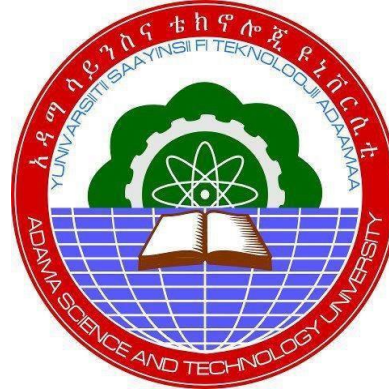




Chapter -1

Fundamentals of software Evolution and Maintenance

SEng4203

Lecture notes

ASTU, 2023

Begna D.

ibmbegna@gmail.com

Chapter outlines

- Definition and methodology
- Nature of Maintenance
- Need for Maintenance
- Maintenance Cost
- Evolution of Software
- Categories of Software

Software Evolution and maintenance concepts

- ◆ The concept of **software maintenance** means preventing software from failing to deliver the intended functionalities by means of bug fixing.
- ◆ The concept of **software evolution** means a continual change from a lesser, simpler, or worse state to a higher or better state.

Bennett and Xu made further distinctions between the two as follows:

- ◆ All support activities carried out *after* delivery of software are put under the category of *maintenance*.
- ◆ All activities carried out to effect changes in requirements are put under the category of *evolution*.

...

Software maintenance comprises all activities associated with the process of changing software for the purposes of:

- ◆ fixing bugs; and/or
- ◆ improving the design of the system so that future changes to the system are less expensive.

Evolution of software systems means creating new but related designs from existing ones. The objectives include:

- ◆ supporting new functionalities
- ◆ making the system perform better and
- ◆ making the system run on a different operating system

Software Evolution

- ◆ Fundamental work in the field of software evolution was done by Lehman and his collaborators.
 - ◆ Based on empirical studies, Lehman and his collaborators formulated some observations and they introduced them as *laws of evolution*.
 - ◆ The “laws” themselves have “evolved” from *three* in 1974 to *eight* by 1997.
- ◆ Those laws are the results of studies of the evolution of large-scale proprietary or closed source software (CSS) systems.
 - ◆ The laws concern a category of software systems called *E-type* systems. The eight laws are briefly explained as follows:

Software Evolution...

1. **Continuing change:** Unless a system is continually modified to satisfy emerging needs of users, the system becomes increasingly less useful.
2. **Increasing complexity:** Unless additional work is done to explicitly reduce the complexity of a system, the system will become increasingly more complex due to maintenance-related changes.
3. **Self-regulation:** The evolution process is self-regulating in the sense that the measures of products and processes, that are produced during the evolution, follow close to *normal* distributions.

Software Evolution...

4. **Conservation of organizational stability:** The average effective global activity rate on an evolving system is almost constant throughout the lifetime of the system. In other words, the average amount of additional effort needed to produce a new release is almost the same.
5. **Conservation of familiarity:** As a system evolves all kinds of personnel, namely, developers and users, for example, must gain a desired level of understanding of the system's content and behavior to realize satisfactory evolution.

Software Evolution...

6. **Continuing growth:** As time passes, the functional content of a system is continually increased to satisfy user needs.
7. **Declining quality:** Unless the design of a system is diligently fine-tuned and adapted to new operational environments, the system's qualities will be perceived as declining over the lifetime of the system.
8. **Feedback system:** The system's evolution process involves multi-loop, multi agent, multi-level feedback among different kinds of activities.
 - ◆ Developers must recognize those complex interactions in order to continually evolve an existing system to deliver more functionalities and higher levels of qualities.

Nature of Software Maintenance....

Here are some key aspects that define the nature of software maintenance:

1. Corrective Maintenance:

- ◆ It involves identifying and fixing software defects or errors that are discovered after the system is deployed.
- ◆ It aims to restore the software to its intended functionality and address any issues reported by users or detected through testing.

2. Adaptive Maintenance:

- ◆ Adaptive maintenance focuses on modifying the software to accommodate changes in the environment.
- ◆ This may include adapting the software to work with new hardware or software platforms, operating systems, databases, or other external dependencies.

Nature of Software Maintenance....

3.Perfective Maintenance:

- Perfective maintenance involves improving the software's functionality, performance, or maintainability without altering its external behavior.
- It aims to enhance the software based on evolving user needs, industry standards, or technological advancements.

4.Preventive Maintenance:

- Preventive maintenance aims to reduce the occurrence of **future defects** by proactively identifying and addressing potential issues.
- It involves activities such as code refactoring, performance optimization, and security enhancements to mitigate risks and improve the overall quality of the software.

Nature of Software Maintenance....

5. Documentation and Knowledge Management:

- Maintaining up-to-date documentation and knowledge about the software system is crucial for effective maintenance.
- This includes documenting the system's architecture, design, configuration, and any changes made during maintenance activities.
- Proper knowledge management ensures that maintenance tasks can be performed efficiently and transferred to new team members when necessary.

6. Impact Analysis:

- Before making changes to the software, an impact analysis is conducted to assess the potential effects of modifications on the system and its interconnected components.
- This analysis helps identify potential risks, dependencies, and areas that may require additional testing or validation.

Nature of Software Maintenance....

7. Regression Testing:

- Regression testing is performed **after making modifications** or enhancements to the software to ensure that the changes **do not introduce new defects** or break existing functionality.

8. Change Management:

- Change management processes and practices are essential for controlling and managing changes to software systems.
 - It includes activities such as change request management, change prioritization, change approval, and tracking changes to ensure proper documentation and coordination among stakeholders.

Nature of Software Maintenance....

9.Resource and Time Constraints:

- Maintenance teams must plan and prioritize tasks effectively, considering factors such as available resources, budget, and the urgency of issues to be addressed.

10.Continuous Improvement:

- Software maintenance is an ongoing process aimed at continuously improving the software system.
- Lessons learned from maintenance activities, user feedback, and industry trends should be incorporated to enhance the software's quality, performance, and user satisfaction over time.

Need for Maintenance

Reasons of software maintenance:

- **Bug Fixing and Issue Resolution:** for the identification and correction of software defects, bugs, and issues that may arise during the software's operational life.
- **Enhanced Performance and Optimization:** Through maintenance activities, software can be optimized to improve its performance, efficiency, and responsiveness.
- **Adaptation to Changing Requirements:** Software maintenance enables the adaptation of software to changing user needs, business requirements, and technological advancements.

Need for Maintenance...

- **Security Updates and Vulnerability Mitigation:** Maintenance is crucial for addressing security vulnerabilities that may be discovered in the software.
- **Compatibility and Integration:** As software ecosystems evolve, maintenance ensures that the software remains compatible with new hardware platforms, operating systems, libraries, and other software components.
- **Regulatory and Compliance :** Maintenance helps address changes in regulatory requirements, industry standards, and legal obligations.

Need for Maintenance...

User Support and Customer Satisfaction: Software maintenance allows for ongoing user support, addressing user queries, providing assistance, and resolving issues promptly.

Long-Term Cost Savings: By investing in software maintenance, organizations can avoid costly system failures, downtime, and major rework.

Knowledge Preservation: Maintenance activities contribute to preserving and enhancing the knowledge about the software system.

Documentation, code reviews, and knowledge transfer ensure that the understanding of the software's design, functionality, and dependencies is maintained over time.

Need for Maintenance...

Continuous Improvement and Innovation: Feedback from users and stakeholders, as well as insights gained from maintenance activities, can be used to identify areas for enhancement and drive innovation within the software.

Maintenance Cost

Software maintenance can incur various costs throughout its lifecycle.

Here are some common costs associated with software maintenance:

Personnel Costs: These costs include salaries, benefits, and training for the maintenance team responsible for managing and maintaining the software.

It involves hiring skilled professionals, providing ongoing training to keep up with evolving technologies, and retaining experienced staff.

Maintenance Cost

Bug Fixing and Issue Resolution: The cost of identifying, analyzing, and fixing software defects and issues can be significant.

Enhancements and Updates: Costs are associated with introducing new features, functionalities, or improvements to the software based on changing user requirements or market demands.

Maintenance Cost...

Testing and Quality Assurance: Software maintenance requires rigorous testing to validate changes, ensure backward compatibility, and verify that the software continues to function as intended.

Documentation and Knowledge Management: Maintaining up-to-date documentation, including system architecture, design, and user manuals, incurs costs.

Hardware and Software Upgrades: As technology evolves, maintenance may require upgrading or replacing hardware components, operating systems, databases, or other software dependencies.

Maintenance Cost...

Security Measures: It includes implementing security patches, conducting security audits, performing vulnerability assessments, and maintaining robust security measures, such as firewalls and intrusion detection systems.

Downtime and Loss of Productivity: During maintenance activities, there may be disruptions to the software's availability, resulting in downtime and potential loss of productivity for users.

License and Subscription Fees: If the software relies on third-party components or services, ongoing license fees or subscription costs may be incurred for their usage.

Maintenance Cost...

Vendor Support and Maintenance Contracts: If the software is obtained from a third-party vendor, there may be costs associated with vendor support and maintenance contracts.

These agreements provide access to technical assistance, updates, and patches from the vendor.

Compliance and Regulatory Costs: Compliance with industry standards, regulations, and legal requirements may require additional costs.

Software Evolution Models and Processes

- ◆ Software maintenance should have its own software maintenance life cycle (SMLC) model.
- ◆ A number of SMLC models with some variations are available in literature.
- ◆ Three common features of the SMLC models found in the literature are:
 - ◆ **Understanding the code;**
 - ◆ **Modifying the code; and**
 - ◆ **Revalidating the code.**

Software Evolution Models and Processes..

- ◆ Other models view software development as *iterative* processes and based on the idea of *change mini-cycle* as explained in the following:
 - ◆ **Iterative models:** Systems are constructed in builds, each of which is a refinement of requirements of the previous build. A build is refined by considering feedback from users.
 - ◆ **Change mini-cycle models:** These models consist of five major phases: change request, analyze and plan change, implement change, verify and validate, and documentation change.
 - ◆ In this process model, several important activities were identified, such as program comprehension, impact analysis, refactoring, and change propagation.

Software Evolution Models ...

- ◆ A different kind of software evolution model, called *staged model of maintenance and evolution*, has been proposed by Rajlich and Bennett.
- ◆ The model is descriptive in nature, and its primary objective is to improve the understanding of how long-lived software evolves.

Software Evolution Models ...

◆ The model considers four distinct, sequential stages of the lifetime of a system, as explained below:

1. **Initial development:** When the initial version of the system is produced, detailed knowledge about the system is fresh. Before delivery of the system, it undergoes many changes. Eventually, a system architecture emerges and soon it stabilizes.
 2. **Evolution:** Significant changes involve higher cost and higher risk. In the period immediately following the initial delivery, knowledge about the system is still almost fresh in the minds of the developers.
- ◆ In general, for many systems, their lifespan are spent in this stage, because the systems continue to be of importance to the organizations.

Software Evolution Models ...

3. **Servicing:** When the knowledge about the system has significantly decreased, the developers mainly focus on maintenance tasks, such as fixing bugs, whereas architectural changes are rarely effected.
4. **Phase out:** When even minimal servicing of a system is not an option, the system enters its very final stage. The organization decides to replace the system for various reasons:
 - (i) it is too expensive to maintain the system; or
 - (ii) there is a newer solution available.

Software Evolution Models ...

- ◆ ***Software Configuration Management***: Configuration management (CM) is the discipline of managing and controlling changes in the evolution of software systems.
 - ◆ The goal of CM is to manage and control the various extensions, adaptations, and corrections that are applied to a system over its lifetime.

Software Evolution Models ...

- ◆ An SCM system has four different elements, each element addressing a distinct user need as follows:
 - ◆ **Identification of software configurations:** This includes the definitions of the different artifacts, their baselines or milestones, and the changes to the artifacts.
 - ◆ **Control of software configurations:** controlling the ways artifacts or configurations are altered with the necessary technical and administrative support.
 - ◆ **Auditing software configurations:** Making the current status of the software system in its life cycle visible to management and determining whether or not the baselines meet their requirements.
 - ◆ **Accounting software configuration status:** providing an administrative history of how the software system has been altered, by recording the activities.

Reengineering

- ◆ Reengineering implies a single cycle of taking an existing system and generating from it a new system, Whereas evolution can go forever.
- ◆ To a large extent, software evolution can be seen as repeated software reengineering.
- ◆ Reengineering includes some kind of reverse engineering activities to design an abstract view of a given system.
 - ◆ The new abstract view is restructured, and forward engineering activities are performed to implement the system in its new form.
- ◆ The aforementioned process is captured by Jacobson and Lindst"orm with the following expression:
- ◆ $\text{Reengineering} = \text{Reverse engineering} + \Delta + \text{Forward engineering}$.

Reengineering...

- ◆ The first element “Reverse engineering” is the activity of defining a more abstract and easier to understand representation of the system.
 - ◆ For example, the input to the reverse engineering process is the source code of the system, and the output is the system architecture.
 - ◆ The core of reverse engineering is the process of examination of the system, and it is not a process of change.
- ◆ The third element “forward engineering” is the traditional process of moving from a high-level abstraction and logical, implementation-independent design to the physical implementation of the system.
- ◆ The second element “ Δ ” captures alterations performed to the original system.

Legacy Systems

- ◆ A legacy software system is an old program that continues to be used because it still meets the users' needs, in spite of the availability of newer technology or more efficient methods of performing the task.
- ◆ There are a number of options available to manage legacy systems.
 - ◆ Typical solution include:
 - ◆ **Freeze**: The organization decides no further work on the legacy system should be performed.
 - ◆ **Outsource**: An organization may decide that supporting software is not core business, and may outsource it to a specialist organization offering this service.

Legacy Systems...

- ♦ **Carry on maintenance:** Despite all the problems of support, the organization decides to carry on maintenance for another period.
- ♦ **Discard and redevelop:** Throw all the software away and redevelop the application once again from scratch.
- ♦ **Wrap:** It is a black-box modernization technique - surrounds the legacy system with a software layer that hides the unwanted complexity of the existing data, individual programs, application systems, and interfaces with the new interfaces.
- ♦ **Migrate:** Legacy system migration basically moves an existing, operational system to a new platform, retaining the legacy system's functionality and causing minimal disruption to the existing operational business environment as possible.

Impact Analysis

- ◆ **Impact analysis** is the task of estimating the parts of the software that can be affected if a proposed change request is made.
- ◆ Impact analysis techniques can be partitioned into two classes:
 - ◆ **Traceability analysis:** In this approach the high-level artifacts such as requirements, design, code and test cases related to the feature to be changed are identified.
 - ◆ A model of inter-artifacts such that each artifact in one level links to other artifacts is constructed, which helps to locate a pieces of design, code and test cases that need to be maintained.
 - ◆ **Dependency (or source-code) analysis:** Dependency analysis attempt to assess the affects of change on semantic dependencies between program entities.
 - ◆ This is achieved by identifying the syntactic dependencies that may signal the presence of such semantic dependencies.
 - ◆ The two dependency-based impact analysis techniques are: call graph based analysis and dependency graph based analysis.

Refactoring

- ◆ **Refactoring**: is the process of making a change to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.
- ◆ It is the object-oriented equivalent of **restructuring**.
- ◆ **Refactoring**, which aims to improve the internal structure of the code, achieve through the removal of duplicate code, simplification, making code easier to understand, help to find defects and adding flexibility to program faster.
- ◆ There are two aspects of the above definition:
 - ◆ It must preserve the “observable behavior” of the software system (through regression).
 - ◆ To improve the internal structure of a software system (improve maintainability).

Software Reuse

- ◆ Software reuse was introduced by Dough McIlroy in his 1968 seminal paper:
 - ◆ The development of an industry of reusable source-code software components and the industrialization of the production of application software from off-the-shelf components.
- ◆ Software reuse is using existing artifacts or software knowledge during the construction of a new software system.
 - ◆ Reusable assets can be either reusable artifacts or software knowledge.
- ◆ Capers Jones identified four types of reusable artifacts:
 - ◆ **data reuse**, involving a standardization of data formats,
 - ◆ **architectural reuse**, which consists of standardizing a set of design and programming conventions dealing with the logical organization of software,
 - ◆ **design reuse**, for some common business applications, and
 - ◆ **program reuse**, which deals with reusing executable code.

Software Reuse...

Benefits of Reuse

- ◆ Increased reliability
- ◆ Reduced process risk
- ◆ Increase productivity
- ◆ Standards compliance
- ◆ Accelerated development
- ◆ Improve maintainability
- ◆ Reduction in maintenance time and effort

Categories of Software

Software can be categorized into several broad categories based on its purpose, functionality, and intended use. Here are some common categories of software:

System Software:

Operating Systems (e.g., Windows, macOS, Linux): Software that manages computer hardware resources and provides a platform for other software applications to run.

Device Drivers: Software that enables communication between hardware devices and the operating system.

Utilities: Software tools that perform specific tasks like file management, disk cleanup, system optimization, and security.

Categories of Software...

Application Software:

Productivity Software: Software that helps users create, manage, and manipulate documents, spreadsheets, presentations, and databases.

Examples include word processors (e.g., Microsoft Word, Google Docs), spreadsheet software (e.g., Microsoft Excel, Google Sheets), and presentation software (e.g., Microsoft PowerPoint, Google Slides).

Communication and Collaboration Software: Software that facilitates communication, collaboration, and information sharing among individuals or groups.

Examples include email clients, instant messaging applications, video conferencing tools, and project management software.

Multimedia Software: Software for creating, editing, and playing multimedia content such as images, audio, and video.

Examples include image editing software (e.g., Adobe Photoshop, GIMP), audio editing software (e.g., Audacity), and video players/editors (e.g., VLC Media Player, Adobe Premiere).

Categories of Software...

Graphics and Design Software: Software used for graphic design, illustration, 3D modeling, and animation. Examples include Adobe Illustrator, CorelDRAW, AutoCAD, and Blender.

Entertainment Software: Software designed for entertainment purposes, such as video games, virtual reality (VR) applications, and media streaming platforms.

Educational Software: Software designed for teaching, learning, and educational purposes. This category includes e-learning platforms, educational games, and simulation software.

Categories of Software...

Financial Software: Software for financial management, accounting, tax preparation, and financial analysis.

Examples include accounting software (e.g., QuickBooks, Xero) and tax preparation software (e.g., TurboTax).

Enterprise Software: Software systems used by organizations to support their business processes and operations.

This includes customer relationship management (CRM) software, enterprise resource planning

Categories of Software...

Embedded Software:

Embedded software is built into hardware devices and performs specific functions.

It is commonly found in devices such as smartphones, automobiles, consumer electronics, industrial equipment, and medical devices

Categories of Software...

Middleware:

Middleware software acts as a bridge between different software applications, enabling communication and data exchange.

Examples include web servers, application servers, message queues, and database management systems.

Development Tools:

Software development tools are used by programmers and developers to create, test, debug, and maintain software applications.

Examples include integrated development environments (IDEs), compilers, debuggers, and version control systems.



Discussion..