

Unit-6

Software Maintenance: Software maintenance, Maintenance Process Models, Maintenance Cost, Software Configuration Management.

Software Reuse: what can be Reused? Why almost No Reuse So Far? Basic Issues in Reuse Approach, Reuse at Organization Level.

Software Maintenance

Software maintenance:

Introduction
Types of software maintenance
Characteristics of software evolution

Introduction:

Software maintenance is becoming an important activity of a large number of software organizations. 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:

- **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

Characteristics of software evolution:

Continuing change - A software system must continue to adapt to the real world changes, else it becomes progressively less useful.

Increasing complexity - A software system evolves, its complexity tends to increase unless work is done to maintain or reduce it.

Continuing growth- In order for a system intended to resolve some business problem, its size of implementing the changes grows according to the lifestyle changes of the business.

Maintenance Process Models:

Maintenance Process Model 1
Maintenance Process Model 2

Maintenance Process Model 1:

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 time, 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.

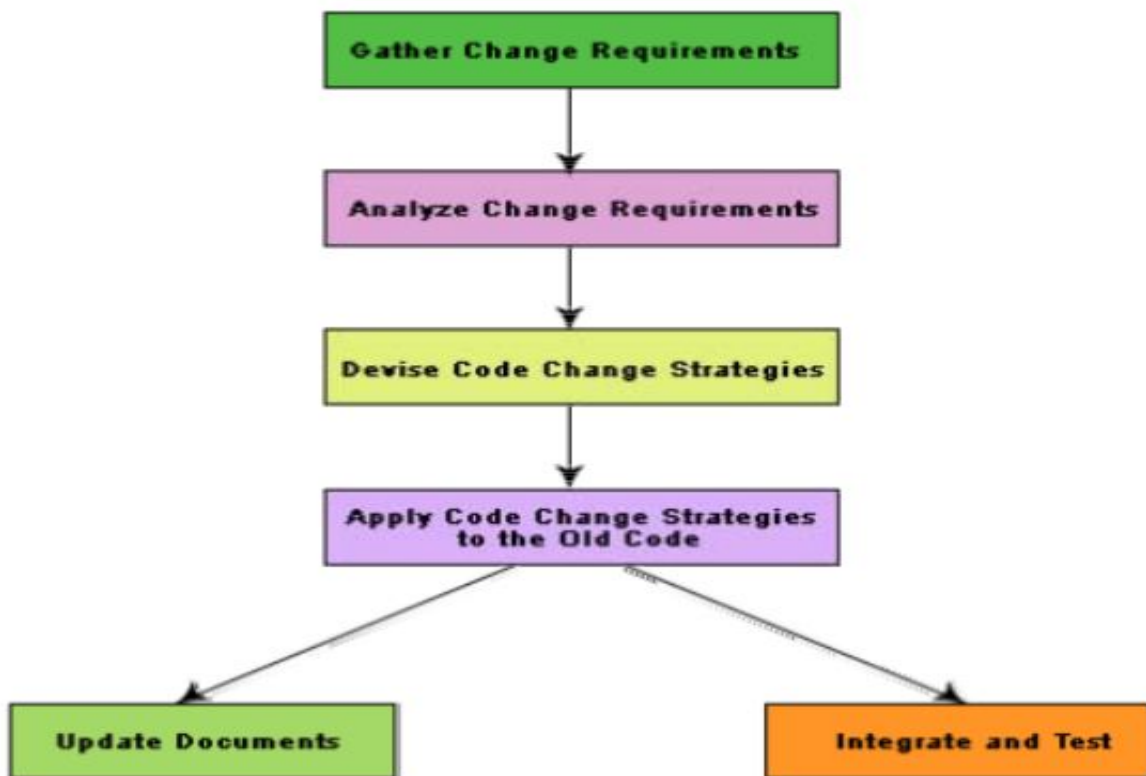


Fig. 14.3: Maintenance process model 1

Maintenance Process Model 2:

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.

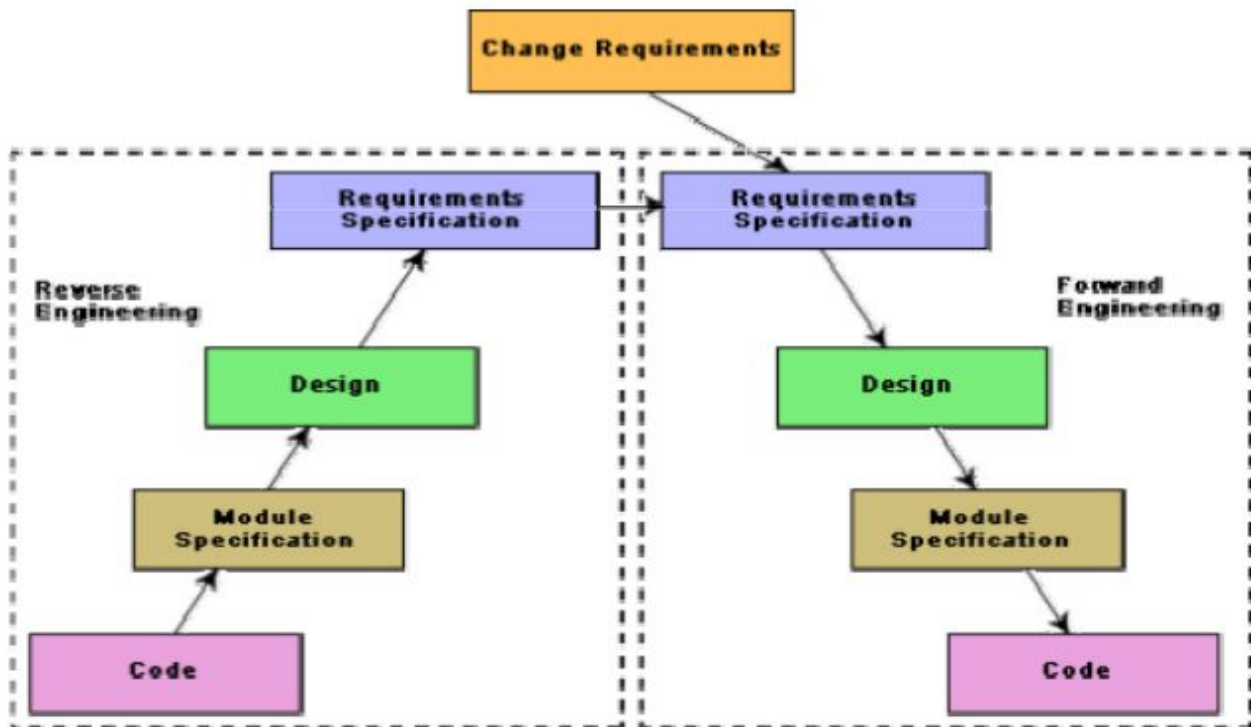


Fig. 14.4: Maintenance process model 2

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 added maintenance. $KLOC_{deleted}$ is the total KLOC deleted during maintenance. deleted. 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:

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

Software Configuration Management:

- The results (also called as the deliverables) of a large software development effort typically consist of a large number of objects, e.g. source code, design document, SRS document, test document, user's manual, etc.
- These objects are usually referred to and modified by a number of software engineers through out the life cycle of the software.
- The state of all these objects at any point of time is called the configuration of the software product.
- The state of each deliverable object changes as development progresses and also as bugs are detected and fixed.

Software Reuse

what can be Reused?

Introduction
Artifacts that can be reused

Introduction:

software products are expensive. Software project managers are worried about the high cost of software development and are desperately look for ways to cut development cost. A possible way to reduce development cost is to reuse parts from previously developed software. In addition to reduced development cost and time, reuse also leads to higher quality of the developed products since the reusable components are ensured to have high quality.

Artifacts that can be reused:

It is important to know about the kinds of the artifacts associated with software development that can be reused. Almost all artifacts associated with software development, including project plan and test plan can be reused. However, the prominent items that can be effectively reused are:

- Requirements specification
- Design
- Code
- Test cases
- Knowledge

Why almost No Reuse So Far?

Engineers working in software development organizations often have a feeling that the current system they are developing is similar to the last few systems. However, no attention is paid on how to duplicate what can be reused from previously developed systems. Everything is being built from the scratch. The current system falls behind schedule and no one has time to figure out how the similarity between the current systems developed in the past can be exploited,

Basic Issues in Reuse Approach:

The following are some of the basic issues that must be clearly understood for starting any reuse program.

- Component creation
- Component indexing and storing
- Component search
- Component understanding
- Component adaptation
- Repository maintenance

Component creation: For component creation, the reusable components have to be first identified. Selection of the right kind of components having potential for reuse is important. Domain analysis is a promising technique which can be used to create reusable components.

Component indexing and storing: Indexing requires classification of the reusable components so that they can be easily searched when looking for a component for reuse. The components need to be stored in a Relational Database Management System (RDBMS) or an Object-Oriented Database System (ODBMS) for efficient access when the number of components becomes large.

Component searching: The programmers need to search for right components matching their requirements in a database of components. To be able to search components efficiently, the programmers require a proper method to describe the components that they are looking for.

Component understanding: The programmers need a precise and sufficiently complete understanding of what the component does to be able to decide whether they can reuse the component. To facilitate understanding, the components should be well documented and should do something simple.

Component adaptation: Often, the components may need adaptation before they can be reused, since a selected component may not exactly fit the problem at hand. However, tinkering with the code is also not a satisfactory solution because this is very likely to be a source of bugs.

Repository maintenance: A component repository once is created requires continuous maintenance. New components, as and when created have to be entered into the repository. The faulty components have to be tracked. Further, when new applications emerge, the older applications become obsolete. In this case, the obsolete components might have to be removed from the repository.

Reuse at Organization Level:

Achieving organization-level reuse requires adoption of the following steps:

- Assessing a product's potential for reuse

- Refining products for greater reusability
- Entering the product in the reuse repository

Assessing a product's potential for reuse:

Assessment of components reuse potential can be obtained from an analysis of a questionnaire circulated among the developers. The questionnaire can be devised to access a component's reusability. The programmers working in similar application domain can be used to answer the questionnaire about the product's reusability. Depending on the answers given by the programmers, either the component be taken up for reuse as it is, it is modified and refined before it is entered into the reuse repository, or it is ignored. A sample questionnaire to assess a component's reusability is the following.

- Is the component's functionality required for implementation of systems in the future?
- How common is the component's function within its domain?
- Is the component hardware dependent?
- If the component is non-reusable, then can it be decomposed to yield some reusable components?
- Can we parameterize a non-reusable component so that it becomes reusable?

Refining products for greater reusability: For a product to be reusable, it must be relatively easy to adapt it to different contexts. Machine dependency must be abstracted out or localized using data encapsulation techniques. The following refinements may be carried out:

- **Name generalization:** The names should be general, rather than being directly related to a specific application.
- **Operation generalization:** Operations should be added to make the component more general. Also, operations that are too specific to an application can be removed.
- **Exception generalization:** This involves checking each component to see which exceptions it might generate. For a general component, several types of exceptions might have to be handled.
- **Handling portability problems:** Programs typically make some assumption regarding the representation of information in the underlying machine. These assumptions are in general not true for all machines. The programs also often need to call some operating system functionality and these calls may not be same on all machines. Also, programs use some function libraries, which may not be available on all host machines.