Module 14

Software Maintenance

Lesson 36

Characteristics of Software Maintenance

Specific Instructional Objectives

At the end of this lesson the student would be able to:

- Explain the necessity of software maintenance.
- Identify the types of software maintenance.
- Identify the disadvantages associated with software maintenance.
- Explain what is meant by software reverse engineering.
- What are legacy software products? Identify the problems in their maintenance.
- Identify the factors upon which software maintenance activities depend.
- Identify the process models for software maintenance.
- Explain what is meant by software reengineering.
- Estimate the approximate maintenance cost of a software product.

Necessity of software maintenance

Software maintenance is becoming an important activity of a large number of software organizations. This is no surprise, given the rate of hardware obsolescence, the immortality of a software product per se, and the demand of the user community to see the existing software products run on newer platforms, run in newer environments, and/or with enhanced features. When the hardware platform is changed, and a software product performs some low-level functions, maintenance is necessary. Also, whenever the support environment of a software product changes, the software product requires rework to cope up with the newer interface. For instance, a software product may need to be maintained when the operating system changes. Thus, every software product continues to evolve after its development through maintenance efforts. Therefore it can be stated that software maintenance is needed to correct errors, enhance features, port the software to new platforms, etc.

Types of software maintenance

There are basically three types of software maintenance. These are:

- **Corrective:** Corrective maintenance of a software product is necessary to rectify the bugs observed while the system is in use.

- **Adaptive:** A software product might need maintenance when the customers need the product to run on new platforms, on new operating systems, or when they need the product to interface with new hardware or software.

- **Perfective:** A software product needs maintenance to support the new features that users want it to support, to change different functionalities of

the system according to customer demands, or to enhance the performance of the system.

Problems associated with software maintenance

Software maintenance work typically is much more expensive than what it should be and takes more time than required. In software organizations, maintenance work is mostly carried out using ad hoc techniques. The primary reason being that software maintenance is one of the most neglected areas of software engineering. Even though software maintenance is fast becoming an important area of work for many companies as the software products of yester years age, still software maintenance is mostly being carried out as fire-fighting operations, rather than through systematic and planned activities.

Software maintenance has a very poor image in industry. Therefore, an organization often cannot employ bright engineers to carry out maintenance work. Even though maintenance suffers from a poor image, the work involved is often more challenging than development work. During maintenance it is necessary to thoroughly understand someone else's work and then carry out the required modifications and extensions.

Another problem associated with maintenance work is that the majority of software products needing maintenance are legacy products.

Software reverse engineering

Software reverse engineering is the process of recovering the design and the requirements specification of a product from an analysis of its code. The purpose of reverse engineering is to facilitate maintenance work by improving the understandability of a system and to produce the necessary documents for a legacy system. Reverse engineering is becoming important, since legacy software products lack proper documentation, and are highly unstructured. Even well-designed products become legacy software as their structure degrades through a series of maintenance efforts.

The first stage of reverse engineering usually focuses on carrying out cosmetic changes to the code to improve its readability, structure, and understandability, without changing of its functionalities. A process model for reverse engineering has been shown in fig. 14.1. A program can be reformatted using any of the several available prettyprinter programs which layout the program neatly. Many legacy software products with complex control structure and unthoughtful variable names are difficult to comprehend. Assigning meaningful variable names is important because meaningful variable names are the most helpful thing in code documentation. All variables, data structures, and functions should be assigned meaningful names wherever possible. Complex

nested conditionals in the program can be replaced by simpler conditional statements or whenever appropriate by case statements.

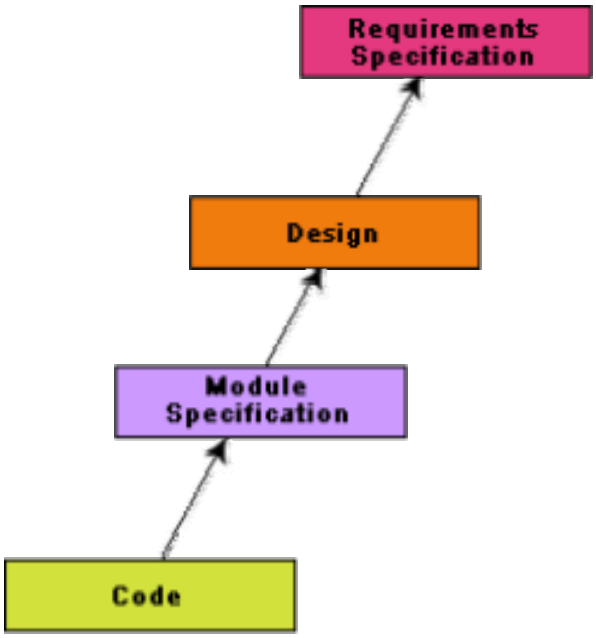

Fig. 14.1: A process model for reverse engineering

After the cosmetic changes have been carried out on a legacy software, the process of extracting the code, design, and the requirements specification can begin. These activities are schematically shown in fig. 14.2. In order to extract the design, a full understanding of the code is needed. Some automatic tools can be used to derive the data flow and control flow diagram from the code. The structure chart (module invocation sequence and data interchange among modules) should also be extracted. The SRS document can be written once the full code has been thoroughly understood and the design extracted.

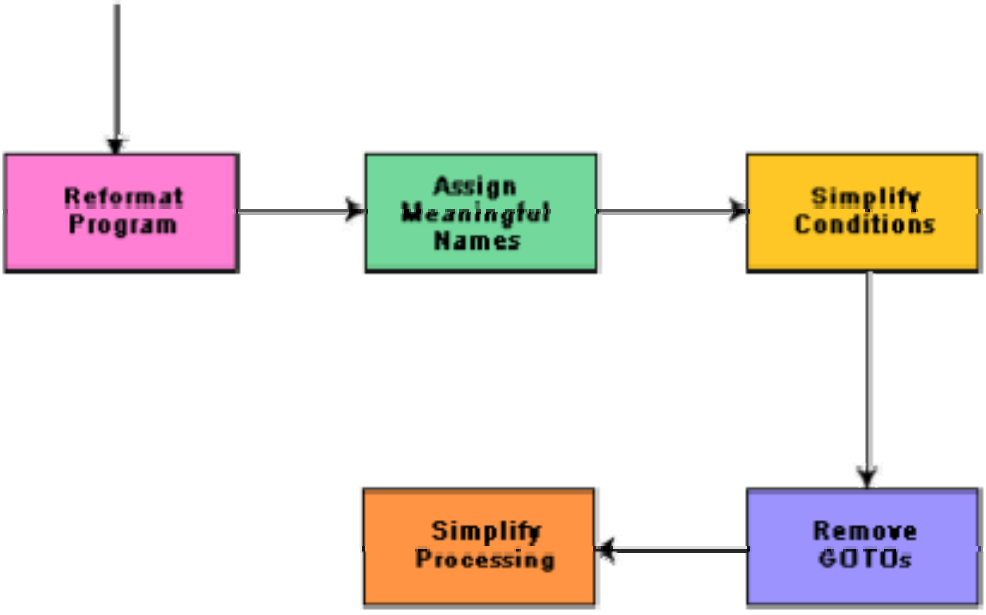

Fig. 14.2: Cosmetic changes carried out before reverse engineering

Legacy software products

It is prudent to define a legacy system as any software system that is hard to maintain. The typical problems associated with legacy systems are poor documentation, unstructured (spaghetti code with ugly control structure), and lack of personnel knowledgeable in the product. Many of the legacy systems were developed long time back. But, it is possible that a recently developed system having poor design and documentation can be considered to be a legacy system.

Factors on which software maintenance activities depend

The activities involved in a software maintenance project are not unique and depend on several factors such as:

- the extent of modification to the product required
- the resources available to the maintenance team
- the conditions of the existing product (e.g., how structured it is, how well documented it is, etc.)
- the expected project risks, etc.

When the changes needed to a software product are minor and straightforward, the code can be directly modified and the changes appropriately reflected in all

the documents. But more elaborate activities are required when the required changes are not so trivial. Usually, for complex maintenance projects for legacy systems, the software process can be represented by a reverse engineering cycle followed by a forward engineering cycle with an emphasis on as much reuse as possible from the existing code and other documents.

Software maintenance process models

Two broad categories of process models for software maintenance can be proposed. The first model is preferred for projects involving small reworks where the code is changed directly and the changes are reflected in the relevant documents later. This maintenance process is graphically presented in fig. 14.3. In this approach, the project starts by gathering the requirements for changes. The requirements are next analyzed to formulate the strategies to be adopted for code change. At this stage, the association of at least a few members of the original development team goes a long way in reducing the cycle team, especially for projects involving unstructured and inadequately documented code. The availability of a working old system to the maintenance engineers at the maintenance site greatly facilitates the task of the maintenance team as they get a good insight into the working of the old system and also can compare the working of their modified system with the old system. Also, debugging of the reengineered system becomes easier as the program traces of both the systems can be compared to localize the bugs.

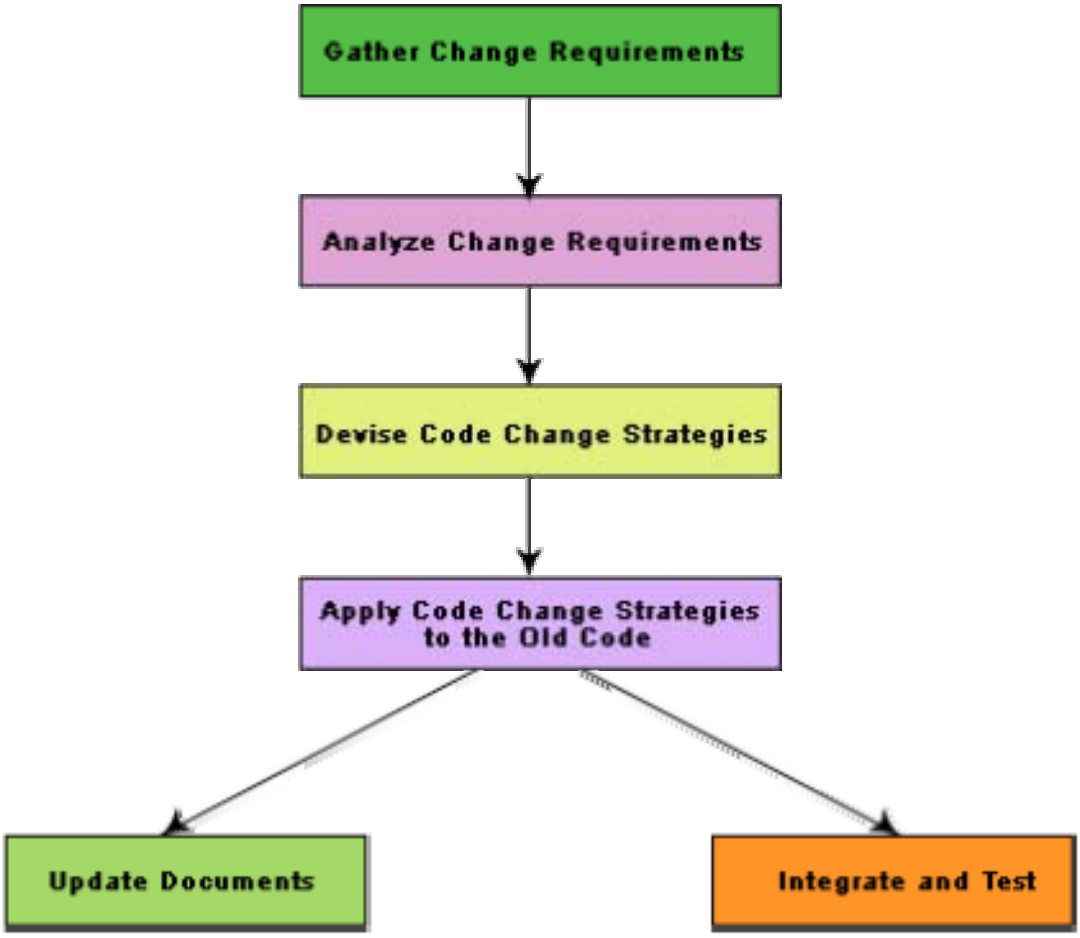

Fig. 14.3: Maintenance process model 1

The second process model for software maintenance is preferred for projects where the amount of rework required is significant. This approach can be represented by a reverse engineering cycle followed by a forward engineering cycle. Such an approach is also known as software reengineering. This process model is depicted in fig. 14.4. The reverse engineering cycle is required for legacy products. During the reverse engineering, the old code is analyzed (abstracted) to extract the module specifications. The module specifications are then analyzed to produce the design. The design is analyzed (abstracted) to produce the original requirements specification. The change requests are then applied to this requirements specification to arrive at the new requirements specification. At the design, module specification, and coding a substantial reuse is made from the reverse engineered products. An important advantage of this approach is that it produces a more structured design compared to what the original product had, produces good documentation, and very often results in increased efficiency. The efficiency improvements are brought about by a more efficient design. However, this approach is more costly than the first approach. An empirical study indicates that process 1 is preferable when the amount of

rework is no more than 15% (as shown in fig. 14.5). Besides the amount of rework, several other factors might affect the decision regarding using process model 1 over process model 2:

- Reengineering might be preferable for products which exhibit a high failure rate.
- Reengineering might also be preferable for legacy products having poor design and code structure.

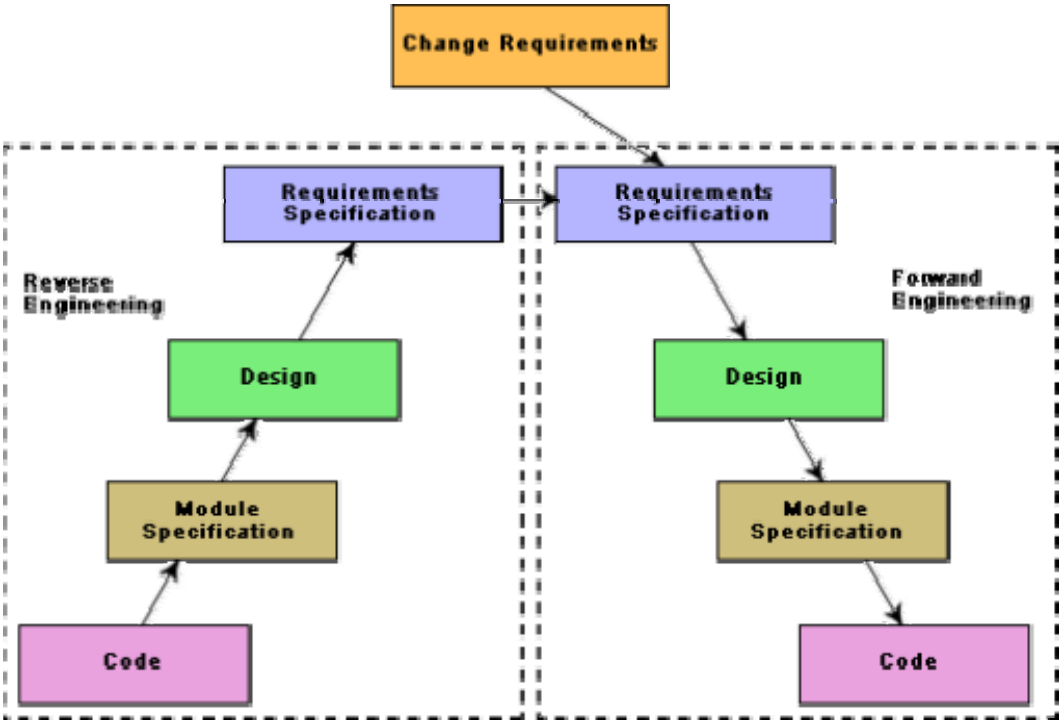

Fig. 14.4: Maintenance process model 2

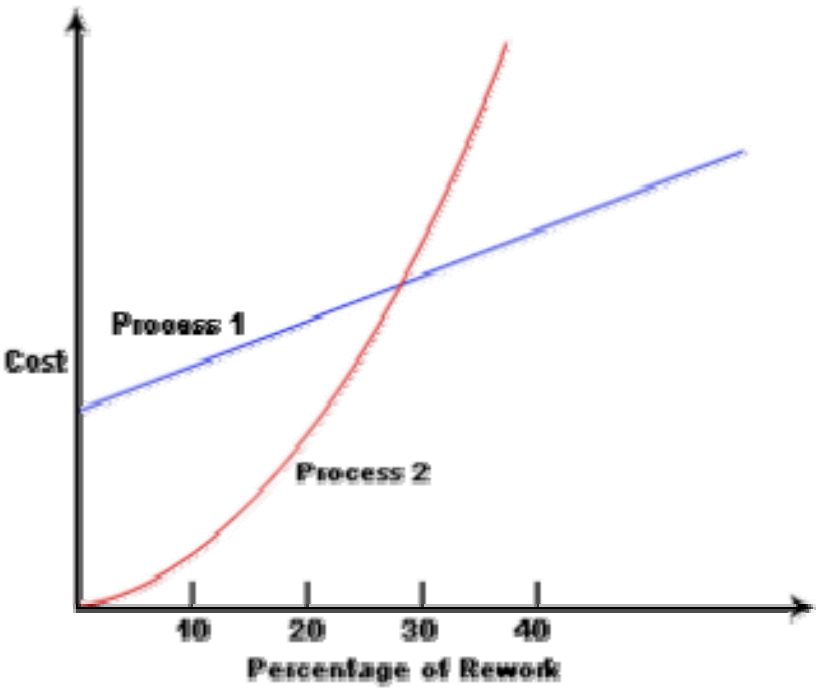

Fig. 14.5: Empirical estimation of maintenance cost versus percentage rework

Software reengineering

Software reengineering is a combination of two consecutive processes i.e. software reverse engineering and software forward engineering as shown in the fig. 14.4.

Estimation of approximate maintenance cost

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

Boehm [1981] proposed a formula for estimating maintenance costs as part of his COCOMO cost estimation model. Boehm's maintenance cost estimation is made in terms of a quantity called the Annual Change Traffic (ACT). Boehm defined ACT as the fraction of a software product's source instructions which undergo change during a typical year either through addition or deletion.

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

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

$$\text{maintenance cost} = ACT \times \text{development cost.}$$

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

The following questions have been designed to test the objectives identified for this module:

1. What for software products are required to maintain?

2. What are the different types of maintenance that a software product might need? Why are these maintenance required?

3. What are the disadvantages associated with software maintenance?

4. What do you mean by the term software reverse engineering? Why is it required? Explain the different activities undertaken during reverse engineering.

5. What is legacy software product? Explain the problems one would encounter while maintaining a legacy product.

6. What are the different factors upon which software maintenance activities depend?

7. What do you mean by the term software reengineering? Why is it required?

8. If the development cost of a software product is Rs. 10,000,000/-, compute the annual maintenance cost given that every year approximately 5% of the code needs modification. Identify the factors which render the maintenance cost estimation inaccurate.

Mark all options which are true.

1. Software products need maintenance to

□ correct errors
□ enhance features
□ port to new platforms
□ overcome wear and tear caused by use

2. Software products need adaptive maintenance for which of the following reasons?

□ to rectify bugs observed while the system is in use
□ when the customers need the product to run on new platforms.
□ to support the new features that users want it to support.
□ all of the above

3. Hardware products need maintenance to

□ correct errors
□ enhance features
□ port to new platforms
□ overcome wear and tear caused by use

4. Legacy software products having poor design and code structure are maintained by performing which task?

□ the code can be directly modified and the changes appropriately reflected in all the relevant documents
□ suitable software maintenance process must be followed by a reverse engineering cycle followed by a forward engineering cycle with an emphasis on as much reuse as possible from the existing code and other documents
□ none of the above

5. A reverse engineering cycle during maintenance phase is required for which type of software products?

□ well documented software products
□ well structured software products
□ legacy software products
□ both well documented and well structured software products

6. Reengineering is preferable for which of the software products?

□ software products exhibiting high failure rates
□ software products having poor design
□ software products having poor code structure
□ all of the above

7. Identify which of the following factors software maintenance cost estimation models should take into account.

□ experience level of the engineers
□ familiarity of the engineers with the product
□ hardware requirements
□ software complexity
□ all of the above

8. Software maintenance effort requires approximately what percentage of the total life cycle cost for a typical software product?

□ about 90%
□ about 70%
□ about 60%
□ about 40%

Mark the following statements as either True or False. Justify your answer.

1. Corrective maintenance is the type of maintenance that is frequently carried out on average software product.

2. Only badly designed software products need maintenance.

3. The structure of a program may be degraded as more and more maintenance is carried out.

4. Legacy software products are very difficult to maintain.

5. Legacy products are those products which have been developed long time back.

6. In the process of reverse engineering, we change the functionalities of an existing code.