Unit 5 – Chapter 12

Software Maintenance

Prerequisites:

Software development life cycle, software testing

Unit Outcomes:

The objective of this chapter is to introduce the concept of software maintenance and CASE tools. At the end of this chapter, students will be able to:

- Understand the software as an evolutionary entity and recognize the need for maintenance.
- Classify maintenance and estimate the cost of maintenance.
- Analyze the features of software re-engineering and reverse engineering
- Explain configuration management activities
- Identify the primary reasons for using a CASE tool and CASE environments.

Software maintenance is necessary for making changes to a software product after its delivery to the client. Every software product needs maintenance once it is in use. When a developed software is run on a new platform or hardware, the components and interface of the software need revision. The new environment such as an enhanced operating system requires the lower component of the software to be advanced. Another reason for software maintenance is the necessity to fix the errors due to regular or extensive usage by the client. The development of the software does not end after its delivery, but it continues after deployment. The software changes due to business status changes or users' expectations. **Software evolution** refers to the modification of the characteristics of a software product over generations.

12.1 Software Evolution

The reasons for software evolution are changes in the requirements of the client's business, defects in the existing system, or changes in the system environment. Some parts of the software may create errors such as incorrect operations or computation.

Some of the non-functional requirements may get neglected in the developed software. This means software evolution is a must to respond to the demands of change. Software evolution is essential as organizations invest a large amount of money in a business and they would prefer modification of the existing system rather than developing a new system from the scratch.

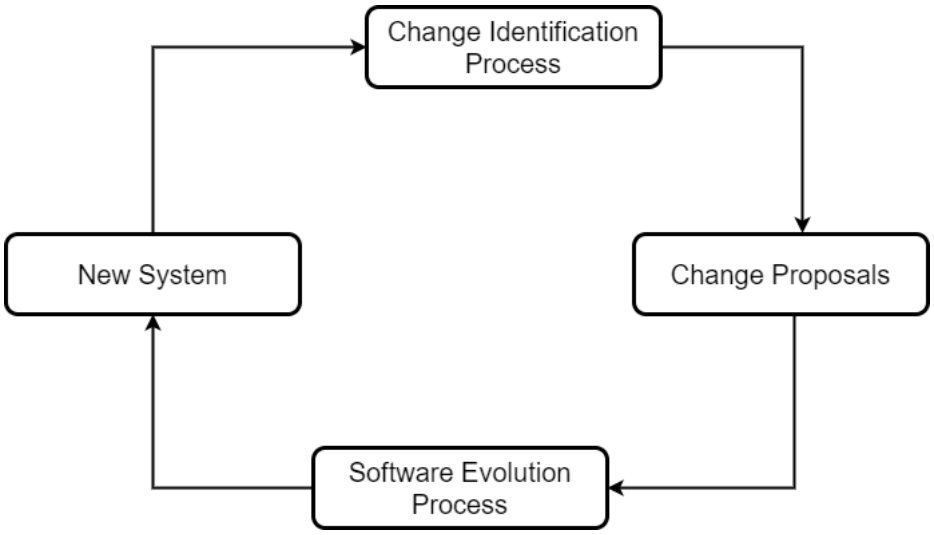

Figure 12.1 Evolution Process

Software evolution can be considered a cyclic process as shown in Figure 12.1. The new system is developed based on the type of software. The people in the system such as users of the software and software developers discuss the changes and propose the new system.

12.2 Software Maintenance: Background

Software maintenance is a general process of modifying a software product after it is delivered to the client. The modifications or changes may be necessary for the following reasons:

- To correct coding bugs
- To accommodate major changes in the design.
- To manage the new requirements.

Need for Software Maintenance: As technology is changing rapidly, customers demand to use the developed software in the present environment with advanced features. Due to changes in the hardware configurations, the operating system, controls between modules, interface with the web, and other devices in the system need modification. This is managed in software maintenance. If there is a critical fault

in the system and it must be repaired to continue with normal operations, then software maintenance is needed. If the competitors of the developed system modify their product with advanced features, then the system requirement, design, and coding of the present system need to change. ‘

The old system with outdated hardware and software, but still in use is referred to as a **legacy system**. Software maintenance mitigates such a system. Mitigation means saving the files in printed form, transferring the data on the cloud or on the virtual machine, and upgrading the existing software. Also discarding the existing system is a part of software maintenance.

Examples of maintenance: Patching in operating systems such as Windows, Apple, Linux/Unix, etc. Updates in all the commercial antivirus software, typesetting software such as Adobe, MS Office, etc.

Software maintenance is classified as follows:

1. **Corrective Maintenance:** Corrective maintenance is related to correcting the faults and errors of a software product. After the system is put in use, the customers give the reports of the bugs if any. These errors may be in the requirements, design, or coding of a system.
2. **Adaptive Maintenance:** Adaptive maintenance is used when the software environment changes. For example, a customer may use a new operating system, or a new web or mobile app-based platform to run the product. Adaptive maintenance is necessary when the new hardware is included in the existing system.
3. **Perfective Software Maintenance:** This type of maintenance is required when the requirements and the features of the system are advanced. The user may realize some new requirements or features that evolve the developed software. Removing unwanted features and enhancing the system with the user’s experience are the goals of this type of maintenance.
4. **Preventive Software Maintenance:** Future problems in the software are prevented using some modifications and updates. This is known as preventative software maintenance. It helps increase the lifetime of the software. The software code is optimized and prepared to handle the changes in the environment. It addresses the stability and maintainability features of the software.

The maintenance of the software can be efficiently achieved by a good understanding of a system. The well-documented legacy system can be easily maintained. A reverse engineering process can be used in such situations.

12.3 Reverse Engineering Process

Reverse engineering is the process of recovering the design, requirement specifications, and functions of a product from an analysis of its code. Several legacy systems are not well structured or there may be wear and tear in the software due to continuous maintenance efforts or lack of documentation. Reverse engineering helps in such cases with the maintenance of the software. The steps in the reverse engineering process are as given below.

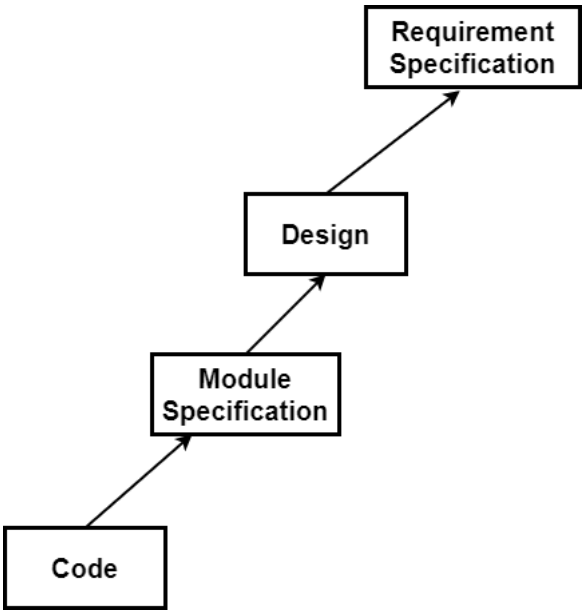

Figure 12.2 Reverse Software Engineering Process Model

1) In the first step, the code is improved for better readability, understanding, and structure. In many legacy software, the programs are not in proper structure. The variable names, and data structures, are given with meaningful names. The functionality is not changed. Complex statements are simplified, and

nested loops are replaced with multiple condition statements. These changes are known as cosmetic changes.

2) In the second step, the code is analyzed completely. The different modules and their interface are understood.
3) In the third step, the design can be extracted using tools such as structure charts, DFDs, and control flow diagrams.
4) In the last step, the SRS is written based on the design.

Software maintenance depends on what improvement is required in the product. The different resources available are identified and to what extent the project can use these resources is verified. Different reasons for software maintenance include the identification of project risks and conditions of the existing product. Some legacy software does not include proper documentation. For legacy systems, a complex maintenance process is required. The brief introduction of the maintenance process model is as follows:

12.4 Software Maintenance Process Models

The software maintenance process models are divided into two categories.

The first model is used when small changes are required in the code and are documented. In this process, a few team members collect the requirements, analyze, and formulate the strategy. In this model, the existing system is useful for modification and debugging of the errors.

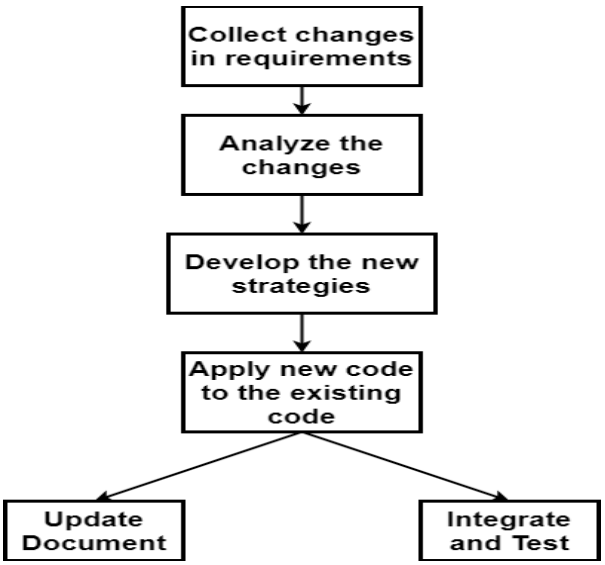

Figure 12.3: Maintenance Process Model 1

When the requirement changes considerably, the amount of rework is more. The modification in the software is done by combining reverse and forward engineering. This process is known as software reengineering. This model is presented in Figure 12.3. The legacy products are reverse engineered to get module specifications. The specifications are further used to generate the design. The design is then explored to produce requirements. The new requirements are added. Thus, a combination of two consecutive processes software reverse engineering and software forward engineering is known as **software reengineering.**

The changes in the requirements are applied in the forward engineering process. In the design and coding phase, the existing components are reused. The product is improved with better documentation. The design and code get improved in terms of changed functionality, efficiency, reliability, and usability. Software reengineering is preferred when the already developed software shows a high failure rate. When there are legacy products with very poor design and code structure, the software reengineering process is useful.

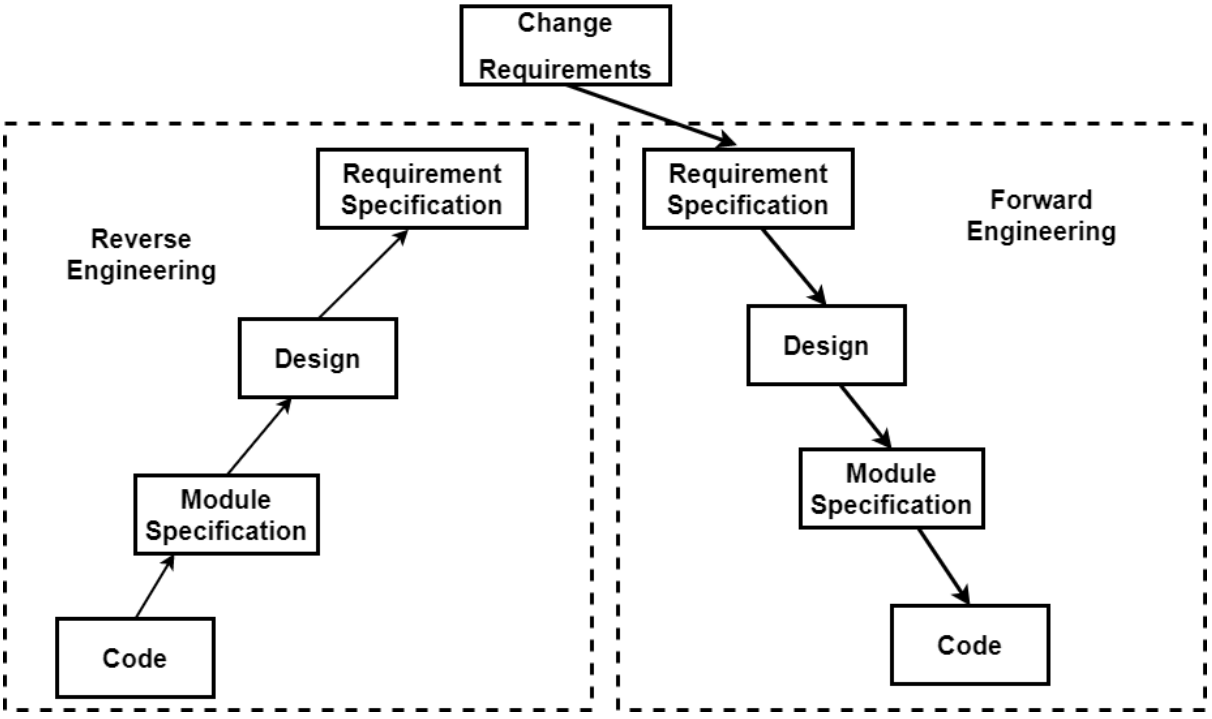

Figure 12.4: Maintenance Process Model 2

The main disadvantage of using this process model is that the reengineering process is very expensive. Thus, most of the study shows that in practice, the software

maintenance model – 1 is used. The cost of the software maintenance process can be determined as given below.

12.5 Software Maintenance Cost

In 1981, software engineering scientist Boehm proposed a **Constructive Cost Model** (COCOMO) to determine maintenance costs. The model includes annual change traffic (ACT) to determine the quality. The ACT is that small subset of software source programs that get changed in a year. ACT is determined as follows:

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

- $KLOC_{added}$: It is a measure of kilo lines of source code added during maintenance.
- $KLOC_{deleted}$: It is a measure of kilo lines of source code deleted during maintenance.
- $KLOC_{TOTAL}$: It is a measure of total lines of code instructions.
- Code changed = code added + code deleted

The ACT is used to determine the change in the code due to new requirements or modified requirements. The maintenance cost is determined using ACT and the development cost. The development cost refers to the cost of coding.

$maintenance\ cost\ =\ ACT\ \times\ development\ cost$

The maintenance cost is always determined approximately as it does not include new hardware and software complexity, the experience of the software engineer, and the familiarity of the product.

12.6 Software Configuration Management

Software configuration refers to the state of the source code, design, SRS, test reports, user manual, or guide used by software engineers during the SDLC. The state of the software product changes during its development and testing. The software gets changed due to the following reasons.

- **Release**: A software is released newly if there is any new bug or new functionality or is small changes in its usage and the technology. The addition of new features

to correct a particular error of the Windows operating system and a release of the new product is one example of a release.

- **Version**: If there is a major change in the hardware and technology in which the software product is working or the functionality of the product is drastically changed then the software configuration is revised. The new version of Android updates for supporting new features is an example of the version. Version Windows 95, Windows 2000, Windows XP, Windows 10, etc are the versions of the Windows Operating system.
- **Revision**: If there are minor bugs or errors, then the software is revised in its configuration. The revision in the typesetting features in the MS Office can be an example of the revision of configuration.

A version, release, or revision is always given concerning the history of the product, such as a variant of or revision of, etc.

12.6.1 Need for the software configuration management:

Software configuration is necessary for software updates and storage. When the objects in software are reused by many software engineers, he/she will have their copy. If one software makes changes to one object and does not inform others, then there will be inconsistent objects used by many people. This will create bugs in the software. This can be avoided with proper configuration management in place.

Also, if the software is modified by several people at the same time, then there will be overwriting of the code by many people. This can disturb the integrity of the software product.

Consider software with multiple modules managed by many software engineers. If one developer is trying to integrate the modules, whereas it has been modified by another engineer. Then the integration must be redone. This can be disappointing. Configuration management provides a freeze option with which the environment for the software development is stable. The different variants of the modules are archived in the configuration management and the respective version is given to the developer. Software configuration management maintains the status of the software product and keeps account of the changes in the software. The configuration management activities are done by the project manager using automatic tools. The configuration

activities performed in the software configuration management process (SCM) are as given below.

12.6.2 Software Configuration Management (SCM) Process

1. Configuration identification of the objects
2. Controlling versions
3. Controlling changes
4. Audit configuration
5. Generating status report

1. Configuration Identification: In this step, the objects in the software are classified into three categories: controlled, pre-controlled and uncontrolled. In the controlled category, the objects of the software are already in a configuration. The pre-controlled objects are those which will be configured in the future. Uncontrolled objects are not and will not be subjected to configuration control. Typical controllable objects are requirement specifications, design, source code, test cases, reports, etc.
2. Different tools and methods are used in version control to create different versions of the objects. Each object gets a different configuration with different features or attributes.
3. In the change control step, a request for a variation or modification to the existing object is received. How this change creates an impact on the objects and the functions of the system, modification in the cost of the system is evaluated. The change control authority (CCA) is a single person or team of people who authorize the request, and they decide whether the change is permitted in the object.
4. While changing the configuration there is an access control policy that checks whether the software engineer has the authority to change the configuration. Also, there is a synchronization control policy that makes sure that there is no overwriting by one or more persons on a single object.
5. After completing the configuration, the SCM generates audit reports which check the original requirements are fulfilled in the new deliverable product. This is done in the configuration audit step.

   The different stakeholders of the project such as the software developer, tester, and customers get a detailed report of the current configuration of the software.

The frequently asked questions (FAQs), manual, instruction or installation guide, necessary software, and hardware requirements for installation of the software are given in the audit reports.

12.7 Computer-Aided Software Engineering (CASE) Tools

CASE tools provide automated support in software engineering. The different activities such as gathering requirements, preparing software design, developing code, and testing can be performed with the help of CASE tools. The software configuration and the project management are also supported by CASE tools. The advantages of using CASE tools are they can be used at a low cost (since there is no development from the scratch) for developing quality software. The productivity of the software development gets improved using CASE tools. CASE tools are collectively stored as toolkits in a common environment. The tools with common information are integrated into a central repository. For example, a data dictionary can be used as a CASE tool for storing elementary and composite data.

The CASE environment is different from than programming environment. In the programming environment, the tools are available only for coding e.g., Java Virtual Machine supporting Java on different platforms, databases, web, etc. The CASE tools environment is for all the phases of SDLC. Different CASE tools related to the different stages in SDLC are shown in Figure 12.5.

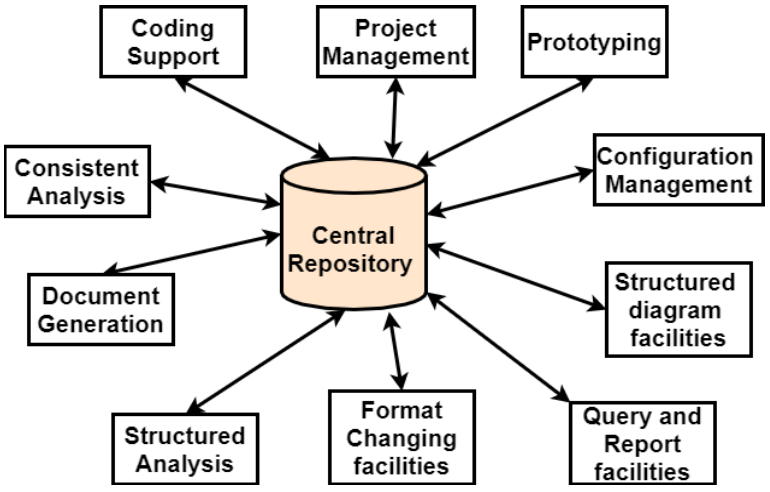

Figure 12.5 : CASE Environment

- **Prototyping CASE tools for requirements:** Prototyping CASE tools are used to understand user interaction, data storage, and retrieval, the control flow in the program, data processing at different steps, etc. during requirement collection. A

prototyping CASE tool with a GUI interface is preferred as the user can fill in different data entry forms and controls. The requirements collected are further necessary to integrate with the data dictionary or the high-level programming language and generate the input/output. For e.g., Accept 360, and CASE Complete tools are used for requirement analysis.

- **Structured analysis and design with CASE tools**: Structured analysis and design CASE tools are used for drawing software design diagrams with fewer efforts. Many hierarchical levels can be analyzed using CASE tools. The tools must support easy navigation and consistency during the design. For e.g., flow chart maker tools, UML diagram tools for use-case diagrams, sequence diagrams, etc. in object-oriented design.
- **Code generation and CASE tools**: CASE tools cannot be directly used to write code. The commonly used programming languages use CASE tools for database tables, interfaces, records, or the outer skeleton of the customized software to be developed. Eclipse and Cscope are used in searching code in C, Adobe Edge Inspect tool is used in web page development.
- **Test case generation using CASE tools**: CASE tools must support testing of requirements and design. The test plan generation must be supported by the CASE tools. The SoapTest is a CASE tool that allows users to validate the functions, unit testing, and functional testing. SoapTest rapidly creates regression tests.

Case tools can be divided based on the components given below. This is shown in Figure 12.6.

- Upper Case Tools - Upper CASE tools are used in requirement collection, planning, analysis, and design stages of SDLC.
- Lower Case Tools - Lower CASE tools are used in coding, testing, and maintenance phases of SDLC.
- Integrated Case Tools - Integrated CASE tools are useful in all the stages such as requirement specification, design, coding, testing, and documentation.
- Central Repository: It stores useful information required in all the stages of SDLC.

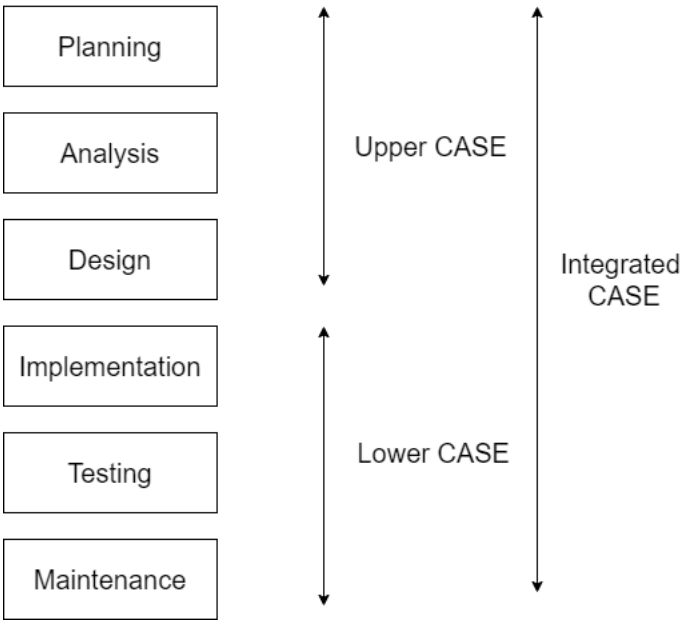

Figure 12.6 Components of CASE Tools

12.7.1 Architecture of CASE Tools

The architecture of a typical modern CASE environment consists of three components. These components are connected to the central repository as shown in Figure 12.6.

1. User interface
2. Toolset
3. Object management system (OMS)

Users can learn how to access the different tools with the help of the interface. This helps in reducing the overhead of learning efforts for a CASE tool. OMS is used to map the different entities of software such as requirements, design, code, text data, project plans to the storage management system or repository. For e.g., in the relational database management system, the relationships between entities, attribute declaration, and initialization, types of relationships, etc. can be mapped using the OMS. The repository consists of different CASE tools used in requirements, design, coding, and testing phases of a software.

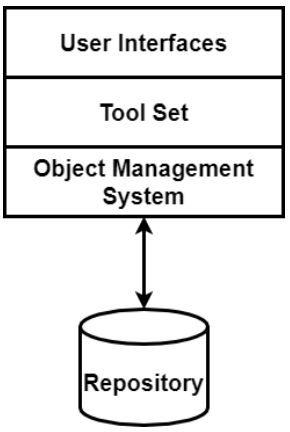

Figure 12.6 Architecture of a modern CASE environment

12.7.2 Advantages and Drawbacks of CASE Tools

CASE tools are beneficial in cost-saving at different phases of SDLC. The cost of development can be saved up to 30 to 40%. As the CASE tools are already undergone testing, the quality of work is better than any newly developed program. The CASE tools are useful in generating consistent documentation. As the central repository is used to store the related data, there are fewer chances of having the human error.

Preparing software design using CASE tools is easier than the manual or other geometric methods. The different phases of SDLC are systematic and consistent due to the usage of the CASE tools. This saves a lot of maintenance costs. The overall working of the SDLC is structured and systematic with the help of CASE tools.

Though CASE tools reduce the time of each phase of SDLC drastically, there are certain drawbacks as given below.

The staff using the CASE tools must be given training on using the tool for the respective phase of the SDLC. In many situations, staff will resist the use of the CASE tool. The size of the project may not be suitable to the CASE tool in certain cases. Also, the level of quality using CASE tools may not be desirable in certain applications.

12.8 Multiple Choice Questions (MCQs)

1. Which of the following is not a maintenance method?
    a. Adaptive maintenance
    b. Corrective maintenance
    c. Preventive maintenance
    d. Consistent maintenance

2. In the maintenance phase, the type of software testing performed is:
    a. Integration testing
    b. System testing
    c. Regression testing
    d. Acceptance testing
3. The method of generating the requirement specifications from the code is known as:
    a. Re-engineering
    b. Reverse engineering
    c. Next reengineering
    d. Forward engineering
4. Legacy system term is used for
    a. Discarded system
    b. New system
    c. Old system in use
    d. System in development
5. Legacy systems can be evolved using the following:
    a. Re-engineering
    b. Forward engineering
    c. Reverse engineering
    d. None of the above
6. CASE tools are
    a. Software Tools
    b. Hardware Tools
    c. System tools
    d. None of the mentioned above
7. CASE tools used in testing and maintenance is known as:
    a. Uppercase CASE
    b. Lowercase CASE
    c. Integrated CASE
    d. None of the above
8. A small subset of software source programs that get changed in a year is.
    a. Annual change traffic
    b. Annual Test plan

c. A test documentation
d. Advances cumulative tool

9. Modification of the characteristics of a software product over generations is:
   a. Software testing
   b. Software generation
   c. Legacy software
   d. Software evolution
10. Consider 100 KLOC is added and 30 KLOC deleted in a software of 1000 KLOC, the value of ACT is:
   a. 1.0
   b. 1.5
   c. 0.07
   d. 7.00

12.9 Review Questions

1. Discuss the necessity of software maintenance.
2. Describe the types of maintenance required for a software product with a suitable example.
3. Compare forward and reverse engineering with a suitable example.
4. What is a legacy product? How to overcome the problems related to the legacy software.
5. Discuss how the maintenance cost is calculated using ACT.
6. Define CASE tool. Explain the advantages and benefits of CASE tools.
7. Identify the features of a prototyping CASE tool.
8. Discuss how the CASE tools can be used as lowercase, uppercase and integrated components.
9. Define case environment with a suitable diagram.
10. Discuss the objectives of the CASE tools.

12.10 Keys to MCQ

1:d	2:c	3:b	4:c	5:a	6:a	7:b	8:a	9:d	10:c