Chapter 9: Software Evolution

9.1. Explain how advances in technology can force a software subsystem to undergo change or run the risk of becoming useless.

Because new technologies often offer improved capabilities, security, or efficiency. Failure to adapt can lead to obsolescence and a loss of value to users and organizations

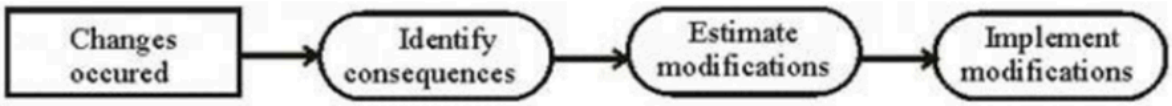
New widespread technological advancements need not always be **backward compatible** with all types of older systems.

Newly designed usage interfaces data volume exchanges, and formats may all require existing systems to undergo updates, or risk becoming obsolete:

- Example 1: With the introduction of widespread cloud-based services, software systems need to adapt to be able to use the cloud infrastructure for their operations similar to what competitive products may be doing.
- Example 2: An application that allowed mobile applications to connect in an ad-hoc manner needs to adapt as newer communication technology is introduced to new releases of mobile devices. If the applications do not adapt, they risk being obsolete

9.2. From Figure 9.4, you can see that impact analysis is an important subprocess in the software evolution process. Using a diagram, suggest what activities might be involved in change impact analysis.

Following diagram shows the activities that might be involved in the change in the impact analysis.



Software evolution processes is vary and depending on the type of software being maintained. And, the change impact analysis includes the activities like identifying the consequences of the changes occurred, estimating the modifications need to be done to accomplish the changes and implementing the modifications

9.3. Explain why legacy systems should be thought of as sociotechnical systems rather than simply software systems that were developed using old technology.

Ans Socio-technical systems – Systems that include technical systems but also operational processes and people who use and interact with the technical system. Socio-technical systems are governed by organisational policies and rules.

Socio-technical system characteristics

- Emergent properties – Properties of the system as a whole that depend on the system components and their relationships.
- Non-deterministic – They do not always produce the same output when presented with the same input because the system's behaviour is partially dependent on human operators.
- Complex relationships with organisational objectives – The extent to which the system supports organisational objectives does not just depend on the system itself.

Sociotechnical systems include hardware, software, libraries and other supporting software and business processes

1. System hardware
2. Support software
3. Application software
4. Application data
5. Business processes

6. Business policies and rules

9.4. Some software subsystems are seen as “low quality, high business value.” Discuss how those subsystems can be re-engineered with minimal impact on the operations of the organization.

"Low quality, high business value" subsystems can be re-engineered with minimal impact by thoroughly analysing the subsystem's flaws.

Also, identifying critical functionality to preserve. Moreover, developing a comprehensive plan for refactoring or rewriting. In addition, it is important to implement changes incrementally, testing at each step. So, continuously monitoring and improving quality while ensuring no disruption to core operations can lead to minimal impact on the operations of the organization.

9.5. What are the strategic options for legacy system evolution? When would you normally replace all or part of a system rather than continue maintenance of the software?

Strategic options for legacy system evolution:

1. Abandon or Replace
2. Continue Maintenance
3. Re-engineering (System Improvement)
4. Encapsulation with New Functionality
5. Decomposition into Components

The decision to replace all or part of a system rather than continue maintenance is typically driven by various factors like hardware changes, strategic misalignment, sub-system Replacements, and low technical quality.

9.6. Explain why problems with support software might mean that an organization has to replace its legacy systems.

Support software is a term that refers to the software that helps maintain and operate a legacy system, such as operating systems, databases, compilers, debuggers, etc. Problems with support software might mean that an organization must replace its legacy systems for several reasons:

- **Lack of vendor support**: If the vendor of the support software discontinues the product or stops providing updates and maintenance, the organization might face difficulties in resolving issues, fixing bugs, or enhancing the functionality of the legacy system[1].
- **Security risks**: Support software that is outdated or no longer updated might expose the legacy system to cyberattacks, data breaches, or malware infections. This could compromise the confidentiality, integrity, and availability of the system and the data it processes[2].
- **Integration challenges**: Support software that is incompatible with newer systems or technologies might prevent the legacy system from communicating or exchanging data with other systems. This could limit the interoperability, scalability, and efficiency of the system and the organization[3].
- **High maintenance costs**: Support software that is scarce, complex, or poorly documented might require a lot of resources to maintain and operate. This could include hiring or training specialized staff, purchasing or upgrading hardware, or investing in custom solutions. These costs could outweigh the benefits of keeping the legacy system

9.7. As a software project manager in a company that specializes in the development of software for the offshore oil industry, you have been given the task of discovering the factors that affect the maintainability of the systems developed by your company. Suggest how you might set up a program to analyze the maintenance process and determine appropriate maintainability metrics for the company.

Analysing the factors affecting maintainability and establishing a program to evaluate these factors in existing systems with known maintenance costs is a comprehensive task.

First, we need to start by collecting historical maintenance data for existing systems. This data should include details of maintenance activities, time spent, resources allocated, and associated costs.

After, we need to recognize the key factors that influence maintainability, such as program and data complexity, meaningful identifiers, programming language, and program documentation.

Also, we need to create a set of maintainability metrics that align with the identified factors. Each metric should be quantifiable and suitable for assessing different aspects of maintainability. It is important to assess the choice of programming languages used in the existing systems. We need to consider factors like language features, readability, and community support. By following this approach, we identify high-cost program units and implement targeted improvements to reduce maintenance costs and enhance the overall quality of your company's software systems in the offshore oil industry.

9.8. Briefly describe the three main types of software maintenance. Why is it sometimes difficult to distinguish between them?

Ans Three main types of software maintenance:

1. **Corrective Maintenance:** This involves making changes to the software to repair reported faults. These faults can include program bugs, specification errors, or omissions. The primary goal is to ensure that the software functions correctly and reliably.

2. **Adaptive Maintenance:** This type of maintenance focuses on changing the software to adapt it to changes in its environment. This can include adjustments to accommodate changes in other software systems, operating systems, hardware, or external dependencies. The goal is to keep the software compatible with its evolving surroundings.

3. **Perfective Maintenance:** Perfective maintenance is about enhancing the software by adding new functionality or features to improve its capabilities. This type of maintenance aims to make the software more efficient, user-friendly, or feature-rich.

It's important to note that these maintenance types can sometimes overlap or be intertwined. For example, addressing a reported fault might involve both corrective and adaptive maintenance if it requires updating other software components or systems. Additionally, when adapting the software to changes in its environment, you might take the opportunity to add new features, blurring the line between adaptive and perfective maintenance. Understanding these maintenance types and their potential overlap is essential for effective software maintenance planning and execution.

9.9. Explain the differences between software reengineering and refactoring?

Ans Refactoring is the process of changing a software system so that it does not alter the external behaviour of the code yet improves its internal structure.

- clean up code that minimizes the chances of introducing bugs.
- improving the design of the code after it has been written.
- refactoring does not add features or functionalities to a software system.
- It makes a software system more accessible to understand and cheaper to modify without changing its observable behaviour by changing its internal structure.

The purposes of refactoring:

1. Refactoring Improves the Design of Software
2. Refactoring Makes Software Easier to Understand
3. Refactoring Helps Finding Bugs
4. Refactoring Helps Programming Faster

Software Re-engineering means - reorganizing or re-structuring or modifying existing software systems to make them more maintainable.

When to re-engineer:

- When system changes are mostly confined to part of the system then re-engineer that part
- When hardware or software support becomes obsolete
- When tools to support re-structuring are available

The purpose of re-engineering:

1. To explain why software re-engineering is a cost-effective option for system evolution
2. To describe the activities involved in the software re-engineering process
3. To distinguish between software and data re-engineering and to explain the problems of data re-engineering

Re-engineering	Refactoring
It is a maintenance process, so that the understandability and the structure of the program can be improved	The original structure of the program is improved. By doing so, the complexity of the program can be reduced
It can be applied even to legacy software	It is limited to object oriented development programs
You can add any new functionality to the already existing system	Addition of new functionality is not allowed
Reverse engineering is allowed	It is based on agile methods, so reverse engineering is not allowed

9.10. Do software engineers have a professional responsibility to develop code that can be easily maintained even if their employer does not explicitly request it?

Yes, It is their responsibility to

- develop a code that is easy to understand,
- maintained,
- changed and
- flexible

for example, it is a system that is meant to last a long time and will have many developers working on it. A software developer will want to reduce the future cost of maintaining their software.

Good software engineering techniques such as

- precise specification,
- test-first development,
- -the use of object-oriented development,
- configuration management

to help reduce the cost of future maintenance promote an ethical approach to the practice of the profession. This can only be done by carefully constructing software that is in the best interests of the employer.