed in the meetings:

- What problems do you have and what is being done to correct them?
- What problems do you anticipate in the future
- Do you need any resources you do not yet have?
- Do you need information you do not have yet?
- Do you know anything that will give you scheduling difficulties?
- Any possibility of your task finishing early/late?
- Will your task be completed under/over/on budget?

2.5 CONTROLLING OF SOFTWARE PROJECTS

Projects are very dynamic – new requirements are discovered leading to changes to project scope, deliverables fail to meet quality standards, funding for the project gets cut, staff turnover, and other changes will necessitate a way of controlling all of these dynamics. Change control is one of the biggest challenges confronting a project since change often increases the risk, costs, and duration of the project. A centralized control process will be needed to manage changes to scope, schedule and budgets. Depending upon how significant the change is, there will be different levels of authority for approving the change.

The project control cycle is shown in Figure 2.6.

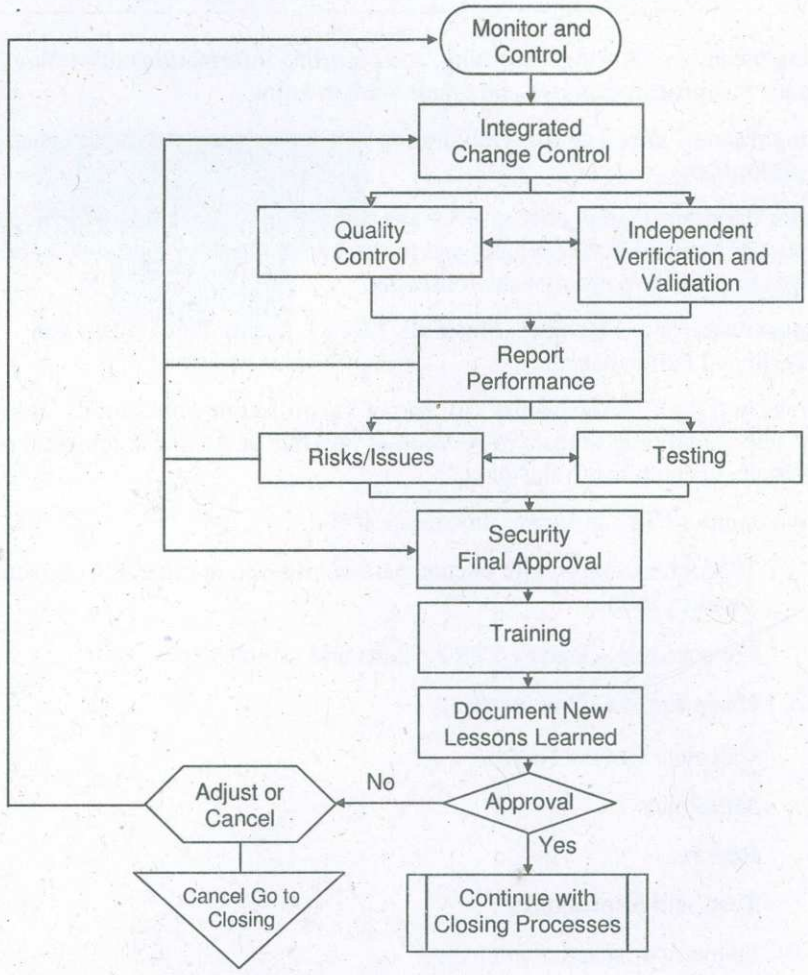


Figure 2.6 : Project Control Cycle

In addition to internal project changes (scope, schedule and budget), projects usually are confronted with external change. External changes are much more difficult to control since the Project Team has little control over the process. Regardless, you will need to address both internal and external changes:

(1) Recognize different changes that require control.

(2) Establish communication channels, and other controls to capture and review the change.

(3) Determine the impact of the change and communicate the change to those affected.

(4) Where appropriate, have approvals and documentation procedures for managing the change.

(5) Process and integrate the change into the project, such as changes to the performance measurement baseline.

(6) Verify that the change was in fact processed.

Projects often attempt to centralize the change control process by using a Change Control Log to document the date of the change, who originated the change, status (approved, open, canceled, etc.), amount of the change if applicable, highest WBS levels impacted, type of change and other important elements. In the case of technology related projects, this may take the form of Configuration Management. Configuration management is a process for managing all of the components and work products that go into building a system. This includes the life cycle of managing the system, from determining the requirements to maintaining the system. Additionally, most systems tend to "evolve" over time and you need a formal model to manage these changes, such as:

(1) Submit – Changes are submitted and processed through a standard form or screen.

(2) Evaluate – Analyze the change and work back upstream in the life cycle to determine how requirements must change. Decide how the change should be handled.

(3) Approve – Approving authority for decisions related to changes often follow threshold impacts and the ultimate authority for a change usually resides with the customer.

(4) Implement – Develop a plan to implement the change, test the change, and report the successful implementation of the change.

(5) Test – Run tests to make sure that the change worked as expected and no new changes pop-up as a result of this change.

(6) Update – Once you know the change is correct, you need to update your product related elements. This might include technical and user training documents. It might also involve software applications. Version control numbers are used for hard documents and release control numbers are used for software.

Projects will also impose enter and exit criteria during the project life cycle. These tollgates keep things on track and put discipline behind overall project management.

2.6 SOFTWARE PROJECTS

A failure is defined as any software project with severe cost or schedule overruns, quality problems, or that suffers outright cancellation. It has been suggested that there is more than one reason for a software development project to fail. However, most of the literature that discusses project failure tends to be rather general, supplying us with lists of risk and failure factors, and focusing on the negative business effects of the failure. Very little research attempted an in-depth investigation of a number of failed projects to identify the factors behind the failure exactly. While there does not appear to be any

overarching set of failure factors we discovered that all of the projects suffered from poor project management. Most projects additionally suffered from organizational factors outside the project manager's control.

The following are the list of the major software project failure factors:

- Organizational structure.
- Unrealistic or unarticulated goals.
- Software that fails to meet the real business needs.
- Badly defined system requirements, user requirements and requirements specification.
- The project management process, poor project management.
- Software development methodologies, sloppy development practices.
- Scheduling and project budget.
- Inaccurate estimates of needed resources.
- Poor reporting of the project status.
- Inability to handle project complexity.
- Unmanaged risks.
- Poor communication among customers, developers and users.
- Use of immature technology.
- Stakeholder politics.
- Commercial pressures.
- Customer satisfaction.
- Product quality.
- Leadership, and upper management support.
- Personality conflicts.
- Business processes and resources.
- Poor, or no tracking tools.

☞ **Check Your Progress 1**

(1) Good software planning can eliminate all interactions. True/False

(2) ..... are used to measure the status of software projects.

(3) A fault occurs when a program misbehaves. True/False.

2.7 SOFTWARE METRICS

Software process and project metrics are quantitative measures that enable software engineers to gain insight into the efficiency of the software process and the projects conducted using the process framework. In software project management, we are primarily concerned with productivity and quality metrics. There are four reasons for measuring software processes, products, and resources (to characterize, to evaluate, to predict, and to improve).

Measures and Metrics

- Measure – provides a quantitative indication of the size of some product or process attribute.
- Measurement – is the act of obtaining a measure.
- Metric – is a quantitative measure of the degree to which a system, component, or process possesses a given attribute.

Process Indicators

- Metrics should be collected so that process and product indicators can be ascertained
- Process indicators enable software project managers to: assess project status, track potential risks, detect problem area early, adjust workflow or tasks, and evaluate team ability to control product quality.

Process Metrics

- Private process metrics are only known to the individual or team concerned.
- Public process metrics enable organizations to make strategic changes to improve the software process.
- Metrics should not be used to evaluate the performance of individuals.
- Statistical software process improvement helps an organization to discover where they are strong and where they are weak.

Statistical Process Control

(1) Errors are categorized by their origin.

(2) Record cost to correct each error and defect.

(3) Count number of errors and defects in each category.

(4) Overall cost of errors and defects computed for each category.

(5) Identify category with greatest cost to organization.

(6) Develop plans to eliminate the most costly class of errors and defects or at least reduce their frequency.

Project Metrics

- A software team can use software project metrics to adapt project workflow and technical activities.
- Project metrics are used to avoid development schedule delays, to mitigate potential risks, and to assess product quality on an on-going basis.
- Every project should measure its inputs (resources), outputs (deliverables), and results (effectiveness of deliverables).

Software Measurement

- Direct measures of a software engineering process include cost and effort.
- Direct measures of the product include lines of code (LOC), execution speed, memory size, defects reported over some time period.
- Indirect product measures examine the quality of the software product itself (e.g., functionality, complexity, efficiency, reliability, and maintainability).

Size-Oriented Metrics

- Derived by normalizing (dividing) any direct measure (e.g., defects or human effort) associated with the product or project by LOC.
- Size-oriented metrics are widely used but their validity and applicability is a matter of some debate.

Function-Oriented Metrics

- Function points are computed from direct measures of the information domain of a business software application and assessment of its complexity.
- Once computed, function points are used like LOC to normalize measures for software productivity, quality, and other attributes.
- The relationship of LOC and function points depend on the language used to implement the software.

Object-Oriented Metrics

- Number of scenario scripts (NSS).
- Number of key classes (NKC).
- Number of support classes (e.g., UI classes, database access classes, computations classes, etc.).
- Average number of support classes per key class.
- Number of subsystems (NSUB).

Web Engineering Project Metrics

- Number of static Web pages (Nsp).
- Number of internal page links.
- Number of persistent data objects.
- Number of external systems interfaced.
- Number of static content objects.
- Number of dynamic content objects.
- Number of executable functions.

Software Quality Metrics

- Factors assessing software quality come from three distinct points of view (Product operation, product revision, product modification).
- Software quality factors requiring measures include
  - correctness (defects per KLOC).
  - maintainability (mean time to change).
  - integrity (threat and security).
  - usability (easy to learn, easy to use, productivity increase, user attitude).
- Defect removal efficiency (DRE) is a measure of the filtering ability of the quality assurance and control activities as they are applied throughout the process framework

  $$DRE = E / (E + D)$$

  where, E = number of errors found before delivery of work product.

  D = number of defects found after work product delivery.

Integrating Metrics with Software Process

- Many software developers do not collect measures.
- Without measurement, it is impossible to determine whether a process is improving or not.
- Baseline metrics data should be collected from a large, representative sampling of past software projects.
- Getting this historic project data is very difficult, if the previous developers did not collect data in an on-going manner.

Metrics for Small Organizations

- Best advice is to choose simple metrics that provide value to the organization and don't require a lot of effort to collect.
- Even small groups can expect a significant return on the investment required to collect metrics, if this activity leads to process improvement.

(1) Identify business goal.

(2) Identify what you want to know.

(3) Identify subgoals.

(4) Identify subgoal entities and attributes.

(5) Formalize measurement goals.

(6) Identify quantifiable questions and indicators related to subgoals.

(7) Identify data elements needed to be collected to construct the indicators.

(8) Define measures to be used and create operational definitions for them.

(9) Identify actions needed to implement the measures.

(10) Prepare a plan to implement the measures.

2.8 APPLICATION OF PERT CHART

Program Evaluation and Review Technique (PERT) is a scheduling method originally designed to plan a manufacturing project by employing a network of interrelated activities, coordinating optimum cost and time criteria. PERT emphasizes the relationship between the time each activity takes, the costs associated with each phase, and the resulting time and cost for the anticipated completion of the entire project.

PERT is an integrated project management system. These systems were designed to manage the complexities of major manufacturing projects, the extensive data necessary for such industrial efforts, and the time lines created by defense industry projects. Most of these management systems are developed following World War II, and each has its advantages and disadvantages.

PERT was first developed in 1958 by the U.S. Navy Special Projects Office on the Polaris missile system. Existing integrated planning on such a large scale was deemed inadequate, so the Navy pulled in the Lockheed Aircraft Corporation and the management consulting firm of Booz, Allen, and Hamilton. Traditional techniques such as line of balance, Gantt charts, and other systems were eliminated, and PERT evolved as a means to deal with the varied time periods it takes to finish the critical activities of an overall project.

The line of balance (LOB) management control technique collected, measured, and analyzed data to show the progress, status, and timing of production projects. It was introduced at Goodyear Tire and Rubber Company in 1941 and fully utilized during World War II in the defense industry. Even older is the Gantt chart, developed during World War I by Harvey Gantt, a pioneer in the field of scientific management. It is a visual management system, on which future time is plotted horizontally and work to be completed is indicated in a vertical line. The critical path method (CPM) evolved parallel to PERT. CPM is a mathematically ordered network of planning and scheduling project management; it was first used in 1957 by E.I. du Pont de Nemours & Co. PERT borrows some CPM applications. The growth of PERT paralleled the rapid expansion in the defense industry and meteoric developments in the space race. After 1960, all defense contractors adopted PERT to manage the massive one-time projects associated with the industry. Smaller businesses, awarded defense related government contracts, found it necessary to use PERT. At the same time, Du Pont developed CPM, which was particularly applied in the construction industry. In the last 30 years, PERT has spread, as has CPM, as a major technique of integrated project management.

PERT centers on the concept of time and allows flexible scheduling due to variations in the amount of time it takes to complete one specific part of the project. A typical PERT network consists of activities and events. An event is the completion of one program

component at a particular time. An activity is defined as the time and resources required to move from one event to another. Therefore, when events and activities are clearly defined, progress of a program is easily monitored, and the path of the project proceeds towards termination. PERT mandates that each preceding event be completed before succeeding events, and thus the final project, can be considered complete.

One key element to PERT's application is that three estimates are required because of the element of uncertainty and to provide time frames for the PERT network. These three estimates are classed as optimistic, most likely, and pessimistic, and are made for each activity of the overall project. Generally, the optimistic time estimate is the minimum time the activity will take – considering that all goes right at the first time and luck holds for the project. The reverse is the pessimistic estimate, or maximum time estimate for completing the activity. This estimate takes into account Murphy's law – whatever can go wrong will – and all possible negative factors are considered when computing the estimate. The third is the most likely estimate, or the normal or realistic time an activity requires. Two other elements comprise the PERT network: the path, or critical path, and slack time. The critical path is a combination of events and activities that will necessitate the greatest expected completion time. Slack time is defined as the difference between the total expected activity time for the project and the actual time for the entire project. Slack time is the spare time experienced in the PERT network.

2.8.1 Symbols of PERT Diagram

Figure 2.7 shows symbols of a PERT chart.

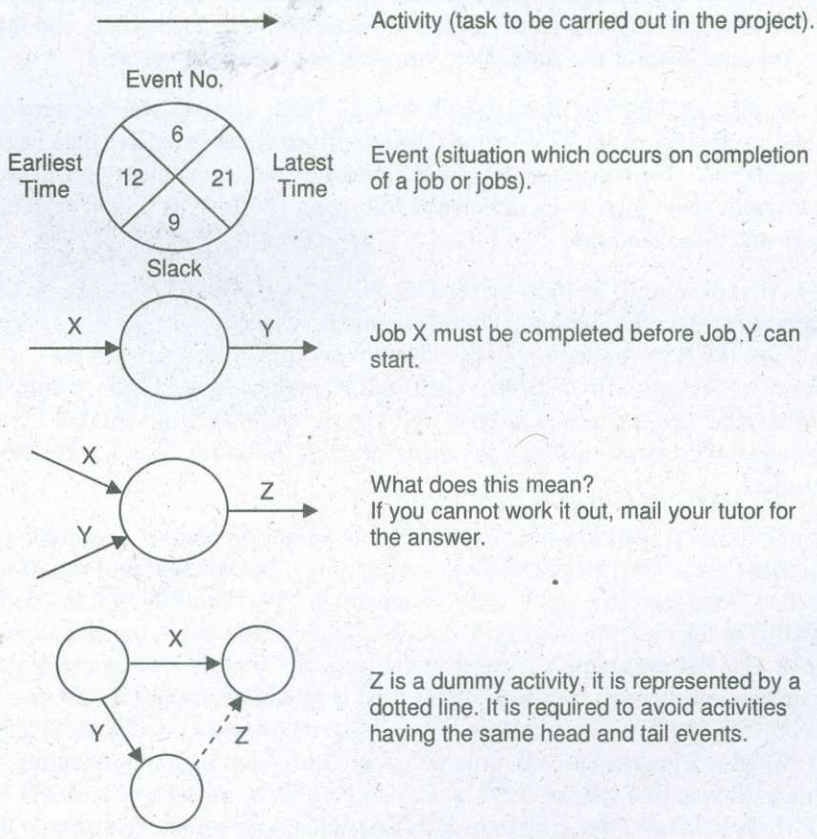


Figure 2.7 : Symbols of a PERT Chart

2.8.2 How to Produce the PERT Diagram

(1) There is a single start and end event.

(2) Time flows from left to right (so does the numbering sequence).

(3) Events are given a unique number (activities then have a unique label i.e. head and tail event numbers).

(4) The network can then be drawn taking into account the dependencies identified.

(5) Working from the start event forward, calculate the earliest times, setting the earliest time of the first event to zero. Add the job duration time to the earliest event time to arrive at the earliest time for the successor event.

(6) Working from the finish event backwards, calculate the latest times. Set the latest time to the earliest time for the finish event. Subtract job duration from the latest time to obtain predecessor latest event times.

(7) Event slack is calculated by subtracting the earliest event time from the latest event time.

(8) Critical path(s) are obtained by joining the events with zero event slack.

2.8.3 PERT Example

Consider the following example of PERT.

The following are the list of activities for the network:

Task	Dependent on	Duration
A	–	3
B	–	6
C	–	3
D	A	5
E	C	2
F	B, D, E	6
G	A	9

The resultant PERT chart is shown in Figure 2.8.

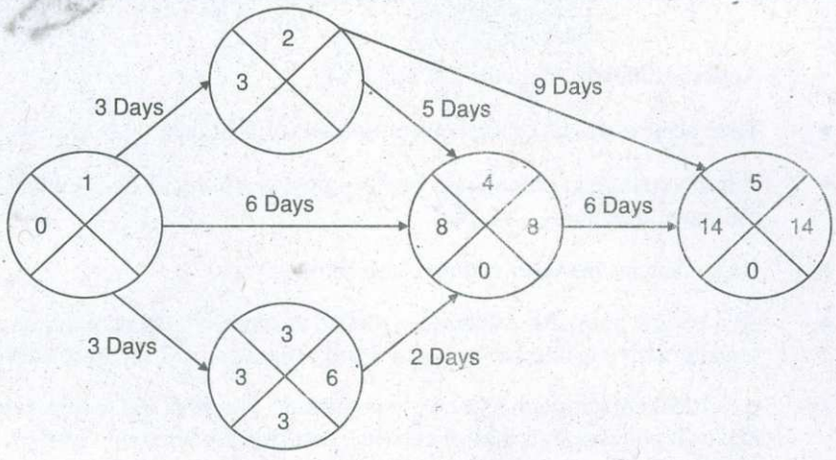


Figure 2.8: PERT Chart

2.9 GANTT CHARTS

Henry Laurence Gantt (1861-1919) was a mechanical engineer, management consultant and industry advisor. Henry Laurence Gantt developed Gantt charts in the second decade of the 20th century. Gantt charts were used as a visual tool to show scheduled and actual progress of projects. Accepted as a commonplace project management tool today, it was

an innovation of world-wide importance in the 1920s. Gantt charts were used on large construction projects like the Hoover Dam started in 1931 and the interstate highway network started in 1956.

2.9.1 GANTT Chart Basics

GANTT chart is a project planning tool that can be used to represent the timing of tasks required to complete a project. Gantt charts are simple to understand and easy to construct. Hence, they are used by most project managers for all but the most complex projects.

In a GANTT chart, each task takes up one row. Dates run along the top in increments of days, weeks or months, depending on the total length of the project. The expected time for each task is represented by a horizontal bar whose left end marks the expected beginning of the task and whose right end marks the expected completion date. Tasks may run sequentially, in parallel or overlapping.

As the project progresses, the chart is updated by filling in the bars to a length proportional to the fraction of work that has been accomplished on the task. This way, one can get a quick reading of project progress by drawing a vertical line through the chart at the current date. Completed tasks lie to the left of the line and are completely filled in. Current tasks cross the line and are behind schedule if their filled-in section is to the left of the line and ahead of schedule if the filled-in section stops to the right of the line. Future tasks lie completely to the right of the line.

In constructing a Gantt chart, keep the tasks to a manageable number (no more than 15 or 20) so that the chart fits on a single page. More complex projects may require subordinate charts which detail the timing of all the subtasks which make up one of the main tasks. For team projects, it often helps to have an additional column containing numbers or initials which identify the team member who is responsible for the task.

Often, the project has important events which you would like to appear on the project timeline, but which are not tasks. For example, you may wish to highlight when a prototype is complete or the date of a design review. You enter these on a Gantt chart as "milestone" events and mark them with a special symbol, often an upside-down triangle.

2.9.2 Features of GANTT Chart

A Gantt chart is a horizontal bar or line chart which will commonly include the following features:

- Activities identified on the left hand side.
- Time scale is drawn on the top (or bottom) of the chart.
- A horizontal open oblong or a line is drawn against each activity indicating estimated duration.
- Dependencies between activities are shown.
- At a review point the oblongs are shaded to represent the actual time spent (an alternative is to represent actual and estimated by 2 separate lines).
- A vertical cursor (such as a transparent ruler) placed at the review point makes it possible to establish activities which are behind or ahead of schedule.

2.9.3 Example of GANTT Chart

Develop the network activity chart and identify the critical path for a project based on the following information. Draw the activity network as a Gantt chart. What is the expected duration of the project?

Activity	Expected Duration	Predecessors
A	5 days	–
B	10 days	A
C	8 days	A
D	1 day	A
E	5 days	B, C
F	10 days	D, E
G	14 days	F
H	3 days	G
I	12 days	F
J	6 days	H, I

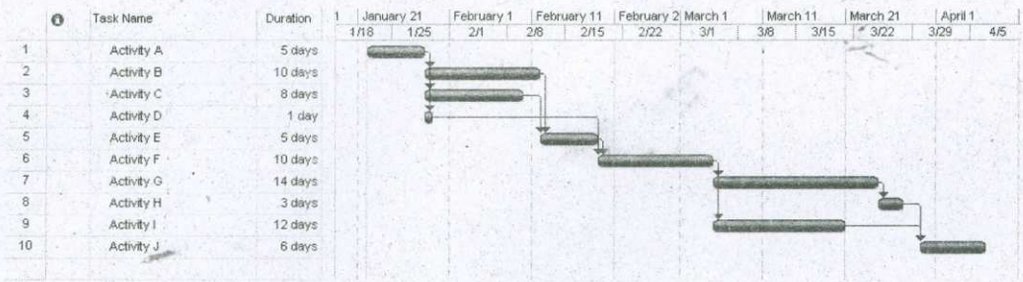


Figure 2.9: GANTT Chart for Activity Network

Expected duration of the project can be found by adding the lengths of the linked paths. In this case, it is 53 days. Figure 2.9 shows GANTT Chart for activity network.

2.10 SUMMARY

Software project management is perhaps the most important factor in the outcome of a project. Without proper project management, a project will fail. Many organizations have evolved effective project management processes. At the top level, the project management process consists of three phases: planning, execution, and closure. When creating a project schedule, managers will find both Program Evaluation and Review Technique (PERT) and Gantt charts to be essential tools for successfully completing the project at hand. Both types of charts provide tools for managers to analyze projects through visualization, helping divide tasks into manageable parts.

2.11 ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress 1

(1) False

(2) Check Points

(3) True

2.12 FURTHER READINGS

Reference Books

(1) *Software Project Management* (5th Edition) 2011 Hughes, Bob and Cotterell, Mike, McGraw Hill Higher Education.

(2) *IT Project Management: On Track from Start to Finish*, Third Edition Joseph Phillips, McGraw Hill.

Reference Websites

http://www.epmbook.com

http://en.wikipedia.org/wiki/Software_project_management

UNIT 3 SOFTWARE ENGINEERING FUNDAMENTALS

Structure

3.0 INTRODUCTION

Software Configuration Management (SCM) is an umbrella activity that is applied throughout the software process. Since change can occur at any time, SCM activities are developed to (1) identify change (2) control change, (3) ensure that change is being properly implemented and (4) report change to others who may have an interest. It is important to make a clear distinction between software maintenance and software configuration management. Maintenance is a set of software engineering activities that occur after software has been delivered to the customer and put into operation. Software configuration management is a set of tracking and control activities that begin when a software project begins and terminate only when the software is taken out of operation. A primary goal of software engineering is to improve the ease with which changes can be accommodated and reduce the amount of effort expended when changes must be made. Quality assurance consists of the auditing and reporting functions of management. The goal of quality assurance is to provide management with the data necessary to be informed about product quality, thereby gaining insight and confidence that product quality is meeting its goals.

3.1 OBJECTIVES

After going through this unit, you should be able to

- understand the importance, need and activities of Software Configuration Management,
- understand Software Maintenance and its types,
- study the various models of Software Maintenance and estimate Maintenance cost,
- understand the concept of Software Quality,
- study the Software Quality Models, and
- understand the SQA activities.

3.2 SOFTWARE CONFIGURATION MANAGEMENT

Changes are inevitable when software is built. A primary goal of software engineering is to improve the ease with which changes can be made to software. Configuration management (CM) is the discipline of controlling the evolution of complex systems; software configuration management (SCM) is its specialization for computer programs and associated documents. Software configuration management (SCM) is the discipline of controlling the evolution of complex software systems.

Every software engineer has to be concerned with how changes made to work products are tracked and propagated throughout a project. To ensure that quality is maintained the change process must be audited. A Software Configuration Management (SCM) Plan defines the strategy to be used for change management. A slightly more formal definition of software configuration management can be: SCM is a software-engineering discipline comprising the tools and techniques (processes or methodology) that a company uses to manage change to its software assets.

Differences between SCM and CM

SCM and CM differ from each other in the following two ways:

(1) Software is easier to change than Hardware, and it therefore changes faster. Even relatively small software systems, developed by a single team, can experience a significant rate of change, and in large systems, such as telecommunications systems, the update activities can totally overwhelm manual configuration management procedures.

(2) SCM is potentially more automatable since all components of a software system are easily stored on-line. CM for physical systems is hampered by having to handle objects that are not within reach of programmable controls. As CAD/CAM and robotics bring manufacturing processes more and more under computer control, physical configuration management will undoubtedly adopt some of the approaches used for software. VLSI has already achieved this by managing Circuit design and circuit processing in a manner similar to software design and compilation.

3.2.1 Basic Concepts and Definitions of SCM

The following are some basic concepts and terminologies used in software configuration management:

Revision Management

This is the core function of SCM and the foundation upon which other functions have been built. Revision management is the storage of multiple .images of the development files in an application, which can be shared by multiple developers.

With revision management, one can get an image, a snapshot in time of the development process and can re-create any file to the way it was at any point in time.

Version Management

While revision management shows development files in progress, a version is a particular instance of an entire development process to view checkpoints and approvals before content is posted. A good SCM solution must manage the progress of the project. A version is usually thought of as all the functions, features and complete builds you can use right now. Files managed, version-labeled and promoted by the revision management system are used to build the version.

Workflow and Process Management

As software or web content is developed, it is said to go through a lifecycle. Whether formal or informal, there are usually milestones in this lifecycle, such as development-test-alpha-beta-release for applications, test and production for packaged applications or, for websites, staging servers, reuse of code and content through the lifecycle. This is many levels above the simple source code protection built into some development platforms.

Build Management

When the code files, compilers, development tools and other components needed to create an application are ready, developers make an application build. Build management imposes automated, repeatable procedures that speed the build process, improve build accuracy and make it easier to create and maintain multiple versions of an application. An SCM toolset can become a vital resource for building an application across multiple platforms. A strong cross-platform build management capability also eliminates the need to redefine the build process (the script) for each platform that is being targeted.

Change Requests

The days of creating software, shrink-wrapping it, selling it and walking away are long gone. Today the software undergoes a constant process of revision. Development teams can be overwhelmed with the many software change requests (SCR) they receive, everything from users' wish lists to new business requirements that must be accommodated. SCM enables teams to streamline the SCR process by automatically tracking updates, change requests, problems and issues; assigning ownership and hand-offs within the development team; and communicating the process through to resolution.

Code and Project Auditing

Cross-platform development introduces special problems because the data to be audited may reside on multiple platforms. The best SCM systems provide a single point of access for auditing all development information. They also work across all kinds of development platforms, operating systems, network protocols and hardware platforms, and provide tailored clients within the developer's preferred development interface.

3.2.2 Software Configuration Items (SCI)

The software items to be configured through SCM include:

- Computer programs (both source and executable)
- Documentation (both technical and user)
- Data (contained within the program or external to it).

3.2.3 Necessity of Software Configuration Management

There are several reasons for putting a SCI under configuration management. But possibly the most important reason is to control the access to the different deliverables.

Unless strict discipline is enforced regarding updation and storage of different SCIs, several problems appear. The following are some of the important problems that appear if SCM is not used.

(a) ***Inconsistency Problem:*** The problem of inconsistency arises when the SCIs are replicated. Consider a situation where every software engineer has a personal copy of the SCI (e.g. source code). As each engineer makes changes to his local copy, he is expected to intimate them to other engineers, so that the changes in interfaces are uniformly changed across all modules. However, most of the times the changes are not communicated to the team thus making the different copies of the SCI inconsistent, leading to the failure of the integrated product.

(b) ***Concurrent Access:*** Simultaneous access and modifications of the different parts of an SCI may lead to unintentional overwriting of the changes made by the other teammates. Thus the final SCI may be deemed unusable for all.

(c) ***Stable Development Environment:*** When a project is underway, the team members need a stable environment to make progress. For example if one engineer is trying to integrate module A, with the modules B and C, he cannot proceed if developer of module C keeps changing C; this can be especially frustrating if a change to module C forces him to recompile A. With an effective SCM in place, the project leader freezes the SCIs to form a base line. Anyone needing an SCI 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.

(d) ***System Accounting and Maintaining Status Information*:** System accounting keeps track of who made a particular change and when the change was made.

(e) ***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.

3.2.4 SCM 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 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: Configuration identification involves deciding which parts of the system should be kept track of. The project manager classifies the SCIs associated with a software development effort into two main categories:

- Controlled: Controlled objects are those which are already put under configuration control. One must follow some formal procedures to change them. Requirements specification document, Design documents, Tools used to build the system such as compilers, linkers, lexical analyzers, parsers, etc., source code for each module, test cases, problem reports are objects in this category
- Uncontrolled: Uncontrolled objects are not and will not be subjected to configuration control.

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 due to configuration management increase to unreasonably higher levels. On the other hand, controlling too little might lead to confusion when something changes.

Configuration control: Configuration control ensures that changes to a system happen smoothly. It 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 has following functions:

- 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
- allow only one person to reserve a module at a time. Once an object is reserved, it does not allow any one else to reserve this module until the reserved module is restored thereby solving the problems associated with concurrent access.

Baselines: A work product becomes a baseline only after it is reviewed and approved. A baseline is a milestone in software development that is marked by the delivery of one or more configuration items. Once a baseline is established, each change request must be evaluated and verified by a formal procedure before it is processed. Baselined work products are placed in a project database or repository.

Steps to modify any object under configuration control:

- The engineer needing to change a module first obtains a private copy of the module through a reserve operation.
- He carries out all necessary changes on this private copy.
- Restoring the changed module to the system configuration requires the permission of a change control board (CCB).
- The CCB (comprising the project manager and some senior members of the development team) reviews the changes made to the controlled object and certifies the following about the change:
  - Change is well-motivated.
  - Developer has considered and documented the effects of the change.
  - Changes interact well with the changes made by other developers.
  - The change has been validated and is consistent with the requirement.

The project manager updates the old base line through a restore operation.

3.2.5 Configuration Management Tools

Source Code Control System (SCCS) and Revision Control System (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. They provide an efficient way of storing versions that minimize the amount of occupied disk space. Suppose, a module SWM is present in three versions SWM1.1, SWM1.2, and SWM1.3. Then, SCCS and RCS store the original module SWM1.1 together with changes needed to transform SWM1.1 to SWM1.2 and SWM1.2 to SWM1.3. 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. That is, reserve and restore operations.

3.3 SOFTWARE MAINTENANCE

Software maintenance is an integral part of Software Development Life Cycle (SDLC). The term 'Maintenance" when associated with software, assumes a meaning profoundly different from the meaning it assumes in any other engineering discipline. In fact, in most engineering disciplines, the term "Maintenance" refers to the process of keeping something in working order. That is, in repair. The key concept is the deterioration of an equipment or machinery due to use and passage of time. The aim of maintenance is to therefore keep the object's functionality in line with that defined and registered at the time of release. This view of maintenance does not apply to software as software does not deteriorate with the use and the passage of time. Nevertheless, the need for modifying a piece of software after delivery has been with us since the very beginning of electronic computing. Hence Software Maintenance as defined by The Institute of Electrical and Electronics Engineers (IEEE) stands as:

Software maintenance is the process of modifying a software system or component after delivery to correct faults, improve performances or other attributes, or adapt to a changed environment.

3.3.1 Need for Software Maintenance

Maintenance is needed to ensure that the system continues to satisfy user requirements. Maintenance is applicable to systems developed using any software development model (e.g., spiral). The need for Software maintenance arises due to corrective and non-corrective software actions. Maintenance must be performed due to some of the following reasons:

- ***Hardware Obsolescence*:** The rate of hardware obsolescence in comparison to the immortal software creates a demand of the user community to see the existing software products run on newer platforms, in newer environments, and/or with enhanced features.
- ***Hardware Evolution:*** As changes are introduced in the hardware platform, a software product performing some low-level functions require maintenance.
- ***Environmental Changes:*** Changes in the support environment of a software product, require rework in the software to cope up with the newer interface.
- ***Correct Errors:*** As the software is modified, maintenance is required to make the software error free.
- ***Enhance Features:*** Enhancements requested in the software by the customer from time to time require an ongoing maintenance activity.
- ***Software Portability:*** Ensuring portability of software across platforms, both existing and upcomimg require maintenance.
- ***Correct Requirements and Design Flaws:*** The ever changing requirements of the customer during the development process and even after implementation need to be handled as maintenance effort.
- ***Interface with other systems:*** Convert programs so that different hardware, software, system features, and telecommunications facilities can be used.
- ***Migrate legacy systems.***

3.3.2 Types of Software Maintenance

The categories of maintenance defined by ISO/IEC are as follows:

- **Corrective maintenance:** Reactive modification of a software product performed after delivery to correct discovered problems.
- **Adaptive maintenance:** Modification of a software product performed after delivery to keep a software product usable in a changed or changing environment.

- **Perfective maintenance:** Modification of a software product after delivery to improve performance or maintainability.
- **Preventive maintenance:** Modification of a software product after delivery to detect and correct latent faults in the software product before they become effective faults.
- **Enhancive maintenance:** Modifications to the software after delivery to satisfy the customer expectations of extended functionalities and features to support other additional applications.

Adaptive, Perfective and Enhancive maintenance can be called as enhancements; Corrective and Preventive maintenance as corrections. Preventive maintenance, the newest category, is defined as maintenance performed for the purpose of preventing problems before they occur. Preventive maintenance is most often performed on software products where safety is critical. The distribution of approximate effort spent during the life time of a software project in maintenance has been depicted in Figure 3.1.

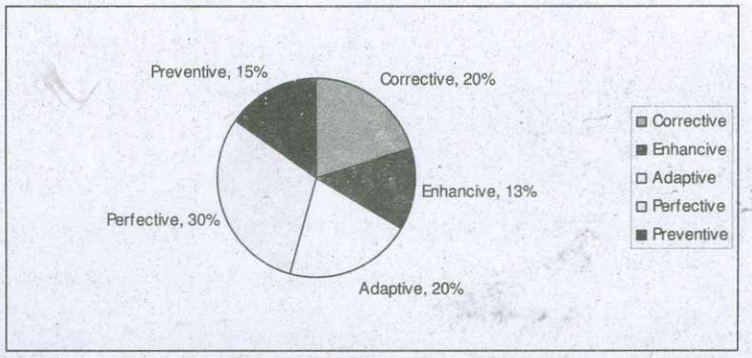


Figure 3.1: Distribution of Maintenance Effort

3.3.3 Software Maintenance Process

As software is due for maintenance owing to any of the reasons stated in Section 3.3.1, the software is subjected to the maintenance process which is divided into the following five phases:

Phase 0: Determine Maintenance Objective: This phase involves identifying the objective of maintenance i.e. correcting program errors, enhancing capabilities, software portability etc.

Phase 1: Program Understanding: The first phase consists of analyzing the program in order to understand. This phase involves analyzing the program complexity, documentation and the program descriptiveness.

Phase 2: Generating Particular Maintenance Proposal: The second phase consists of generating a particular maintenance proposal to accomplish the implementation of the maintenance objective.

Phase 3: Ripple Effect: The third phase consists of accounting for all of the ripple effects as a consequence of program modifications.

Phase 4: Modified Program Testing: The fourth phase consists of testing the modified program to ensure that the modified program has at least the same reliability level as before.

The process is depicted in Figure 3.2.

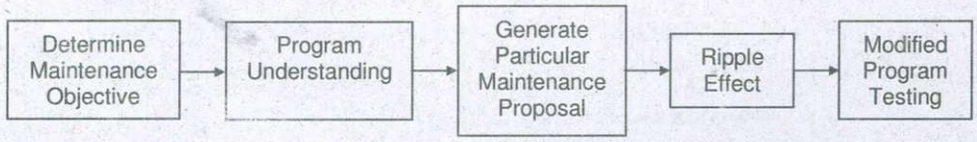


Figure 3.2: Process of Software Maintenance

3.3.4 Software Maintenance Models

The following are some of the models of Software Maintenance:

Quick-Fix Model

This is basically an adhoc approach to maintenance of software (refer to Figure 3.3). It is a fire fighting approach, waiting for the problem to occur and then trying to fix it as quickly as possible.

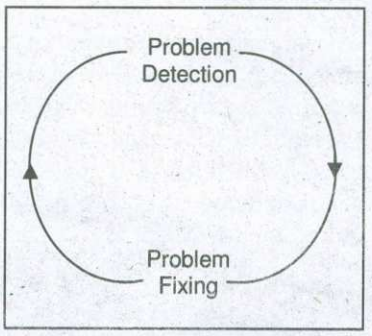


Figure 3.3: Quick-Fix Model

Iterative Enhancement Model

The model consists of three stages in each iteration (refer to Figure 3.4):

(1) Analysis of the Existing system: This includes study of the existing system.

(2) Characterization of proposed modifications.

(3) Redesign of the current version and implementation.

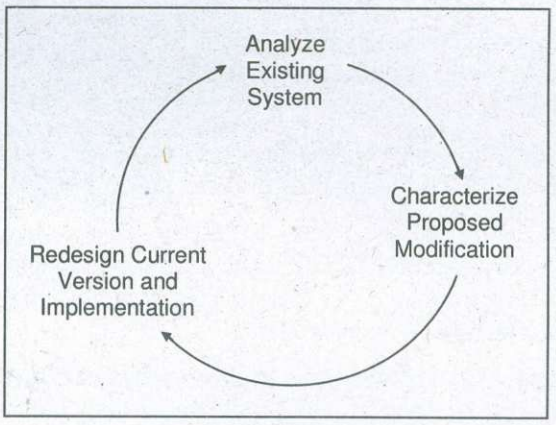


Figure 3.4: Iterative Enhancement Model

Reuse Oriented Model

The model emphasizes component reuse in the following way (refer to Figure 3.5):

(1) Identification of the parts of the old system that are candidates for reuse.

(2) Understanding these system parts.

(3) Modification of the old system parts appropriate to the new requirements.

(4) Integration of the modified parts into the new system.

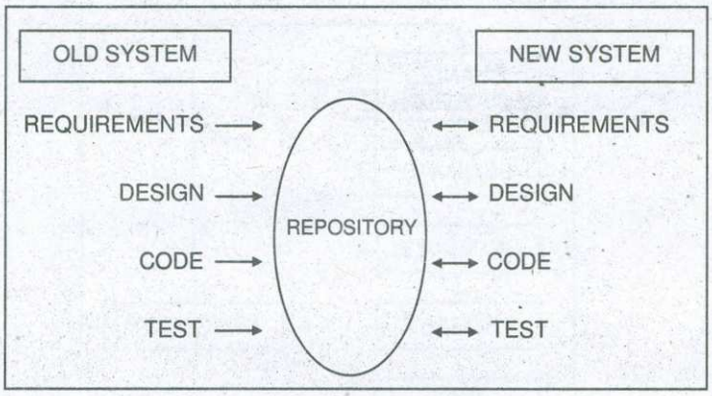


Figure 3.5: Reuse Oriented Model

Boehm's Model

The maintenance process model proposed by Boehm is based on the economic models and principles. He represents the maintenance process as a closed loop cycle as depicted in Figure 3.6.

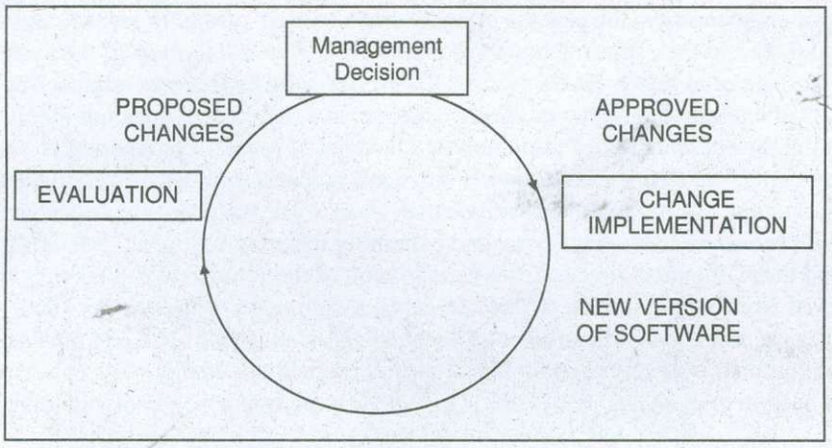


Figure 3.6: Boehm's Model

Taute Maintenance Model

The model depicted in Figure 3.7 consists of the following eight phases:

(1) **Change request phase:** Changes are accepted from the customer.

(2) **Estimate phase:** The effort and cost estimates are computed for the requested changes.

(3) **Schedule phase:** The schedule for the work to be done is prepared for incorporating the changes.

(4) **Programming phase:** The changes are implemented.

(5) **Test phase:** The modified software is tested.

(6) **Documentation phase:** The documentation for the tested software is prepared.

(7) **Release phase:** The modified software is released.

(8) **Operation phase:** The modified software is finally fully operational after correcting the post release bugs.

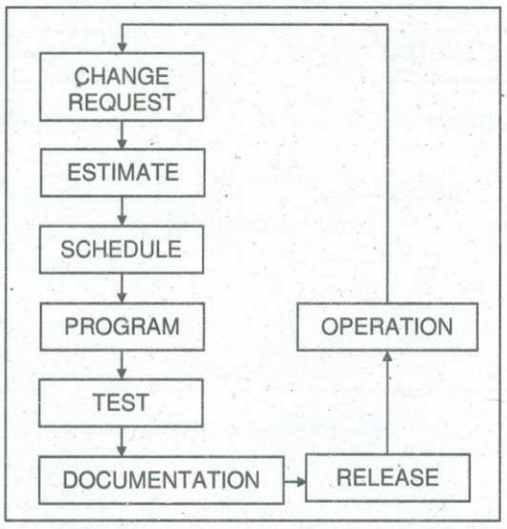


Figure 3.7: Taute Maintenance Model

3.3.5 Reverse Engineering

Reverse engineering is the process of analyzing a subject system to identify the system's components and their interrelationships and to create representations of the system in another form or at higher levels of abstraction. Accordingly, reverse engineering is a process of examination, not a process of change, and therefore it does not involve changing the software under examination. One type of reverse engineering is redocumentation. ***Redocumentation*** is the recreation of a semantically equivalent representation within the same relative abstraction level. Another type is design recovery. ***Design recovery*** entails identifying and extracting meaningful higher level abstractions beyond those obtained directly from examination of the source code. This may be achieved from a combination of code, existing design documentation, personal experience, and knowledge of the problem and application domains. ***Refactoring***, a program transformation that reorganizes a program without changing its behavior, is now being used in reverse engineering to improve the structure of object oriented programs.

Although software reverse engineering originated in software maintenance, it is applicable to many problem areas. Six key objectives of reverse engineering are:

- coping with complexity.
- generating alternate views.
- recovering lost information.
- detecting side effects.
- synthesizing higher abstractions.
- facilitating reuse.

Reverse engineering has been recommended as a key supporting technology to deal with systems that have the source code as the only reliable representation. Examples of problem areas where reverse engineering has been successfully applied include identifying reusable assets, finding objects in procedural programs, discovering architectures, deriving conceptual data models, detecting duplications, transforming binary programs into source code, renewing user interfaces, parallelizing sequential programs, and translating, downsizing, migrating, and wrapping legacy code. Reverse engineering principles have also been applied to business process reengineering to create a model of an existing enterprise.

Reverse engineering has been viewed as a two step process: information extraction and abstraction. *Information extraction* analyses the subject system artifacts – primarily the source code – to gather row data, whereas *information abstraction* creates user-oriented documents and views.

The IEEE Standard for Software Maintenance suggests that the process of reverse engineering evolves though the following six steps:

- dissection of source code into formal units.
- semantic description of formal units and creation of functional units.
- description of links for each unit (input/output schematics of units).
- creation of a map of all units and successions of consecutively connected units (linear circuits).
- declaration and semantic description of system applications.
- creation of an anatomy of the system.

The first three steps concern local analysis at a unit level while the other three steps are for global analysis at a system level.

3.3.6 Re-Engineering

Re-engineering is defined as the examination and alteration of the subject system to reconstitute it in a new form, and the subsequent implementation of the new form. In other words, re-engineering is the activity of better understanding and maintenance of a software system. Reengineering is often not undertaken to improve maintainability but is used to replace aging legacy systems. Formally defining "Software Re-engineering is any activity that: (1) improves one's understanding of software, or (2) prepares or improves the software itself, usually for increased maintainability, reusability, or evolvability."

Re-engineering entails some form of reverse engineering to create a more abstract view of a system, a regeneration of this abstract view followed by forward engineering activities to realize the system in the new form. This process is illustrated in Figure 3.8. The presence of a reverse engineering step distinguishes re-engineering from restructuring, the latter consisting of transforming an artifact from one form to another at the same relative level of abstraction.

Software re-engineering has proven important for several reasons. Seven main reasons that demonstrate the relevance of re-engineering are the following:

- Re-engineering can help reduce an organization's evolution risk.
- Re-engineering can help an organization recoup its investment in software.
- Re-engineering can make software easier to change.
- Re-engineering is a big business.
- Re-engineering capability extends CASE toolsets.
- Re-engineering is a catalyst for automatic software maintenance.
- Re-engineering is a catalyst for applying artificial intelligence techniques to solve software re-engineering problems.

Examples of scenarios in which re-engineering has proven useful include migrating a system from one platform to another, downsizing, translating, reducing maintenance costs, improving quality, and migrating and re-engineering data.

Software re-engineering is a complex process that re-engineering tools can only support, not completely automate. There is a good deal of human intervention with any software reengineering project. Re-engineering tools can provide help in moving a system to a new maintenance environment, for example one based on a repository, but they cannot define such an environment nor the optimal path along which to migrate the system to it. These are activities that only human beings can perform.

Another problem that re-engineering tools only marginally tackle is the creation of an adequate testbed to prove that the end product of reengineering is fully equivalent to the original system. This still involves much hand-checking, partially because very rarely an application is re-engineered without existing functions being changed and new functions being added.

Finally, re-engineering tools often fail to take the unique aspects of a system into account, such as the use of a JCL or a TP-Monitor, the access to a particular DBMS or the presence of embedded calls to modules in other languages.

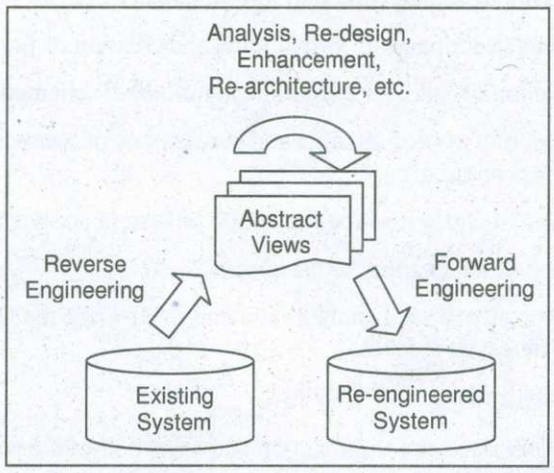


Figure 3.8: Reverse Engineering and Re-Engineering

3.3.7 Estimation of Approximate Maintenance Cost

It is well known that maintenance efforts require about 60% of the total life cycle cost for a typical software product. However, maintenance costs vary widely from one application domain to another. For embedded systems, the maintenance cost can be as much as 2 to 4 times the development cost.

Boehm proposed a formula for estimating maintenance costs as part of his COCOMO cost estimation model. Boehm's maintenance cost estimation is made in terms of a quantity called the Annual Change Traffic (ACT). Boehm defined ACT as the fraction of a software product's source instructions which undergo change during a typical year either through addition or deletion.

$$ACT = \frac{KLOC_{added} + KLOC_{deleted}}{KLOC_{total}}$$

where, $KLOC_{added}$ is the total kilo lines of source code added during maintenance. $KLOC_{deleted}$ the total KLOC deleted during maintenance. Thus, the code that is changed, should be counted in both the code added and the code deleted. The annual change traffic (ACT) is multiplied with the total development cost to arrive at the maintenance cost:

$$\text{Maintenance Cost} = ACT \times \text{Development Cost}$$

Most maintenance cost estimation models, however, yield only approximate results because they do not take into account of several factors such as experience level of the engineers, and familiarity of the engineers with the product, hardware requirements, software complexity, etc.

☞ **Check Your Progress 1**

(1) Differentiate between Software Re-engineering and Reverse Engineering.

(2) If the development cost of a software product is Rs. 20000000/-, compute the annual maintenance cost given that every year approximately 10% of the code needs modification.

3.4 SOFTWARE QUALITY ASSURANCE (SQA)

The term "Software Quality" refers to conformance to explicitly stated requirements and standards, as well as implicit characteristics that customers assume will be present in any professionally developed software. The SQA group must look at software from the customer's perspective, as well as assessing its technical merits. Software Quality

Assurance controls variation among products. Software engineers are concerned with controlling the variation in their processes, resource expenditures, and the quality attributes of the end products. The activities performed by the SQA group involve quality planning, oversight, record keeping, analysis and reporting. An elaborate definition of SQA can be given as the following:

A systematic, planned set of actions necessary to provide adequate confidence that the software development process or the maintenance process of the software system product conforms to established functional technical requirements as well as with the managerial requirements of keeping the schedule and operating within budgetary confines.

Software Quality Assurance (SQA) consists of a means of monitoring the software engineering processes and methods used to ensure quality. It does this by means of audits of the quality management system under which the software system is created. These audits are backed by one or more standards, usually ISO 9000.

It is distinct from software quality control. Quality Control (QC) is a set of activities (including reviewing requirements documents, and software testing) carried out with the main objective of withholding products from shipment if they do not qualify. Quality Assurance (QA) is meant to minimize the costs of quality by introducing a variety of activities throughout the development process and maintenance process in order to prevent the causes of errors, detect them, and correct them in the early stages of the development. As a result, quality assurance substantially reduces the rate of non-qualifying products. Software quality control is a control of products, software quality assurance is a control of processes.

3.4.1 Software Errors, Faults and Failures

In this section, we discuss about errors, faults and failures in Software.

***Software Error*:** Section of the code that is incorrect as a result of grammatical, logical or other mistakes made by a system analyst, a programmer, or another member of the software development team. Software errors can arise due to the following reasons:

- Faulty requirements
- Client-developer communication failures
- Deliberate deviations from software requirements
- Logical design errors
- Coding errors
- Non compliance with documentation and coding instructions
- Shortcoming of the testing process
- Procedure errors
- Documentation errors

Software Fault: Software faults are software errors that causes the incorrect functioning of the software.

***Software Failure*:** Software faults become software failures only when they are "activated".

3.4.2 Cost of Quality

Cost of quality includes all costs incurred in the pursuit of quality or in performing quality related activities. The basis of normalization is almost always dollars. Once we have normalized quality costs on a dollar basis, we have the necessary data to evaluate where the opportunities lie to improve our processes. Furthermore, we can evaluate the affect of changes in dollar-based terms. Quality costs may be divided into costs associated with prevention, appraisal, and failure.

Prevention costs *include:*

- Quality planning
- Formal technical reviews
- Test equipment
- Training

Appraisal costs include activities to gain insight into product condition "first time through" each process. Examples of appraisal costs include:

- In-process and inter process inspection
- Equipment calibration and maintenance
- Testing

Failure costs are costs that would disappear if no defects appeared before shipping a product to customers. Failure costs may be subdivided into internal failure costs and external failure costs.

- ***Internal failure costs*** are the costs incurred when we detect an error in our product prior to shipment. Internal failure costs include:
  - Rework
  - Repair
  - Failure mode analysis
- ***External failure costs*** are the costs associated with defects found after the product has been shipped to the customer. Examples of external failure costs are:
  - Complaint resolution
  - Product return and replacement
  - Help line support
  - Warranty work

As expected, the relative costs to find and repair a defect increase dramatically as we go from prevention to detection and from internal failure to external failure.

3.4.3 Software Quality Models

Software quality can be described by specific quality models. Two such models have been described in this section.

McCall Software Quality Model

The concept of software quality and the efforts to understand it in terms of measurable quantities i.e quality factors and quality criteria was attempted by McCall. A *Quality Factor* represents a behavioral characteristic of a system. Quality factors are external attributes of a software system. Customers, software developers, and quality assurance engineers are interested in different quality factors to different extents. For example, customers may want an efficient and reliable software with less concern for portability. The developers strive to meet customer needs by making their system efficient and reliable, at the same time making the product portable and reusable to reduce the cost of software development. The software quality assurance team is more interested in the testability of a system so that some other factors, such as correctness, reliability, and efficiency, can be easily verified through testing. McCall identified 11 quality factors. The 11 quality factors have been grouped into three broad categories, depicted in Figure 3.9, as follows:

(i) **Product Operation:** Factors which are related to the operation of a product are combined. These five factors are related to operational performance,

convenience, ease of usage and its correctness. These factors play a very significant role in building customer's satisfaction. The factors are:

- *Correctness*: Extent to which a program satisfies its specifications and fulfills the user's mission objectives.
- *Efficiency:* Amount of computing resources and code required by a program to perform a function.
- *Integrity:* Extent of access to software or data by unauthorized persons can be controlled.
- *Reliability:* Extent to which a program can be expected to perform its intended function with required precision.
- *Usability:* Effort required to learn, operate, prepare input and interpret output of a program.

(ii) **Product Revision:** These factors pertain to the testing & maintainability of software. They give us idea about ease of maintenance, flexibility and testing effort. Hence, they are combined under the umbrella of product revision.

- *Maintainability:* Effort required for locating and fixing a defect in an operational program.
- *Flexibility:* Effort required for modifying an operational program.
- *Testability:* Effort required for testing a program to ensure that it performs its intended functions.

(iii) **Product Transition:** We may have to transfer a product from one platform to another platform or from one technology to another technology. The factors related to such a transfer are combined and given below:

- *Portability:* Effort required to transfer a program from one hardware and/or software environment to another
- *Reusability:* Extent to which parts of a software system can be reused in other applications.
- *Interoperability:* Effort required to couple one system with another.

It may be noted that the above three categories relate more to post-development activities expectations and less to in-development activities. In other words, McCall's quality factors emphasize more on the quality levels of a product delivered by an organization and the quality levels of a delivered product relevant to product maintenance.

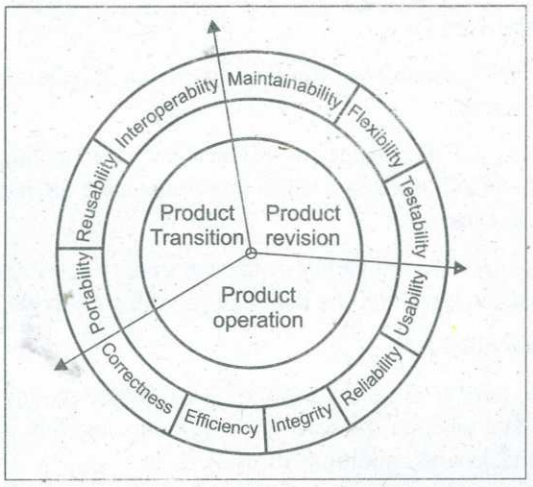


Figure 3.9: McCalls Software Quality Factors

ISO 9126 Software Quality Model

To define a general framework for software quality, an expert group, under the aegis of the ISO, standardized a software quality document, namely, ISO 9126, which defines six broad, independent categories of quality characteristics as follows:

- *Functionality*: A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.
- *Reliability*: A set of attributes that bear on the capability of software to maintain its performance level under stated conditions for a stated period of time.
- *Usability*: A set of attributes that bear on the effort needed for use and on the individual assessment of such use by a stated or implied set of users.
- *Efficiency*: A set of attributes that bear on the relationship between the software's performance and the amount of resource used under stated conditions.
- *Maintainability*: A set of attributes that bear on the effort needed to make specified modifications (which may include corrections, improvements, or adaptations of software to environmental changes and changes in the requirements and functional specifications).
- *Portability*: A set of attributes that bear on the ability of software to be transferred from one environment to another (this includes the organizational, hardware or, software environment).

The ISO 9126 standard further decomposes the quality characteristics into more concrete sub characteristics.

Functionality has been subdivided into:

- *Suitability*: The capability of the software to provide an adequate set of functions for specified tasks and user objectives.
- *Accuracy*: The capability of the software to provide the right or agreed-upon results or effects.
- *Interoperability*: The capability of the software to interact with one or more specified systems.
- *Security*: The capability of the software to prevent unintended access and resist deliberate attacks intended to gain unauthorized access to confidential information or to make unauthorized modifications to information or to the program so as to provide the attacker with some advantage or so as to deny service to legitimate users.

Reliability has been subdivided into:

- *Maturity*: The capability of the software to avoid failure as a result of faults in the software.
- *Fault Tolerance*: The capability of the software to maintain a specified level of performance in the case of software faults or of infringement of its specified interface.
- *Recoverability*: The capability of the software to re-establish its level of performance and recover the data directly affected in the case of a failure.

Usability has been subdivided into:

- *Understandability*: The capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use.

- *Learnability*: The capability of the software product to enable the user to learn its applications.
- *Operability*: The capability of the software product to enable the user to operate and control it.
- *Attractiveness*: The capability of the software product to be liked by the user.

Efficiency has been subdivided into:

- *Time Behavior*: The capability of the software to provide appropriate response and processing times and throughput rates when performing its function under stated conditions.
- *Resource Utilization*: The capability of the software to use appropriate resources in an appropriate time when the software performs its function under stated condition.

Maintainability has been subdivided into:

- *Analyzability*: The capability of the software product to be diagnosed for deficiencies or causes of failures in the software or for the parts to be modified to be identified.
- *Changeability*: The capability of the software product to enable a specified modification to be implemented.
- *Stability* : The capability of the software to minimize unexpected effects from modifications of the software.
- *Testability* : The capability of the software product to enable modified software to be validated.

Portability has been subdivided into:

- *Adaptability*: The capability of the software to be modified for different specified environments without applying actions or means other than those provided for this purpose for the software considered.
- *Installability* : The capability of the software to be installed in a specified environment.
- *Coexistence*: The capability of the software to coexist with other independent software in a common environment sharing common resources.
- *Replaceability*: The capability of the software to be used in place of other specified software in the environment of that software.

Differences between McCall's quality model and ISO 9126 model

- The ISO 9126 model emphasizes characteristics *visible* to the users, whereas the McCall model considers *internal* qualities as well. For example, reusability is an internal characteristic of a product. Product developers strive to produce reusable components, whereas its impact is not perceived by customers.
- In McCall's model, one quality criterion can impact several quality factors, whereas in the ISO 9126 model, one sub-characteristic impacts exactly one quality characteristic.
- A high-level quality factor, such as testability, in the McCall model is a low-level sub-characteristic of maintainability in the ISO 9126 model.

3.4.4 SQA Activities

Software quality assurance comprises of a variety of tasks associated with two different groups. That is, the software engineers who do technical work and the SQA group that has responsibility for quality assurance planning, over sight, record keeping, analysis and reporting. Software engineers address quality and perform quality assurance by applying

technical methods and measures, conducting formal technical reviews, and performing well-planned software testing. The SQA group assists the software engineering team in obtaining a high quality end product.

The Software Engineering Institute recommends a set of SQA activities that address quality assurance planning. These activities are performed by an independent SQA group. The SQA plan is developed during project planning and is reviewed by all interested parties. The plan identifies:

- Evaluations to be performed
- Audits and reviews to be performed
- Standards that are applicable to the project
- Procedures for error reporting and tracking
- Documents to be produced by the SQA group
- Amount of feedback provided to software project team.

3.4.5 Formal Technical Reviews

A *Formal Technical Review (FTR)* is a software quality assurance activity performed by software engineers. The objectives of the FTR are:

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

In addition, the FTR serves as a training ground, enabling junior engineers to observe different approaches to software analysis, design, and implementation. The FTR also serves to promote backup and continuity since number of people will become familiar with parts of the software that they may not have otherwise seen.

The FTR is actually a class of reviews that include walkthroughs, inspections, round-robin reviews, and other small group technical assessments of software. Each FTR is conducted as a meeting and is successful only if it is properly planned, controlled, and attended.

The Review Meeting

The FTR focuses on a specific and small part of the overall software. For example rather than attempting to review an entire design, walkthroughs are conducted for each module or small group of modules. With a narrower focus, FTR has a higher likelihood of uncovering errors. Regardless of the chosen FTR format, every review meeting should abide by the following constraints:

- The review should involve three to five people.
- Advance preparations must be made by each person.
- The duration of the review meeting should be normally less than two hours.

The focus of the FTR is on a work product-a component of the software. The individual who has developed the work product informs the project leader that the work product is complete and that a review is required. The project leader contacts a review leader, who evaluates the work product for readiness. Generates copies, and distributes them to two or three reviewers for advance preparation. Each reviewer is expected to spend between one and two hours reviewing the work

product, making notes and otherwise becoming familiar with the work. Concurrently, the review leader also reviews the work product and prepares for the review meeting which is typically scheduled for the next day.

The review meeting is attended by the review leader, all reviewers and the producer. One of the reviewers takes on the role of the recorder i.e. the individual who records (in writing) all important issues raised during the review. The FTR begins with an introduction of the agenda and a brief introduction by the producer. The producer then proceeds to "walk through" the work product, explaining the material, while reviewers raise issues based on their advance preparation. When valid problems or errors are discovered the recorder notes each. At the end of the review, all attendees of the FTR must decide whether to:

- accept the work product without further modification.
- reject the work product due to severe errors (once corrected, another review must be performed) or
- accept the work product provisionally (minor errors have been encountered and must be corrected, but no additional review will be required).

The decision is finally agreed upon and all FTR attendees complete a sign-off indicating their participation in the review and their concurrence with the review team's findings.

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 a review issues list is produced. In addition, a simple review summary report is completed. A review summary report answers three questions:

- What was reviewed?
- Who reviewed it?
- What were the findings and conclusions?

The review summary report is typically a 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 issues list serves two purposes: (1) to identify problem areas within the product and (2) to serve as an action item checklist that guides the producer as corrections are made. An issues list is normally attached to the summary report. It is important to establish a follow-up procedure to ensure that items on the issues list have been properly corrected. The responsibility for follow-up is assigned to the review leader.

Review Guidelines

Guidelines for the conduct of formal technical reviews must be established in advance, distributed to all reviewers, agreed upon and then followed. A review that is uncontrolled can often be worse than no review at all. Following are a set of guidelines for formal technical reviews:

(1) *Review the product, not the producer:* An FTR involves people and egos. Conducted properly, the FTR should leave all participants with a warm feeling of accomplishment. Errors should be pointed out gently; the tone of the meeting should be loose and constructive; and the intent should not be to embarrass or belittle. The review leader should conduct the review meeting to ensure that the proper tone and attitude are maintained and should immediately halt a review that has gotten out of control.

(2) *Set an agenda and maintain it:* An FTR must be kept on track and on schedule.

(3) *Limit debate and rebuttal*: When an issue is raised by a reviewer, there may not be universal agreement on its impact. Rather than spending time debating the question, the issue should be recorded for further discussion off-line.

(4) *Enunciate problem areas, but don't attempt to solve every problem noted*: A review is not a problem solving session. The solution of a problem can often be accomplished by the producer alone or with the help of only one other individual. Problem solving should be postponed until after the review meeting.

(5) *Take written notes*: It is sometimes a good idea for the recorder to make notes on as wall board, so that wording and prioritization can be assessed by other reviewers as information is recorded.

(6) *Limit the number of participants and insist upon advance preparation*: Keep the number of people involved to the necessary minimum. However, all review team members must prepare in advance.

(7) *Develop a checklist for each work product that is likely to be reviewed*: Checklists should be developed for analysis, design, coding, and even test documents.

(8) *Allocate resources and time schedule for FTR.*

(9) *Conduct meaningful training for all reviewers*: To be effective all review participants should receive some formal training.

(10) *Review your earlier reviews.*

☞ **Check Your Progress 2**

(1) Describe the concept of Software Quality. How is quality of software products different from that of hardware products?

(2) Describe the importance of Formal Technical Reviews in the Software Quality Assurance process.

3.5 SUMMARY

In this chapter the concepts of Software Configuration Management have been elaborated. SCM is a set of activities that help in incorporating changes in software. Maintenance refers to the set of activities that occur after the software has been delivered to the customer. SQA is a set of activities that provide the data necessary about product quality, thereby gaining insight and confidence that product is meeting set quality standards.

3.6 FURTHER READINGS

Software Engineering: A Practitioner's Approach, R. S. Pressman, McGraw Hill.

Software Engineering Best Practices: Lessons from Successful Projects in the Top Companies, Capers Jones, McGraw Hill.

Reference Websites

http://en.wikipedia.org/wiki/Software_configuration_management

http://en.wikipedia.org/wiki/Software_maintenance

http://en.wikipedia.org/wiki/Software_quality_assurance

MPDD-IGNOU/P.O. 2.0 K/Sep. 2018 (Reprint)

ISBN : 978-81-266-6612-6