Software Engineering

Lecture 11

Software Maintenance

Dr. Obuhuma James

Description

This topic covers software maintenance as a phase that occurs after testing of the system. The three types of maintenance will be covered, namely, maintenance to adapt to a different environment, maintenance to fix bugs, and maintenance to add new functionalities. The topic also covers the cost factors associated with the cost of software maintenance. In addition, the concept of system reengineering as applied to software engineering has been explored.

Learning Outcomes

By the end of this topic, you will be able to:

- Discuss the concept of software maintenance as applied in Software Engineering.
- Differentiate among the three types of software maintenance.
- Discuss the concept of system reengineering in relation to software maintenance.

Overview of Software Maintenance

Upon completion of system development, the system is deployed to the customer's environment for use. At this point, the software maintenance stage commences, whereby software maintenance involves any modifications and updating activities that are applied to the system during its usage to either fix errors or bugs, add new functionalities, or adapt the system to a new environment. The modifications may be triggered by market conditions, client requirements, host modifications, and organizational changes. Unfortunately, many businesses spend a great deal of money on maintenance [2].

Software Maintenance

After a system has been deployed, it must be maintained in form of modification and keeping it up to date. According to Kendall and Kendall [2], system maintenance takes up 48 to 60 percent of the total time spent on system development in any given department. Software maintenance is carried out for two main reasons [2]:

1. Maintenance to correct errors.
2. Maintenance to enhance software capabilities. This may be as a result of the following
    a) Users requests for new functionalities.
    b) Business changes over time.
    c) Frequent changes in hardware and software.

Based on the above stated points, software maintenance can be summarized as an ongoing process in the life cycle of a software system. The term software maintenance is used for changing custom software while evolution is used for generic software products [1]. Hence, in most cases, maintenance rarely involves major changes to a given system's architecture since changes are mainly implemented by modifying existing components and adding new components to the system.

Types of Maintenance

The type of maintenance varies based on the nature of the software. Thus, maintenance may just be a routine task upon discovery of bugs or a large major issue of its own kind. The various types of maintenance are hence visualized differently by different authors and contexts. For instance, types of maintenance based on their characteristics can be corrective maintenance, adaptive maintenance, perfective maintenance, and preventive maintenance. Sommerville [1] categorized maintenance types as three

- Maintenance to repair software faults

  This entails altering a system to correct deficiencies in such a manner that makes it meet its requirements. This type of maintenance can be mapped to the corrective maintenance type.

- Maintenance to adapt software to different operating environments

  This entails altering a system to fit it in a new different environment from its initial or current environment. This type of maintenance can be mapped to the adaptive maintenance type.

- Maintenance to add to or modify the system's functionality

  This entails modifying a system to accommodate new requirements. This type of maintenance can be mapped to the perfective maintenance type.

According to Sommerville [1], maintenance effort distribution is as outlined in Figure 1 where 17% effort is spent on repair of faults, 18% on environmental adaptation, as 65% gets spent on functionality addition or modification.

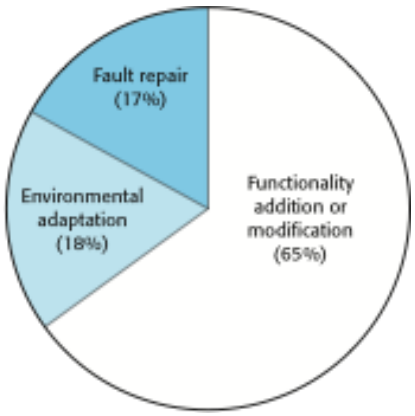

Figure 1. Distribution of Maintenance Effort [1]

Cost of Maintenance

The ease of system maintenance depends on how the system was designed. Thus, according to Kendall and Kendall [2], the better the system design, the easier it will be to maintain and the lesser the money that will be required for maintenance. In case proper mechanisms for early detection and correction of errors during system design are put in place, it results in a less costly process compared to keeping the errors unnoticed until maintenance stage. On average, the cost of maintenance is always higher than the total cost incurred in the other phases of system development. The percentage distribution is as outlined in Figure 2.

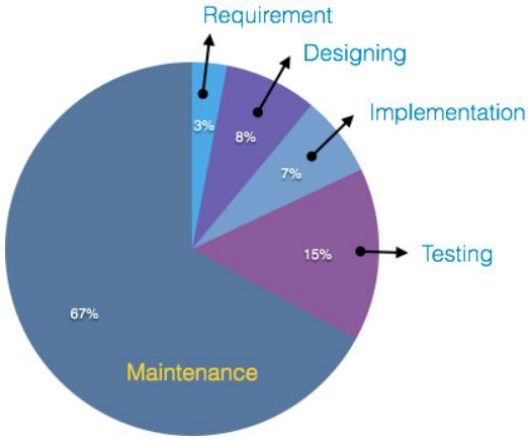

Figure 2. Distribution of Maintenance Cost

According to Sommerville [1] and in relation to Figure 2, the cost of software maintenance is usually 2 to 100 times greater than the development cost depending on the type and nature of the application. The following are some real-world factors that affect maintenance cost

- The cost is normally affected by both technical and non-technical factors around the software.
- The cost keeps increasing during the maintenance lifetime of the software since under normal circumstances, maintenance corrupts the structure of the software making further maintenance more difficult.
- As software ages, the cost of support increases. This may be due to old programming languages and compilers, degradation of structure, among others.
- In most cases software changes during maintenance are not documented. This may result to more conflicts in future.
- Technological advances make it more costly to maintain old software.
- The stability of the development and maintenance team is a determinant on the maintenance cost. In an ideal case, the cost of maintenance reduces if the same maintenance team is engaged as opposed to change of team.
- Lack of contractual responsibility for maintenance for software developers may result to no incentive to design for future change.
- The skill levels of the maintenance staff have a direct impact on the resulting cost of maintenance.

It is thus essential to have mechanisms in place that could aid in maintenance prediction. Such predictions result to an assessment of parts of the system that are deemed to be likely to cause problems and to be having high maintenance costs.

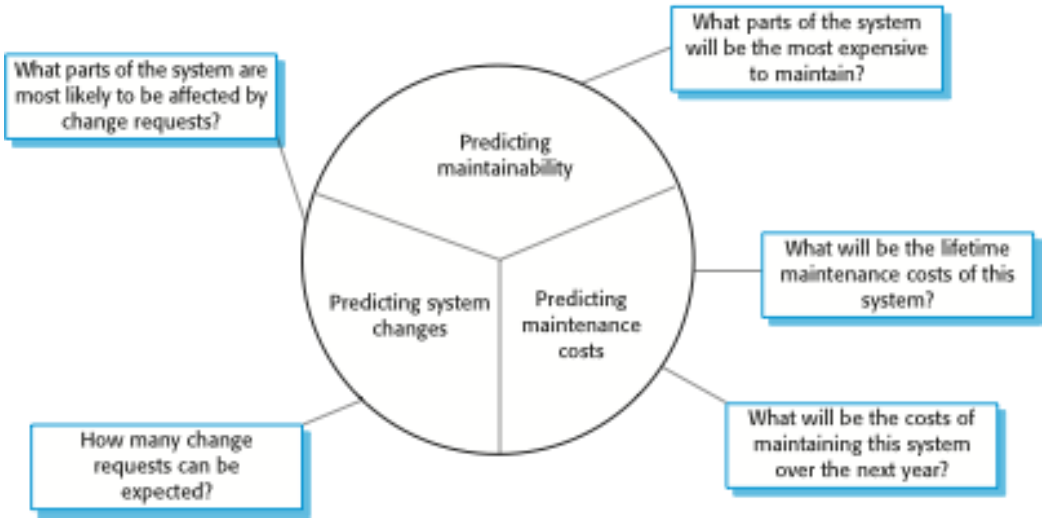

Figure 3. Prediction of Maintenance [1]

Figure 3 shows a guide to maintenance predictions where according to Sommerville [1]

- Change acceptance depends on the maintainability of the components affected by the change.
- Implementing changes degrades the system and reduces its maintainability.
- Maintenance costs depend on the number of changes and costs of change depends on maintainability.

Better prediction of the number of changes requires a proper understanding of the relationship between a given system and its environment. This is influenced by the number and complexity of system interfaces, number of inherently volatile system requirements, and the business processes in the system's environment [1]. In this case,

- System complexity relates to the complexity of control structures, complexity of data structures, size of objects, methods and modules.
- Process metrics for assessing maintainability include the number of corrective maintenance requests, average impact analysis time required, average time taken for change request implementation, and the number of outstanding change requests.

In a real case scenario, a rise in any of the factors serves as an indicator for a decline in maintainability.

System Reengineering

Over time, the cost of maintenance becomes unacceptably high, mainly as a result of cases where some if not all subsystems of a larger system warrant for frequent maintenance. Such cases trigger for a need for system re-engineering where system reengineering entails restructuring or re-writing part or all of a legacy system without changing its functionality [1]. Through system reengineering, the system ends up being restructured and redocumented with the prime goal of returning it to a state that allows for easy maintenance. System reengineering reduces risks in software development and reduces costs since the software is not development afresh from scratch. Figure 4 shows the process of system reengineering.

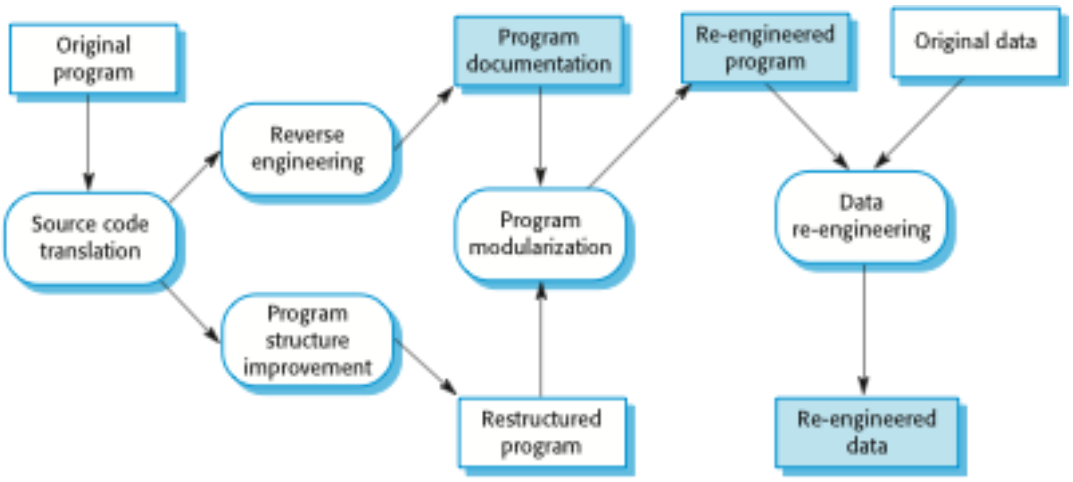

Figure 4. System Reengineering Process [1]

The cost of reengineering depends on a number of factors including the quality of the software to be reengineered, availability of tools for reengineering, amount of data conversion required, and the availability of reengineering experts.

Preventative maintenance can be achieved through refactoring which is a process of making of improving a software to slow down degradation through change [1]. According to Sommerville [1], refactoring involves program modification aimed at improving the program structure, reducing program complexity or making it easier for program understanding. Through refactoring, the prime goal lies in program improvement rather than addition of functionalities.

Software Evolution and Servicing

As part of a software's lifetime, it must undergo through some form of evolution and servicing as outlined in Figure 5.

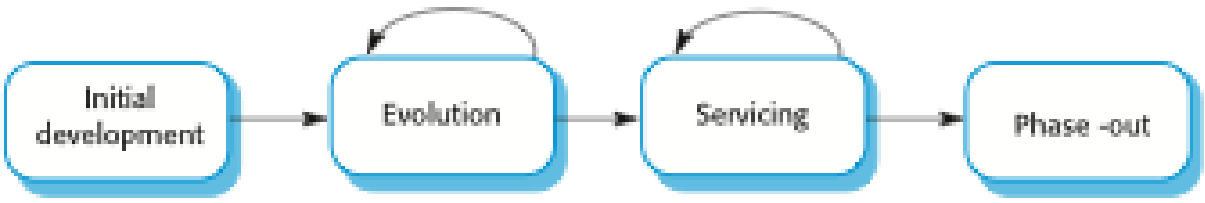

Figure 5. Software Evolution and Servicing Process Summary [1]

Software evolution is a stage in a software's lifetime where it is in operational use and is growing as new requirements are proposed and applied. Figure 6 shows the software evolution process. Software servicing on the other hand applies to a state when the software

is still in use with the only changes made remaining to be those needed to keep it operational. These includes fixing of bugs and address to changes in the software environment. It is worth noting that no new functionality is added during the software servicing process. Finally, during phase-out, the software may remain useful, however, no further changes are done on it.

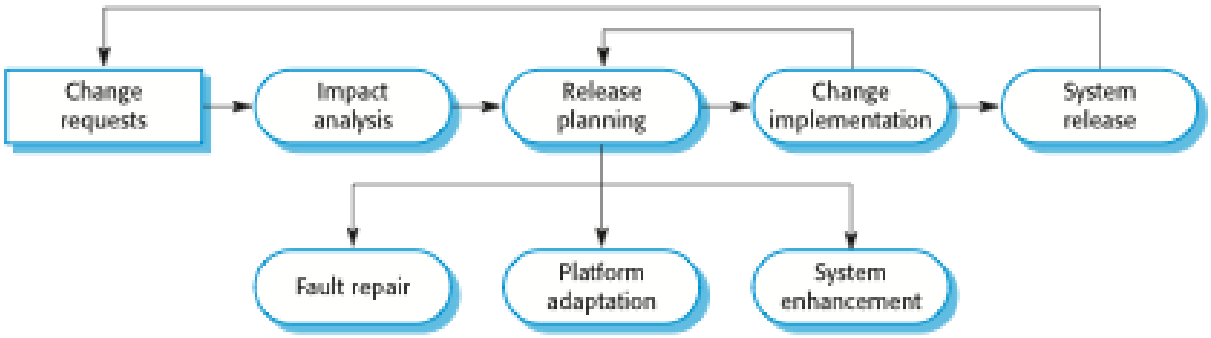

Figure 6. Software Evolution Process [1]

Summary

The topic has explored software maintenance at length. The various types of maintenance have been covered including corrective maintenance, adaptive maintenance, and perfective maintenance. Preventive maintenance has also been covered as a system refactoring concept. The factors attributed to cost of maintenance have been clearly outlined. The concept of software reengineering as a way of restructuring or re-writing part or all of a legacy system without changing its functionality has also been covered. Finally, the relationship between software evolution and software servicing has been outlined.

Check Points

1. Discuss the concept of software maintenance as applied to software engineering.
2. State and explain the various types of software maintenance.
3. Relate the cost of maintenance to the cost involved in other phases of the system development lifecycle.
4. Discuss the mechanisms used in prediction of software maintenance.
5. Discuss the concept of system reengineering as applied to software engineering.
6. Differentiate between software evolution and software servicing.

Core Textbooks

1. Sommerville, I., Software Engineering, 10th Edition, Addison Wesley, 2016.

Other Resources

References

[1] Sommerville, I., Software Engineering, 10th Edition, Addison Wesley, 2016.

[2] Kendall, K. E. and Kendall, J. E., Systems Analysis and Design, 8th Edition, Pearson, 2011.

[3] Pries, K. H., & Quigley, J. M. (2010). Scrum project management. CRC press.

[4] Schwaber, K. (2004). Agile project management with Scrum. Microsoft press.

[5] Gotterbarn, D., Miller, K., & Rogerson, S. (1997). Software engineering code of ethics. Communications of the ACM, 40(11), 110-118.

[6] Pressman, R. S. (2005). Software engineering: a practitioner's approach. Palgrave macmillan.