INTRODUCTION TO SOFTWARE ENGINEERING

AFTER STUDYING THIS CHAPTER YOU WILL LEARN ABOUT

- Software crisis.
- What is software engineering?
- Different stages of software development process.
- Why Software Development Lifecycles (SDLC) are important?
- Popular Software Development Lifecycles used for developing software.
- Selecting a SDLC for a project.
- Types of software.

1.1 INTRODUCTION

In the current scenario, information systems are important part of any organization. As compared to 1970's and 1980's, they are becoming more and more complex. In the early years i.e., 1940's, software development was not an independent established discipline. Instead it was only an extension of the hardware. Earlier programs were written mostly in assembly language and were not complex. The persons/users who did programming were the one who also executed, tested and fixed the problems/errors in the software.

As the information systems became more and more complex and organizations became more dependent on software, a need was felt to develop the software in a systematic fashion. A survey was conducted by researchers in seventies and it was found that
most of the software used by companies was of poor quality. Also companies were spending most of their time and money in maintaining the software. They found that software product was not same as hardware product as it was to be engineered or developed and it did not wear out. Programmers were clear about civil, mechanical, electrical and computer engineering but it was always a topic of debate that what engineering might mean for software.

Most of the people consider the program and software to be same. But software not only consists of programs but also the supporting manuals.

---

Software can be defined as a set of instructions which when executed on a computer accepts the inputs and after doing required computations produces the output or result as per users requirements. Additionally it is also accompanied by the user manual so as to understand the features and working of the software.

Hence the software consists of following:

- Source code
- Executables
- User manuals
- Requirement analysis and design documents
- Installation manuals

Some important characteristics of software are:

- As compared to hardware which follows the “bathtub curve” as shown in Fig. 1.1, software does not wear out as over a period of time it will become more reliable.

![Fig. 1.1 Bath Tub Curve](image)

Instead the software becomes obsolete due to new operating environment, new user requirements etc.. Hence it can only retire but not wear out. So it should follow the curve shown in Fig. 1.2.
Just like hardware or any other engineering product, software is not manufactured. Instead it is developed or engineered. It is built only once and then multiple copies can be produced. On the other hand in other streams every product has to be made from the start using the new components.

Most software is custom-built, rather than being assembled from existing components.

It is flexible and hence can easily accommodate the changes. If changes are not accommodated in a planned manner it can also led to failure of the product.

1.2 SOFTWARE CRISIS

From 1960’s to 1980’s, software engineering was spurred by so called software crisis. A number of large size projects failed called software runaways because of following reasons:

- Development teams exceeding the budget
- Late delivery of software
- Poor quality
- User requirements not completely supported by the software
- Difficult maintenance
- Unreliable software

Statistics show that only 2% of the projects were used as they were delivered, 3% of the projects used after modifications, 47% of the software was never used only delivered, 19% of the software rejected or reworked and 29% was not even delivered. The problems increased because of increased dependence of business on software and lack of systematic approach to build the software. Developers and researchers realized that development of software was not an easy and straight forward task, instead it required lot of engineering principles.
Some of the examples of software project failures are listed below:

- In June 1996, Anane 5 launcher broke up and exploded after 40 seconds of takeoff at an attitude of less than 4 kilometers. The total loss was $500 million. It was found that the error was due to overflow in the conversion from a 64 bit floating point number to 16 bit signed integer.
- In early eighties, M/s Sperry Corporations of US was hired by the Internal Revenue Service to automate the processing of income tax forms but as system was inefficient to handle the load it had to be replaced. As a result $90 million was spent in addition to the original $103 million to enhance the Sperry Corporation equipment. Internal Revenue Service paid $40.2 million as interest to customers and $21.3 million in overtime wages to its employees.
- Another shocking case of improperly designed software is about Therac-25 a radiation therapy and X-ray machine. This machine killed several patients due to malfunctioning of arrow keys which were not programmed properly by the designers. As a result high dose of radiation was given to patients whereas only low levels were required.
- In early nineties British House of Common Public Accounts Committee, highlighted in a report – an overspend of £4 billion pounds as a result of project cost over-runs on an annual budget of £8.2 billion, the software being the major culprit.
- Ministry of Agriculture in UK alone had to undergo a loss of 12 million pounds because of software errors.
- Even the launch of space shuttle Columbia was delayed by three years thus costing millions of dollars.

Developers and researchers found that there was lot of scope for building quality software. As a result of this a need was felt to systematize the development of software. A NATO Science Committee sponsored two conferences in 1968 (in Garnish, Germany) and 1969. Many believed that these conferences marked the official start of the software engineering profession. The conference focused on the design and development of techniques for producing error free software. The concept of software life cycle was also introduced to depict the different stages of software development.

1.3 NO SILVER BULLET

A lot of methodologies and automated tools were introduced to support different stages of software life cycle from 1970’s to 1990’s. As finding the solution to software crisis was the main objective of the researchers and development organizations, every new tool or technology proposed from 1970’s to 1990’s was used as a silver bullet to solve the software crisis. In fact whereas in seventies lot of software engineering principles were introduced, in eighties they were supported by CASE (Computer Aided Software Engineering) tools. Following were touted as silver bullets:
Structured Programming Languages
- Object Oriented Programming Languages like Java, C++
- Formal methods like Modern Structured Analysis, Object Modeling Technique, Unified Modeling Language
- Process Models like Software Engineering Institute’s Capability Maturity Model (CMM), SPICE (Software Process Improvement and Capability Determination),
- Standards representing requirements, design etc.
- Professionalism on the part of team members
- Project management techniques and many more

It was also found by the practitioners that no single silver bullet can solve all the problems of software engineering disciplines. It is only the combination of some of them which can make the projects succeed.

1.4 WHAT IS SOFTWARE ENGINEERING?

Over the years software developers have understood that software development is not merely coding. It is something which starts long before one actually starts programming and continues even after the first version of the software is delivered. Hence it consists of number of activities in addition to programming. Software engineering as a discipline provides methods of systematically developing and maintaining that software.

A number of definitions of software engineering can be found in literature. Some of the definitions are given below:

According to (Pfleeger 87) “Software engineering is a strategy for producing quality software.”

Still another definition of software engineering given at NATO Conference is:

“It is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.”

Software engineering can also be defined as the application of scientific principles to

(i) Systematic transformation of a problem into a working software solution.

(ii) The subsequent maintenance of that software after delivery until the end of its life.

According to IEEE standard 610.12-1990 software engineering is defined as

1. The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software i.e., the application of engineering to software.

2. The study of approaches as in (1).
In short software engineering as a discipline provides tools and techniques for developing the software in an orderly fashion. The advantages of using software engineering for developing software are:

- Improved quality
- Improved requirement specification
- Improved cost and schedule estimates
- Better use of automated tools and techniques
- Less defects in final product
- Better maintenance of delivered software
- Well defined processes
- Improved productivity
- Improved reliability

1.4.1 How Software Engineering is Different from Other Professions?

The area of computer science is concerned with theory and fundamentals and focuses on developing new algorithms, data structures, languages, tools and techniques. Software engineering uses these concepts and techniques to build the application and hence is concerned with the practicalities of developing the software.

Another important area is that of systems engineering. Systems engineering is concerned with all the aspects of computer based systems development and hence includes software, hardware and processes as well. Hence software engineering is a part of systems engineering and focuses on developing the software specification, architectural design, testing and deployment of software part of the system.

1.5 SOFTWARE DEVELOPMENT PROCESS

Developers should be able to deliver good quality software to the end users using a well defined, well managed, consistent and cost-effective process. A software process framework therefore describes the different phases of the project via the activities performed in each phase without telling about the sequence in which these phases or activities will be conducted.

A software life cycle model is a type of process that represents the order also in which the activities will take place. It describes the sequence of activities graphically to build the final product. As different phases may follow one another, repeat themselves or even run concurrently, the sequence may or may not be linearly sequential.

The different phases of software development process are shown in Fig. 1.3. Each of these phases will have different activities to meet its objectives. The process starts by collecting the user requirements and representing them in a suitable form which must be understandable by developers, analyst, users, testers and all other stake holders of the project. This stage is called the requirements analysis stage and the output of this stage
is a document called Requirement Specification Document (RSD) or Software Requirement Specification (SRS). System design stage focuses on high level design of software. In program implementation stage actual coding is done and thereafter product is tested to ensure an error free product. Once the product is delivered, it’s maintenance is done by the organization. In rest of the book we will discuss different phases of the software development life cycle in detail.

1.6 WHY ARE SOFTWARE LIFE CYCLE MODELS IMPORTANT?

The duration of time that begins with conceptualization of software being developed and ends after system is discarded after its usage, is denoted by Software Development Life Cycle (SDLC). A number of software life cycle models have been proposed by the researchers to organize the software engineering activities into phases. While adopting a software process for developing a product, the question which immediately comes to the mind is that whether the process is the right process to be adopted which will ensure a good quality product. It is normally seen that as complexity and size of the project increases, need for a formal process also increases. In order to deliver a high quality software project managers and their team members, end users and other stake holders of the project must agree on common terminology and a software development life cycle model. The model thus selected guides all the stake holders of the project in monitoring their progress. Hence model to be used in any project must be carefully selected. In the next section characteristics of popular software development lifecycle models are highlighted which guide the project members in selecting the appropriate model for their project.
1.7 SOFTWARE DEVELOPMENT LIFE CYCLE MODELS

Some of the widely used, well known software life cycle models are the Waterfall model, V model, Prototyping model, Incremental model, Spiral model etc. Depending upon the scope, complexity and magnitude of the project a particular software life cycle model is selected and this selection of life cycle model significantly contributes towards the successful completion of the project. In the following sections we will be briefly discussing each of these models.

1.7.1 Build and Fix Model

Techniques used in the initial years of software development resulted into the term Build and Fix model. In fact the model resulted in a number of project failures because the product was not constructed using proper specification and design. Instead the product was reworked number of times in order to satisfy the clients as shown in Fig. 1.4. This model has only historical importance now.

The advantages and disadvantages of the model are:

**Advantages**

- The model is useful only for small size projects, in fact only for programming exercise 100 or 150 lines long.

**Disadvantages**

- The model is not suitable for large projects.
- As specifications are not defined, it results into product full of errors.
- Reworking of product results into increased cost.
- Maintenance of the product is extremely difficult.

![Fig. 1.4 Build and Fix Model](image-url)
1.7.2 The Waterfall Model

The Waterfall model is one of the most used model of 70’s. It was proposed by Royce in 1970 as an alternative to Build and Fix software development method in which code was written and debugged. System was not formally designed and there was no way to check the quality criteria. Different phases of Waterfall model are shown in Figure 1.5.

Given below is a brief description of different phases of Waterfall model.

❖ **Feasibility study** explores system requirements to determine project feasibility. All projects are feasible given unlimited resources and infinite time (Pressman92). Feasibility can be categorized into
  - Economic feasibility
  - Technical feasibility
  - Operational feasibility
  - Schedule feasibility
  - Legal and contractual feasibility
  - Political feasibility

![Fig. 1.5 The Waterfall Model](image)

Economic feasibility is also called *cost-benefit analysis* and focuses on determining the project costs and benefits. Benefits and costs can be both tangible as well as
intangible. Tangible costs and benefits are the ones which can be easily measured whereas intangible ones cannot be easily measured. Examples of tangible benefits are reduced errors, improved planning and control, reduced costs etc. Intangible benefits include timely information, better resource control, improved information processing, better assets utilization and many more. Similarly tangible costs include cost of hardware and software required in the project, training costs, operational costs and labor costs whereas intangible costs include loss of customer goodwill, decreased operation efficiency and many more. Economic feasibility uses the concept of time-value of money (TVM) which compares the present cash outlays to future expected returns.

Technical feasibility focuses on organization’s ability to construct the proposed system in terms of hardware, software, operating environments, project size, complexity, experience of the organization in handling the similar types of projects and the risk analysis. Operational feasibility deals with assessing the degree to which a proposed system solves business problems. Similarly schedule feasibility ensures that the project will be completed well in time. Legal and contractual feasibility relates to issue like intellectual property rights, copyright laws, labor laws and different trade regulations. Political feasibility finally evaluates how the key stakeholders within the organization view the proposed system.

- **Requirements analysis** phase focuses on understanding the problem domain and representing the requirements in a form which are understandable by all the stakeholders of the project i.e. analyst, user, programmer, tester etc. The output of this stage is a document called Requirements Specification Document (RSD) or Software Requirements Specification (SRS). All the successive stages of software life cycle are dependent on this stage as SRS produced here is used in all the other stages of the software lifecycle.

- **System design** phase translates the SRS into the design document which depicts the overall modular structure of the program and the interaction between these modules. This phase focuses on the high level design and low level design of the software. High level design describes the main components of a software and their externals and internals. Low level design focuses on transforming the high level design into a more detailed level in terms of a algorithms used, data structures used etc.

- **Implementation** phase transforms the low level design part of software design description into a working software product by writing the code.

- **Testing phase** is responsible for testing the code written during implementation phase. This phase can be broadly divided into unit testing (tests individual modules), integration testing (tests groups of interrelated modules) and system testing (testing of system as a whole). Unit testing verifies the code against the component’s high level and low level design. It also ensures that all the statements in the code are executed at least once and branches are executed in all directions.
Additionally it also checks the correctness of the logic. Integration testing tests the inter modular interfaces and ensures that the module drivers are functionally complete and are of acceptable quality. System testing validates the product and verifies that the final product is ready to be delivered to the customers. Additionally several tests like volume tests, stress tests, performance tests etc., are also done at the system testing level.

- **Deployment phase** makes the system operational through installation of system and also focuses on training of user.
- **Operations and maintenance** phase resolves the software errors, failures etc., enhances the requirements if required and modifies the functionality to meet the customer demands. This is something which continues throughout the use of product by the customer.

Advantages and disadvantages of the Waterfall model are listed below:

**Advantages**

- Easy to understand even by non-technical persons i.e., customers.
- Each phase has well defined inputs and outputs e.g., input to system design stage is Requirement Specification Document (RSD) and output is the design document.
- Easy to use as software development proceeds.
- Each stage has well defined deliverables or milestones.
- Helps the project manager in proper planning of the project.

**Disadvantages**

- The biggest drawback of Waterfall model is that it does not support iteration. Software development on the other hand is iterative i.e., while designing activities are being carried out, new requirements can come up. Similarly while product is being coded, new design and requirement problems can come up.
- Another disadvantage of Waterfall model is that it is sequential in nature. One cannot start with a stage till preceding stage is completed e.g., one cannot start with the system design till all the requirements are understood and represented.
- Users have little interaction with the project team. Their feedback is not taken during development.
- Customer gets opportunity to review the product very late in life cycle because the working version of product is available very late in software development life cycle.
- Model is very rigid because output of each phase is prerequisite for successive stage.
- The Waterfall model also has difficulty in accommodating changes in the product after the development process starts.
- Amount of documentation produced is very high.
- The model in no way supports delivery of system in pieces.
- The model is not suitable for new projects because of uncertainty in the specifications.

![Figure 1.6: The Waterfall Model with Feedback](image)

Though Waterfall model has been used for large projects in the past, its use must be limited to projects in which requirements are well understood or the company is working on a product of similar kind which it has developed in the past. Modified version of Waterfall model shown in Fig. 1.6 allows feedback to preceding stages and hence is not very rigid. This model clearly shows that the development team can go back to previous phases in order to have better clarity and understanding.

The Waterfall model is suited for well understood projects using familiar technology. It can also be used for existing projects if changes to be made are well defined.

### 1.7.3 The V-Model

This model was developed to relate the analysis and design activities with the testing activities and thus focuses on verification and validation activities of the product. As this model relates the analysis and design phase to testing phase, testing activities are planned early in software lifecycle as shown in the Fig. 1.7.
The dotted lines in the figure 1.7 indicate that the corresponding phases must be carried out in parallel. As in the case of waterfall model, V model should be used, when all the requirements of the project are available in the beginning of the project. The advantages and disadvantages of the model are listed below.

### Advantages
- The model is simple and easy to use.
- The V model focuses on testing of all intermediate products, not only the final software.
- The model plans for verification and validation activities early in the life cycle thereby enhancing the probability of building an error free and good quality product.

### Disadvantages
- The model does not support iteration of phases and change in requirements throughout the life cycle.
- It does not take into account risk analysis.

The V model is used for systems in which reliability is very important e.g., systems developed to monitor the state of the patients, software used in radiation therapy machines.

#### 1.7.4 The Prototype Model
The concept of prototyping is not new in various streams of engineering. A prototype is a partially developed product. Robert T. Futrell and Shafer in their book Quality Software Project Management define prototyping as a process of developing working replica of a system(Robert02). This activity of prototyping now forms the basis of prototype software development life cycle model. Most of the users do not exactly know what they want until they actually see the product. Prototyping is used for developing a mock-up of product and is used for obtaining user feedback in order to refine it further as shown in Fig. 1.8.
In this process model, system is partially implemented before or during analysis phase thus giving the end users an opportunity to see the product early in the life cycle. The process starts by interviewing the users and developing the incomplete high level paper model. This document is used to build the initial prototype supporting only the basic functionality as desired by the user. The prototype is then demonstrated to the user for his/her feedback. After user pinpoints the problems, prototype is further refined to eliminate the problems. This process continues till the user approves the rapid prototype and finds the working model to be satisfactory.

Two approaches of prototyping can be followed:

(i) **Rapid Throwaway Prototyping:** This approach is used for developing the systems or part of the systems where the development team does not have the understanding of the system. The quick and dirty prototypes are built, verified with the customers and thrown away. This process continues till a satisfactory prototype is built. At this stage now the full scale development of the product begins.

(ii) **Evolutionary Prototyping:** This approach is used when there is some understanding of the requirements. The prototypes thus built are not thrown away but evolved with time. The block diagram of the prototype model is shown in Fig. 1.9. The concept of prototyping has also led to the Rapid prototyping model and the Spiral model.

The advantages and disadvantages of the prototyping model are listed below:

**Advantages**

- A partial product is built in the initial stages. Therefore customers get a chance to see the product early in the life cycle and thus give necessary feedback.
- New requirements can be easily accommodated, as there is scope for refinement.
- Requirements become more clear resulting into an accurate product.
- As user is involved from the starting of the project, he tends to be more secure, comfortable and satisfied.
- Flexibility in design and development is also supported by the model.

**Disadvantages**

- After seeing an early prototype end users demand the actual system to be delivered soon.
End users may not like to know the difference between a prototype and a well engineered fully developed system.

- Developers in a hurry to build prototypes may end up with sub-optimal solutions.
- If not managed properly, the iterative process of prototype demonstration and refinement can continue for long duration.
- If end user is not satisfied with initial prototype, he may loose interest in the project.
- Poor documentation.
Prototype model should be used when requirements of the system are not clearly understood or are unstable. It can also be used if requirements are changing quickly. This model can be successfully used for developing user interfaces, high technology software intensive systems, and systems with complex algorithms and interfaces. It is also a very good choice to demonstrate technical feasibility of the product.

### 1.7.5 The Incremental Software Development Life Cycle Model

Software like all other complex systems is bound to evolve due to changing business requirements or new requirements coming up. Hence there is a need to have a model which can accommodate the changes in the product. The models discussed earlier do not take into consideration the evolutionary nature of the product. Evolutionary models are also iterative in nature. The incremental software development life cycle model is one of the popular evolutionary software process model used by industry.

The incremental model releases the product partially to the customer in the beginning and slowly adds increased functionality to the system. That is why the model is called incremental model. The model prioritizes the system requirements and implements them in groups. Each new release of the system enhances the functionality of the previously released system thereby reducing the cost of the project. The Fig. 1.10 shows the working of the incremental model.

In the first release only the functionality A of the product is offered to the customer. The functionality A here consists of core requirements which are critical to the success of the project. In the second release functionality A plus functionality B is offered and finally in release 3 functionality A, B as well as C is offered. Therefore, with each release in addition to incorporating new functionality in the system, functionality of earlier releases may also be enhanced. For example if a text editor is to be built, first version of the
The product will have basic editing facilities. The next version of the product can be released with enhanced editing facilities like formatting along with new features like spell checking. The incremental model is used when requirements are defined at the beginning of the project but at the same time they are expected to evolve over time. It can also be used for projects with development schedules more than one year or if deliveries are to be made at regular intervals. The advantages and disadvantages of incremental model are listed below:

**Advantages**
- As product is to be delivered in parts, total cost of project is distributed.
- Limited number of persons can be put on project because work is to be delivered in parts.
- As development activities for next release and use of early version of product is done simultaneously, if found errors can be corrected.
- Customers or end users get the chance to see the useful functionality early in the software development life cycle.
- As a result of end user’s feedback requirements for successive releases become more clear.
- As functionality is incremented in steps, testing also becomes easy.
- Risk of failure of a product is decreased as users start using the product early.

**Disadvantages**
- As product is delivered in parts, total development cost is higher.
- Well defined interfaces are required to connect modules developed with each phase.
- The model requires well defined project planning schedule to distribute the work properly.
- Testing of modules also results into overhead and increased cost.

### 1.7.6 The Spiral Model

The Spiral model is also one of the popular evolutionary process model used by the industry. The Spiral model was proposed by Boehm in 1988 and is a popular model used for large size projects. The model focuses on minimizing the risk through the use of prototype. One can view the Spiral model as a Waterfall model with each stage preceded by the risk analysis stage. A simplified view of Spiral model is shown in Fig. 1.11.
The radial coordinate in the diagram represents the total costs incurred till date. Each loop of the spiral represents one phase of the development. The model is divided into four quadrants, each with a specific purpose. Each spiral represents the progress made in the project. In the first quadrant, objectives, alternative means to develop product and constraints imposed on the product are identified. The next quadrant (right upper) deals with identification of risks and strategies to resolve the risks. The third bottom right quadrant represents the Waterfall model consisting of activities like design, detailed design, coding and test. With each phase after customer evaluates the product, requirements are further refined and so is the product. It is to be noted that number of loops through the quadrants are not fixed and vary from project to project.

The steps followed while using Spiral model can be described as follows. The process starts with identification and prioritization of risks. A series of prototypes are developed for the risks identified (highest risk is considered first). For each development cycle of the Spiral model shown in Fig. 1.11 Waterfall model is used. If a risk is resolved successfully planning for the next cycle is done. If at some stage risk cannot be resolved project is terminated.

Spiral model is also termed as process model generator or meta model. For example if any project requirements are not clear models like Prototyping or Incremental can be derived from the spiral model. The advantages and disadvantages of spiral model are listed below:

**Advantages**

- The model tries to resolve all possible risks involved in the project starting with the highest risk.
- End users get a chance to see the product early in life cycle.
- With each phase as product is refined after customer feedback, the model ensures a good quality product.
❖ The model makes use of techniques like reuse, prototyping and component based design.

**Disadvantages**
❖ The model requires expertise in risk management and excellent management skills.
❖ The model is not suitable for small projects as cost of risk analysis may exceed the actual cost of the project.
❖ Different persons involved in the project may find it complex to use.

Spiral model is generally used for large projects with medium to high risk because in small projects it is possible that cost of risk analysis may exceed the actual cost of project. In other words, Spiral model is a practical approach for solving large scale software development related problems.

This can also be used if requirements of the project are very complex or if the company is planning to introduce new technologies. Some areas where Spiral model is successfully used are decision support system, defense, aerospace, and large business projects.

1.7.7 The Rapid Application Development (RAD) Model

The Rapid Application Development (RAD) model was proposed by IBM in 1980s and later on was introduced to software community by James Martin through his book Rapid Application development. The important feature of RAD model is increased involvement of the user/customer at all stages of life cycle through the use of powerful development tools.

If the requirements of the software to be developed can be modularized in such a way that each of them can be completed by different teams in a fixed time then the software is a candidate for RAD. The independent modules can be integrated to build the final product. The important feature of the RAD model is quick turnaround time from requirement analysis to the final delivered system. The time frame for each delivery is normally 60 to 90 days called *time box* which is achieved by using powerful developer tools like Visual C++, JAVA, Visual BASIC, XML, .NET etc. Block diagram of RAD model is shown in Fig. 1.12.

The RAD model consists of following four phases:
❖ Requirements Planning – focuses on collecting requirements using elicitation techniques like brainstorming,
❖ User Description – Requirements are detailed by taking users feedback by building prototype using development tools.
❖ Construction – The prototype is refined to build the product and released to the customer.
❖ Cutover – involves acceptance testing by the user and their training.
The process therefore starts with building a rapid prototype (a working model which is functionally equivalent to a subset of final product) and is delivered to customer for use and his feedback. Once the user/customer validates the rapid prototype after using it, Requirement Specification Document is derived and design is done to give final shape to the product. After the product is installed, maintenance of the product is continued by refining the requirements, specification, design or coding phase.

The advantages and disadvantages of RAD model are discussed below:

**Advantages**
- As customer is involved at all stages of development, it leads to a product achieving customer satisfaction.
- Usage of powerful development tools results into reduced software development cycle time.
- Feedback from the customer/user is available at the initial stages.
- Makes use of reusable components, to decrease the cycle time.
- Results into reduced costs as less developers are required.

**Disadvantages**
- The model makes use of efficient tools, to develop the prototype quickly, which calls for hiring skilled professional.
- Team leader must work closely with developers and customers/users to close the project in time.
- Absence of reusable components can lead to failure of the project.

The RAD model should be used for system with known requirements and requiring
short development time. Also it is appropriate to use RAD model for the system that can be modularized and also reusable components are available for development. The model can also be used when already existing system components can be reused in developing a new system with minimum changes.

Comparison of different process models in tabular form is shown in Table 1.1.

<table>
<thead>
<tr>
<th></th>
<th>Waterfall</th>
<th>V Model</th>
<th>Incremental</th>
<th>Spiral</th>
<th>Prototype</th>
<th>RAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Well defined</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>requirements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Domain knowledge of the</td>
<td>Adequate</td>
<td>Adequate</td>
<td>Adequate</td>
<td>Very less</td>
<td>Very less</td>
<td>Adequate</td>
</tr>
<tr>
<td>team members</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Expertise of users in</td>
<td>Very less</td>
<td>Very less</td>
<td>Adequate</td>
<td>Very less</td>
<td>Adequate</td>
<td>Adequate</td>
</tr>
<tr>
<td>problem domain</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Availability</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>of reusable components</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Users involvement in</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>all phases of SDLC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Complexity of system</td>
<td>Simple</td>
<td>Simple</td>
<td>Complex</td>
<td>Complex</td>
<td>Complex</td>
<td>Medium</td>
</tr>
</tbody>
</table>

1.8 TYPES OF SOFTWARE

The software is being used in almost all the spheres of human life e.g., hospitals, banks, defense, finance, predicting stock rates, making pictures, running other software and so on. In fact the list is endless. Though it is somewhat difficult to categorize the software in different types but based on their applications the software can be categorized into following areas:

- System software
- Scientific software
- Networking and web applications software
- Embedded software
- Business software
- Utilities software
- Artificial intelligence software

**System Software:** Systems software is necessary to manage the computer resources and support the execution of application programs. Software like operating systems, compilers,
Software editors and drivers etc., come under this category. Operating systems are needed to link the machine-dependent needs of a program with the capabilities of the machine on which it runs. Compilers translate programs from high-level languages into machine languages. Without the presence of system software, a computer cannot function.

**Scientific Software:** Scientific and engineering software satisfies the needs of a scientific or engineering user to perform enterprise-specific tasks. Such software are written for specific applications using the principles, techniques and formulae specific to that field. Examples are software like MATLAB, AUTOCAD, PSPICE, ORCAD etc.

**Networking and Web Applications Software:** Networking software provides the required support necessary for computers to interact with each other, and with data storage facilities, in a situation where multiple computers are necessary to perform a task. The networking software is also used when software is running on a network of computers (such as the Internet or the World Wide Web). This category of software include all network management software, server software, security and encryption software and software to develop Web based applications like HTML, PHP, XML etc.

**Embedded Software:** This type of the software is embedded into the hardware normally in the Read Only memory (ROM) as a part of large system and is used to support certain functionality under the control conditions. Examples are software used in instrumentation and control applications, washing machines, satellites etc.

**Business Software:** This category of software is used to support the business applications and is the most widely used category of software. Examples are software for inventory management, accounts, banking, hospital, schools, stock markets etc., The software written for Enterprise Resource planning (ERP), project management, workflow management etc., also come under this category.

**Utilities Software:** The programs coming under this category perform specific tasks and are different from other software in terms of size, cost and complexity. Examples are anti-virus software, voice recognition software, compression programs etc.

**Artificial Intelligence Software:** Software like expert systems, decision support systems, pattern recognition software, artificial neural networks etc., come in this category of software. Such type of software solve complex problems which are not affected by complex computations using non numerical algorithms.

In terms of **copyright**, there are four broad types of software:

- Commercial
- Shareware
- Freeware
- Public domain

**Commercial** software represents the majority of software which we purchase from software companies, commercial computer stores, etc. In this case when we buy the software, we actually acquire a license to use it, but not own it. Users are not allowed
to make the copies of the software. The copyright of the program is owned by the company. **Shareware** software is also covered by copyright, but the copyright holders for shareware allow purchasers to make and distribute copies of the software, but demand that if, after testing the software, purchaser adopts it for use, then he must pay for it. In both the types of software changes to the software is not allowed and derivative works is also not allowed without the permission of the copyright holder.

**Freeware** software is also covered by copyright subject to the conditions defined by the holder of the copyright. In general, according to freeware software licenses, copies of the software can be made for both archival and distribution purposes but in this case distribution cannot be for making profit. Derivative works and modifications to the software are allowed and encouraged. Decompiling of the program code is also allowed without the explicit permission of the copyright holder. In the case of **Public domain** software the original copyright holder explicitly relinquishes all rights to the software. Hence software copies can be made for both archival and distribution purposes with no restrictions as to distribution. Similarly modifications to the software are allowed and reverse engineering of the program code is allowed. Development of new works built upon the package is allowed without conditions on the distribution.

In addition to above there are software with **academic** licenses under which the owner of the software allows the universities, colleges and schools to use the software for education and research purpose at a very nominal price. But the users in this case cannot use the software commercially. The copyright of the software is also owned by the company.

**SUMMARY**

- Software worth billions and trillions of dollars have gone waste in the past due to lack of proper techniques used for developing software resulting into software crisis.
- Software engineering is application of engineering principles to develop quality software.
- The duration of time that begins with conceptualization of software being developed and ends after system is discarded after its usage is called Software Development Life Cycle (SDLC).
- A number of software process models are proposed to organize software engineering activities into phases.
- Main phases of software development life cycle are—requirements analysis, system design, coding, testing, delivery and maintenance.
- Popular process models are waterfall model, V model, incremental model, Spiral model, Prototyping model etc.
- Several process maturity frameworks and quality standards are proposed to improve software quality e.g., CMM, SPICE, Bootstrap, ISO9001 etc.
REVIEW PROBLEMS

1. Define software engineering.
2. List the impact of software engineering on developing software.
3. What are the advantages of software process models?
4. What is software crisis? Discuss the reasons which resulted into software crisis.
5. Draw a schematic diagram to represent the incremental software development life cycle. List the advantages and disadvantages of Incremental process model.
6. What is a prototype? What are the advantages of prototyping software development life cycle model?
7. What is the role of risk management activity in Spiral model?
8. Give examples of software for which spiral model is not useful.
9. Define a prototype. Under what conditions prototyping model is preferred over other models.
10. Why feasibility study is important in a project.
11. Explain the following terms – (i) Technical feasibility, (ii) Economic feasibility, (iii) Legal and contractual feasibility.
12. Spiral model is often termed as meta-model. Justify this statement.
13. Can Spiral model be used for small sized projects? Justify your answer.
14. What do the different cycles in the Spiral model indicate?
15. What process model you will follow for developing
   (a) Editor
   (b) CAD software
   (c) Radiation therapy machine software
   (d) A game
   (e) A hospital management system
   (f) A compiler for new language
   Justify your answer.
16. Classify the software based on the copyright. Make a list of software you are using in the laboratory or home and label them based on the copyright classification.
17. What are the different phases of waterfall model? Explain, which phase of the waterfall model requires maximum effort?