Lecture 2

Long Question

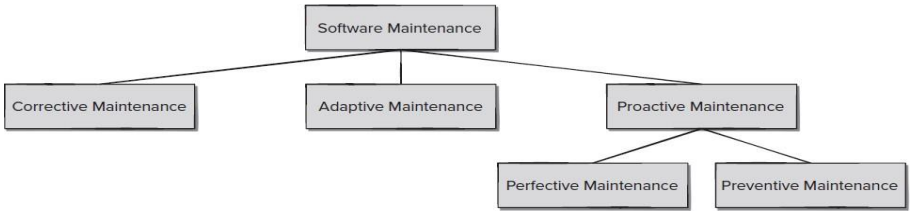
1. Software Maintenance

- **Software maintenance** is becoming an important activity of a large number of organizations.
- Software product requires rework to cope up with the newer interface, maintenance is necessary.
    - when the hardware platform changes,
    - software product perform some low-level functions,
    - whenever the support environment of a software product changes, and
    - when the operating system changes,
- The growing queues of bug fixes, adaptation requests, and outright enhancements must be planned, scheduled, and ultimately accomplished.
- Thus, every software product continues to evolve after its development through maintenance efforts.
- Then the challenge of software maintenance has begun.

Types of Software Maintenance

- Four types of software maintenance
    1. Corrective Maintenance
        - necessary either to rectify the bugs, fault repairs to fix bugs and vulnerabilities observed while the system is in use
    2. Adaptive Maintenance
        - It is an environmental adaptation to adapt the software to new platforms and environments
    3. Perfective Maintenance
        - It is functionality addition to add new features and to support new requirements.
    4. Preventive Maintenance

- modifications and updating to prevent future problems of the software and
- to prevent the deterioration of software as it continues to adapt and change



2. Maintenance Process Model – 2

.The **second model** is preferred for **projects** where

.the amount of rework required is significant.

.This approach uses

. reverse engineering cycle followed by a forward engineering cycle.

.Also called, **Software Re-engineering**.

.An important advantage of this approach is that

.produces a more structured design compared to what the original product had,

.produces good documentation, and

.very often results in increased efficiency that provides a more efficient design.

(1) **The Reverse Engineering Cycle**

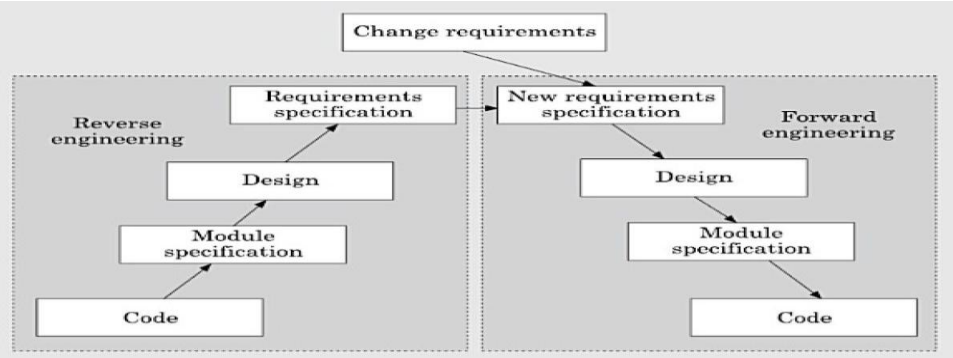
- for legacy products, the old code is analyzed to **extract** the module specifications.
- The module specifications are then analyzed to

produce the design.

- The design is analyzed to **produce** the original requirements specification.

(1) **The forward engineering cycle**

- The change requests are then applied to this requirements specification to arrive at the new requirements specification.
- At this point a forward engineering is carried out to produce the new code.
- At the design, module specification, and coding a substantial reuse is made from the reverse engineered products.



3. Challenges in Software Maintenance

- **Software maintenance** work currently is typically
  - much **more expensive** than what it should be and
  - takes **more time than required**.
- The reasons for this situation are the following:

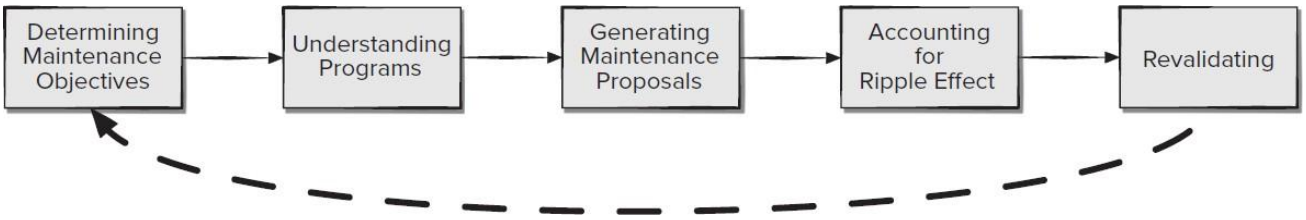
1. Software maintenance work in organizations is mostly carried out using ad-hoc techniques.
2. Software maintenance has a very poor image in industry.

- During maintenance it is necessary to **thoroughly understand someone else's work, and then carry out the required modifications and extensions**
- 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**.

3. Developers may not want to spend time on risk assessment or planning.
    - the ***possibility that the changes required to fix one problem will cause new problems*** to other portions of the program.

Software Maintenance Tasks



- Figure 3.4 shows a set of generic tasks of a controlled software maintenance process.
- **The first task** is to understand the system that needs to be modified
- **It is essential to** test the modified software carefully to ensure software changes have their intended effect and do not break the software in other places.

Short Notes

1. A Unified Theory for Software Evolution

1. Law of Continuing Change (1974).
2. Law of Increasing Complexity (1974).
3. Law of Conservation of Familiarity (1980).
4. Law of Continuing Growth (1980).
5. Law of Declining Quality (1996).

2. LEGACY SYSTEMS

•These older software systems are called legacy systems.

•Legacy systems are rely on languages and technology that are no longer used for new systems development.
•Legacy software may be dependent on older hardware, such as mainframe computers
•and may have associated legacy processes and procedures.
•It may be impossible to change to more effective business processes because the legacy software cannot be modified to support new processes.
•Legacy software systems that are expensive to change.
•The costs of managing and maintaining the legacy system become so high that it has to be replaced with a new system.

3. **Reverse Engineering Process**
    - **Reverse engineering** requires developers to evaluate the old software system and developing a meaningful specification to
        - understand **source code** by examining its (often undocumented) source code,
        - understand the **processing** being performed,
        - understand the **user interface** that was used,
        - understand the **data** and **internal data structures,**
        - understand associated **database structure**.
        - **Two stages** in software reverse Engineering Process

    1) Cosmetic changes to the program code (Code Refactoring)
    2) Extracting the code, design, and the requirements specification

4. **Refactoring**

    Refactoring (also known as restructuring)

    •is the process of making improvements to a program to slow down degradation through change
    •does not modify the overall program architecture.
    •focus on the design details of individual modules and on local data structures
    •should not add functionality

•should concentrate on program improvement
•refactoring as “preventative maintenance”
•design refactoring
•involves identifying relevant design patterns and replacing existing code with code that implements these design patterns.

5.Code Refactoring

•Code refactoring is a performed design that produces the same function but with higher quality than the original program.

- identify bad code design practices and
- suggest possible solutions
- to reduce coupling and improve cohesion.

•Although code refactoring can improve immediate problems associated with debugging or small changes, it is not reengineering.

6.**Architecture Refactoring**

- architectural or design changes to a software product already in production can be a costly and time-consuming process.
- consider architectural refactoring (design refactoring) as one of the design trade-offs.

. In general, for a messy program like this you have the following options:

1. You can struggle through modification after modification
2. You can attempt to understand the broader inner workings of the program
3. You can revise (redesign, recode, and test)
4. You Can completely redo (redesign, recode, and test) the complete program

4.You can completely redo (redesign, recode, and test) the complete program