UNIT 1 SOFTWARE ENGINEERING AND ITS MODELS

Structure **Page Nos.**

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.

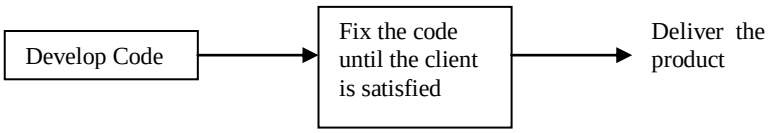

Figure 1.1 : Build and fix model

(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.

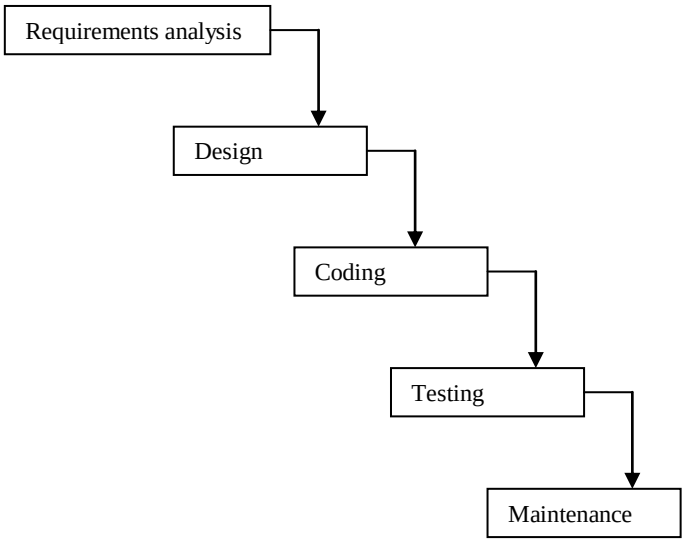

Figure 1.2 : Water fall model

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.

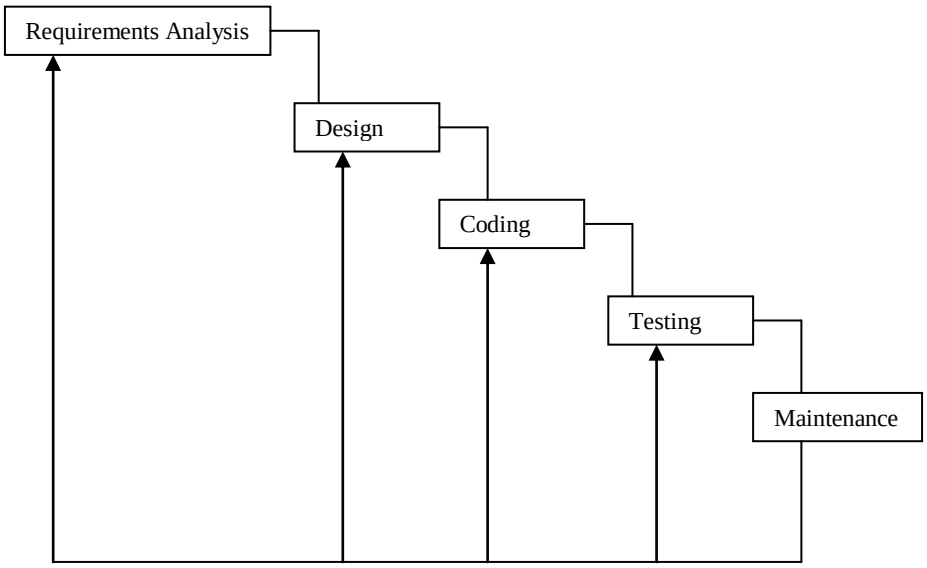

Figure 1.3 : Water fall model with feedback

(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.

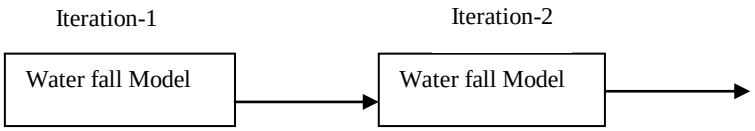

Figure 1.4 : Iterative enhancement model

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.

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.

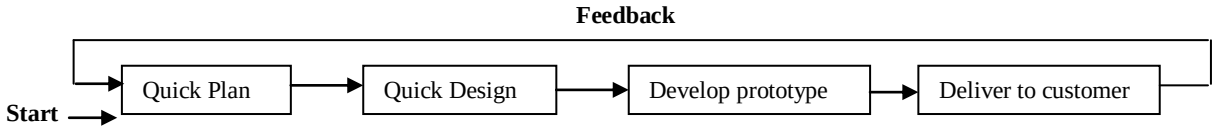

Figure 1.5: Prototyping model

(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).

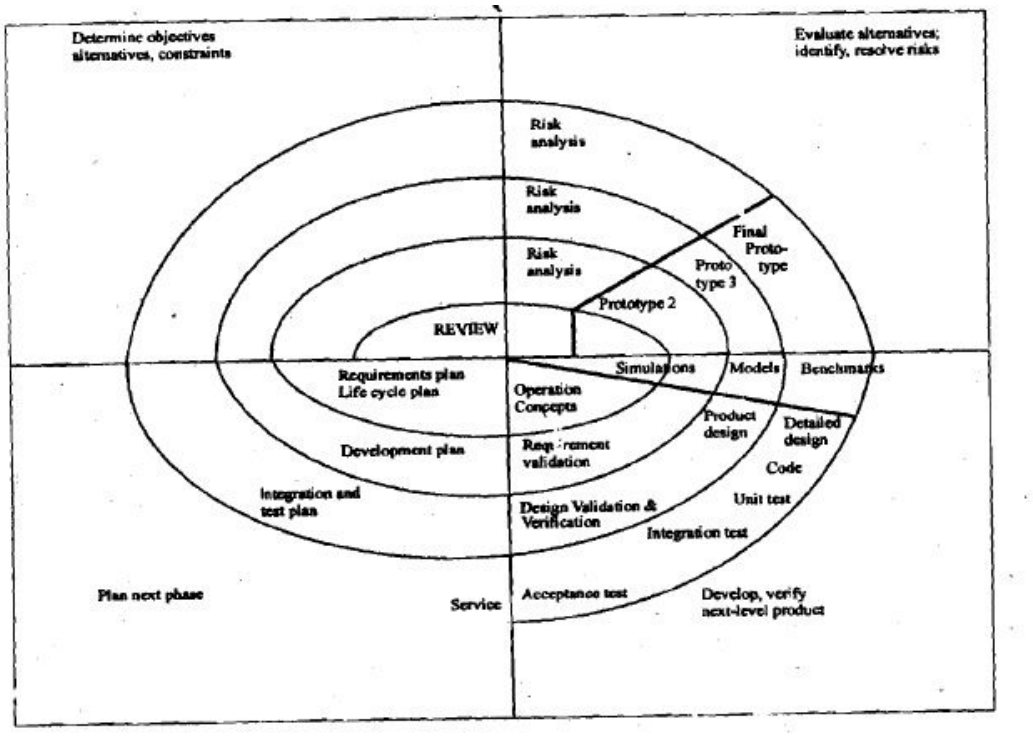

Figure 1.6: Spiral model

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 Levels

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 Areas

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 & 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 & 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

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 subsystems, 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 SOLUTIONS/ANSWERS

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

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

2) *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