Chapter 3

Fundamentals of Software Maintenance

Overview

- Categories of Software Maintenance
  - Corrective (Design error, Logical error, coding error)
  - Preventive (code restructuring, refactoring etc)
  - Perfective (new functionality)
  - Adaptive (hardware software changes)
- Relationship between Maintenance Categories
- Ongoing support
  - Effective Communication, Training End User, and Provide business information
- Lehman’s Laws
  - Continuing Change
  - Increasing Complexity
  - Self Regulation
  - Conservation of Organizational Stability
  - Continuing growth
  - Declining Quality
  - Feedback system

Categories of Software Maintenance

There are four types of changes in software maintenance.

1. Corrective
2. Adaptive
3. Perfective
4. Preventive

(CAPP)

Corrective Maintenance

Def: Modification initiated by defects are known as corrective maintenance of software.

A defect can result from following errors:

1. Design errors: These occur when, for example, changes made to the software are incorrect, incomplete, wrongly communicated or the change request is misunderstood.
2. Logic errors: These result from invalid tests and conclusions, incorrect implementation of design specifications, faulty logic flow or incomplete testing of data.
3. Coding errors: These are caused by incorrect implementation of detailed logic design and incorrect use of the source code logic. Defects are also caused by data processing errors and system performance errors.

All these errors, sometimes called 'residual errors' or 'bugs', prevent the software from conforming to its agreed specification.

Corrective Maintenance

Software Patch: A software patch or fix is a quick-repair job for a piece of programming designed to resolve functionality issues, improve security and add new features.

This approach gives rise to a range of problems that include increased program complexity and unforeseen ripple effects. It thus leads to spaghetti syndrome.

Spaghetti syndrome: Degeneration of program structure which makes the program increasingly difficult to understand. It also indicates the resistance to change of the program at its maximum

Adaptive Maintenance

Def: Adaptive Maintenance is a change driven by the need to accommodate modifications in the environment of the software system.

Environment in this context refers to the totality of all conditions and influences which act from outside upon the system.

For example business rules, government policies, work patterns, software and hardware operating platforms.

Adaptive Maintenance

Adaptive maintenance includes any work initiated as a consequence of moving the software to a different hardware or software platform - compiler, operating system or new processor.

For example a system was designed to work on Win10, It is showing errors in Win11. Now it has to be compatible with Win11 as well.

When Euro became the official currency of European Countries, systems like accounting programs, banking apps, billing systems, and shopping websites had to do adaptive changes.

Perfective Maintenance

Def: Maintenance undertaken to expand the existing requirements of a system.

With existing enhancement of existing system functionality or improvement in computational efficiency and new cases beyond the scope for which it was initially developed

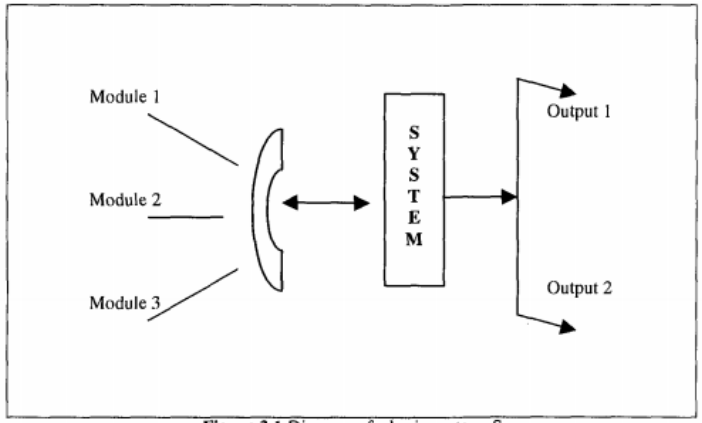

Figure 3.1 Diagram of a basic system, S

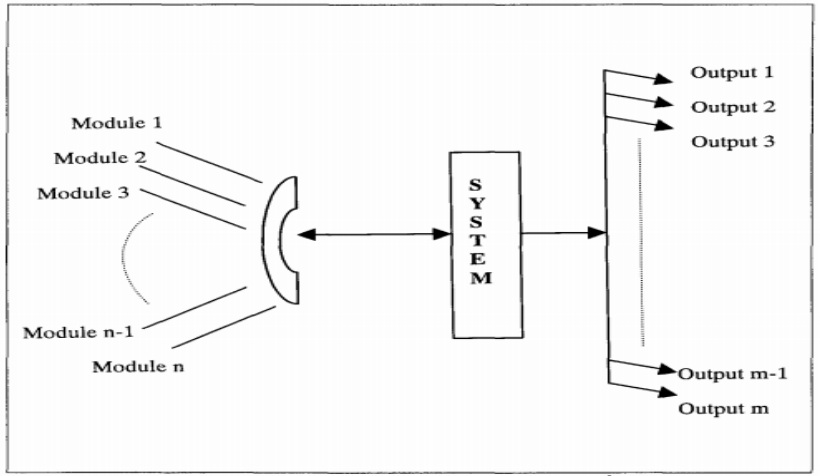

Figure 3.2 Diagram of an enhanced system, S'

Preventive Maintenance

Def: Work done on a software system to address problems of deteriorating structure is known as preventive change.

Preventive change is undertaken to prevent malfunctions or to improve maintainability of the software

Examples of preventive change include code restructuring, code optimisation and documentation updating.

Relationship Between Maintenance Categories

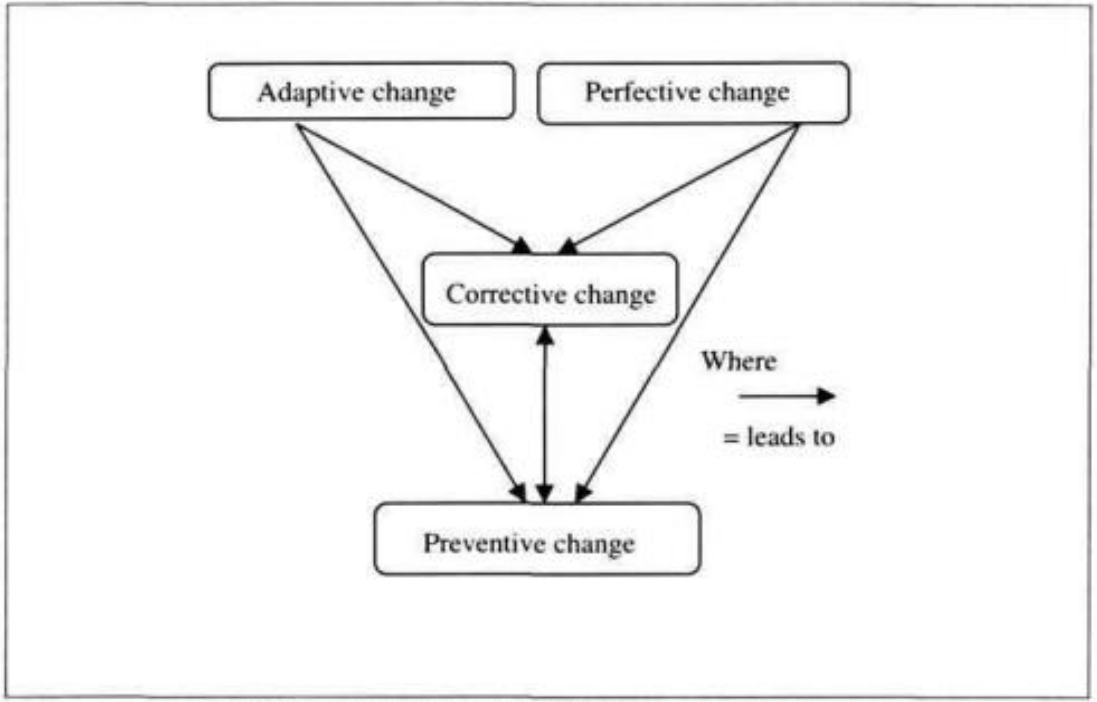

Figure 3.3 Potential relation between software changes

Ongoing Support

- **Def**: This category of maintenance work refers to the service provided to satisfy non-programming-related work requests.
  It does not create software change, but is essential for successful communication of desired changes.

- Followings are some of the ongoing support activities:
  - **Effective communication:** Maintenance is the most customer-intensive part of the software life-cycle. It is necessary to establish good customer relations and cooperation, and continually demonstrate that they aim to satisfy the customers
  - **Training of end-users**: Training refers to the process of equipping users with sufficient knowledge and skills to enable them use a system to its full potential.
  - **Providing business information:** Users need various types of timely and accurate information to enable them take strategic business decisions.

Change Management

- Change management in software projects is the process of transiting from the current defective state to the improved state.

Reasons to Change

- The project requirements changed.
- Need bug fixing.
- Some team members left the project.
- Your company has been reorganized.
- Market demands have shifted.
- Project performance requires some improvements.

Benefits of Change Management

Types of Change Management

- **Anticipatory change:** This takes place when we know in advance that a certain change or series of changes is bound to happen. Such planned shifts are significantly easier to implement as here; the project manager has time to tackle the expected situation.
- **Incremental change:** The changes in projects that happen relatively often and gradually. They don't involve immense shifts that turn the entire project on its head. Instead, the changes are introduced progressively and often may not be noticeable at first glance.
- **Emergency (or urgent) change:** The changes that need to be introduced immediately. Otherwise, the project may become a failure or its execution may be impossible.
- **Reactive change:** Shifts that occur due to an event or a series of events. They often happen when least expected. For that reason, reactive changes are particularly challenging to manage as, in most cases, they can't be planned in advance.
- **Strategic change**: They involve the whole organization and result from the decisions of C-level management.

Tools for Conducting Change Management

- Wrike
- The Change Compass
- JIRA
- Trello
- The Change Shop

Change Management Steps

Reference

https://www.miquido.com/blog/change-management-in-software-development/

Lehman's Laws

Lehman's eight Laws of Software Evolution

1. Law of continuing change: Systems must be continually adapted or they become progressively less satisfactory to use.
2. Law of increasing complexity: As a system evolves, its complexity increases unless work is done to maintain or reduce it.
3. Self-Regulation: Evolutionary aspects of system evolution processes tend to display a degree of statistical regularity
4. Law of conservation of organisational stability: Average work rate in an real world process tends to remain constant over periods of system evolution.

Lehman's Laws

Lehman's eight Laws of Software Evolution

5. Law of conservation of familiarity: The average incremental growth of systems tends to remain constant or decline.
6. Law of continuing growth: Functional capability must be continually increased over a system's lifetime to maintain user satisfaction.
7. Law of declining quality: Unless rigorously adapted to meet changes in the operational environment, system quality will appear to decline.
8. Law of feedback systems: Evolution processes are multi-level, multi-loop, multi-agent feedback systems.