Chapter 7: Software Evolution

Software evolution is the process of modifying software systems to meet changing needs. This can be due to changes in business needs, user expectations, or technology. Software evolution can be expensive, but it is necessary to maintain the value of software systems.

Reasons for Software Evolution

Software systems need to evolve and adapt throughout their lifespan to remain relevant and useful. Various factors drive the need for software change, including:

- Business changes and evolving user expectations: As businesses adapt to changing market conditions and customer demands, their software systems need to align with these shifts. New features, functionalities, and integrations may be required to address emerging needs and maintain a competitive edge.

- Error correction and bug fixes: Software inevitably encounters bugs and errors during operation. These issues need to be identified, analyzed, and resolved promptly to ensure the system's stability, reliability, and security.

- Hardware and software platform changes: Technological advancements in hardware and software platforms often necessitate adaptations to existing software systems. Compatibility issues, performance optimizations, and resource utilization adjustments may be required to ensure seamless operation on updated platforms.

- Performance enhancements and non-functional improvements: Software systems may undergo modifications to enhance their performance, efficiency, scalability, and other non-functional characteristics. This could involve optimizing algorithms, data structures, and resource management techniques.

- Competitive landscape and feature adoption: To stay ahead of the competition, software products and apps need to evolve and incorporate new features that rival offerings may introduce. Keeping up with industry trends and user expectations is crucial for maintaining market relevance.

Models of Software Evolution

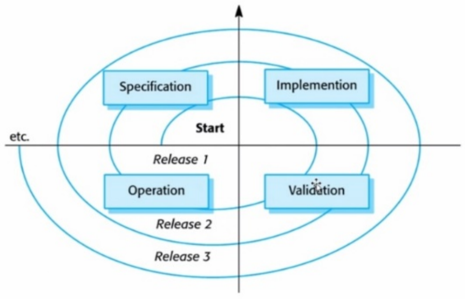
1. Iterative Software Development Lifecycle

This model is characterized by repeating cycles of development, allowing for continuous refinement and expansion of the system with each iteration.

Here's a breakdown of the stages:

- Specification: This is the first stage where the requirements of the software are gathered and analyzed. It involves understanding and defining what the software is supposed to achieve.
- Implementation: In this stage, the software is developed and coded. The specifications are translated into a software product.
- Validation: This stage involves testing the software to ensure it meets the specified requirements and functions as intended. Any bugs or issues found are addressed.
- Operation: Once validated, the software is deployed and used in its intended environment. Feedback is gathered from the users during this stage.

The labels "Release 1", "Release 2", and "Release 3" on the outer edge of the circle indicate the iterative nature of this model. After the operation stage of "Release 1", the process can loop back to the specification stage for "Release 2", allowing for the incorporation of changes and improvements. This cycle can continue for subsequent releases, enabling the software to evolve according to user feedback and changing requirements. This model is particularly effective when requirements are expected to change or evolve over the course of the project.



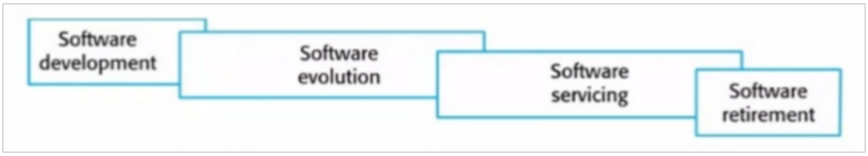
This model of software evolution is applicable when the same company is responsible for the software throughout its lifetime. There is a seamless transition from development to evolution, and the same software development methods and processes are applied throughout the lifetime of the software. Software products and apps are developed using this approach.

2. Staged Model of Software Lifecycle

The Staged Model of Software Lifecycle is a model proposed by Keith Bennett and Vaclav Rajlich that presents the software lifecycle as a series of stages that every software system goes through. Here's a breakdown of the stages:

1. Software Development: This is the initial stage where the software is being developed for the first time.
2. Software Evolution: After the software has been developed and is in use, it enters this stage. Here, the software is continually updated and improved to meet changing user needs and to add new features.
3. Software Servicing: In this stage, only minor adaptations are made to the software, and no new features are added. The focus is on maintaining the software's functionality and fixing any issues that arise.
4. Software Retirement: This is the final stage where the software is discontinued and no longer in use.

The blue line connecting these stages represents the progression of time. This model is particularly useful for understanding and managing the evolution of business systems.



Evolution Processes

As with all software processes, there is no such thing as a standard software change or evolution process. The most appropriate evolution process for a software system depends on:

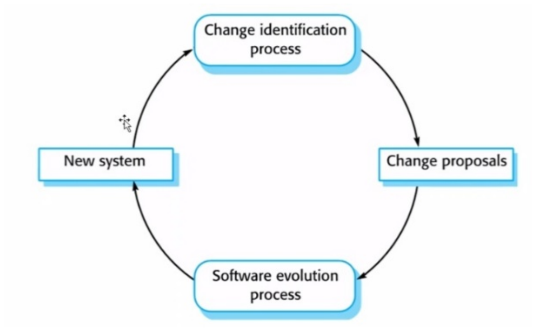
- the type of software being maintained
- the software development processes being used
- the skills of the people involved

Change Identification and Evolution Processes

This process is a part of the software development lifecycle where the software undergoes changes and evolution after its initial development.

Here's a breakdown of the stages:

1. New System: This is the initial stage where a new software system is proposed and developed.
2. Change Identification Process: This is the stage where the proposed changes are identified, analyzed, and documented.
3. Change Proposals: After the system is in use, users or stakeholders may propose changes to improve the system or to adapt it to new requirements or environments.
4. Software Evolution Process: In this stage, the identified changes are implemented in the system. This could involve modifying the existing system, adding new features, or improving performance.



After changes are implemented, new change proposals may arise, leading to further identification and implementation of changes. This cycle continues throughout the lifetime of the software, allowing it to evolve and adapt over time.

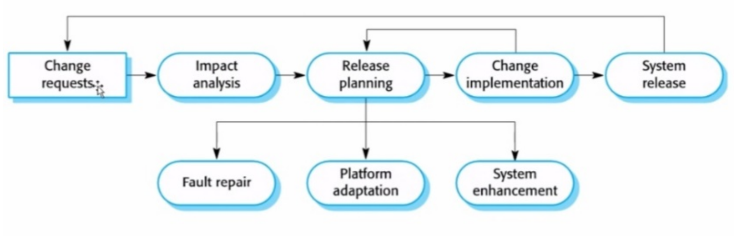
Activities Involved in the Software Evolution Process

Here's a breakdown of the stages:

1. Change Requests: This is the initial stage where users or stakeholders propose changes to improve the system or to adapt it to new requirements or environments.
2. Impact Analysis: In this stage, the proposed changes are identified, analyzed, and documented. The impact of the changes on the system is assessed.
3. Release Planning: This stage involves planning the implementation of the changes. It includes scheduling, resource allocation, and coordination with different teams.

4. Change Implementation: In this stage, the identified changes are implemented in the system. This could involve modifying the existing system, adding new features, or improving performance.

5. System Release: Once the changes have been implemented and tested, the new version of the system is released for use.



Legacy Systems

Legacy systems are older systems that rely on languages and technology that are no longer used for new systems development. Typically, they have been maintained over a long period, and their structure may have been degraded by the changes that have been made.

Logical Parts of a Legacy System

Legacy systems are indeed complex sociotechnical systems that encompass various elements, including hardware, software, libraries, supporting software, and business processes. These components intertwine to form the foundation of a legacy system.

1. System Hardware: Legacy systems might have been designed for hardware that is no longer readily available, leading to compatibility issues and performance limitations.

2. Support Software: Legacy systems may rely on various support software, such as operating systems, databases, and middleware, which might be outdated or no longer supported by the vendors. This can pose security vulnerabilities and maintenance challenges.

3. Application Software: The application system, responsible for providing business services, typically consists of multiple application programs. These programs might be written in obsolete programming languages, making them difficult to modify or integrate with newer technologies.

4. Application Data: Application data, processed by the application system, might be inconsistent, duplicated, or stored in different databases. This can hinder data integrity and complicate data analysis.

5. Business Processes: Business processes are workflows designed to achieve specific business objectives. They might be tightly coupled with the legacy system, limiting flexibility and adaptability to changing business needs.
6. Business Policies and Rules: Business policies and rules define how the business operates and impose constraints on its operations. The legacy system might be embedded within these policies and rules, making it challenging to modify or replace the system without affecting the business's core processes.

Several Reasons Why it is Expensive and Risky to Replace Legacy Systems with New Systems

1. Lack of Complete Specification: Legacy systems often lack a complete, up-to-date specification. This makes it challenging to create a new system that is functionally identical to the existing one.
2. Entwined Business Processes: Business processes often evolve alongside legacy systems, taking advantage of their services and working around their shortcomings. Replacing the system necessitates changes to these processes, which can lead to unpredictable costs and consequences.
3. Embedded Business Rules: Legacy systems often contain important business rules that may not be documented elsewhere. Losing these rules during a system replacement can have unpredictable business consequences.
4. Inherent Risks in New Software Development: Developing new software comes with inherent risks, including potential delays and cost overruns. There may also be unexpected problems with the new system.

Legacy System Management

When you are assessing a legacy system, you have to look at it from both a business perspective and a technical perspective. From a business perspective, you have to decide whether or not the business needs the system. From a technical perspective, you have to assess the quality of the application software and the system's support software and hardware. You then use a combination of the business value and the system quality to inform your decision on what to do with the legacy system.

Factors used in Business Value Assessment

Factor	Questions
Use of the System	How frequently is the system used? How many users rely on the system? What is the importance of the system's use cases to the business?
Business Processes that are Supported	Does the system's inflexibility hinder the evolution of business processes? Are the business processes supported by the system efficient and up to date? Does the system allow for adapting business processes to changing market conditions?
System Dependability	How reliable is the system in terms of technical performance and uptime? How quickly and effectively are system problems resolved? What is the impact of system disruptions on critical business processes?
System Outputs	How important are the system's outputs to the successful functioning of the business? Could the system's outputs be produced more cheaply elsewhere? How frequently are the system's outputs used?

Factors used in Environment Assessment

Factor	Questions

Factor Supplier stability	Is the supplier still in existence? Is the supplier financially stable and likely to continue in existence? If the supplier is no longer in business, does someone else maintain the systems?
Failure rate	Does the hardware have a high rate of reported failures? Does the support software crash and force the system to restart?
Age	How old is the hardware and software? The older the hardware and support software, the more obsolete it will be. It may still function correctly, but there could be significant economic and business benefits to moving to a more modern system.
Performance	Is the performance of the system adequate? Do performance problems have a significant effect on system users?
Support requirements	What local support is required by the hardware and software? If high costs are associated with this support, it may be worth considering system replacement.
Maintenance costs	What are the costs of hardware maintenance and support software licenses? Older hardware may have higher maintenance costs than modern systems. Support software may have high annual licensing costs.
Interoperability	Are there problems interfacing the system to other systems? Can compilers, for example, be used with current versions of the operating system?

Factors used in Application Assessment

Factor	Questions
Understandability	How difficult is it to understand the source code of the current system? How complex are the control structures that are used? Do variables have meaningful names that reflect their function?
Documentation	What system documentation is available? Is the documentation complete, consistent, and current?
Data	Is there an explicit data model for the system? To what extent is data duplicated across files? Is the data used by the system up to date and consistent?
Performance	Is the performance of the application adequate? Do performance problems have a significant effect on system users?
Programming language	Are modern compilers available for the programming language used to develop the system? Is the programming language still used for new system development?
Configuration management	Are all versions of all parts of the system managed by a configuration management system? Is there an explicit description of the versions of components that are used in the current system?
Test data	Does test data for the system exist? Is there a record of regression tests carried out when new features have been added to the system?
Personnel skills	Are there people available who have the skills to maintain the application? Are there people available who have experience with the system?

Ideally, objective assessment should be used to inform decisions about what to do with a legacy system.

Software Maintenance

Software Maintenance is the process of modifying a system after it has been delivered. Changes can range from simple coding corrections to extensive design modifications and significant enhancements to meet new requirements.

The three types of software maintenance

1. Corrective Maintenance

Objectives: Primarily focuses on identifying, analyzing, and correcting errors, bugs, and vulnerabilities within the software. I

Impact: Coding errors are generally less expensive to rectify, while design errors may incur higher costs due to the potential rework of multiple program components. Requirements errors pose the most significant financial burden as they may necessitate extensive system redesign.

2. Adaptive Maintenance

Objectives: Focuses on adapting the software to accommodate changes in the operating environment, such as hardware upgrades, new operating systems, or modified support software.

Impact: Application systems may require modifications to maintain compatibility and functionality within the evolving environment.

3. Perfective Maintenance

Objectives: Enhances the software's functionality by introducing new features, improving performance, or addressing changing business requirements.

Impact: The scope of changes required for perfective maintenance often exceeds that of corrective or adaptive maintenance, potentially involving substantial modifications to the software architecture and codebase.

4. Preventative Maintenance

Objectives: Aims to prevent future problems by improving the overall structure, maintainability, and understandability of the software.

Impact: Refactoring, making small program changes that preserve functionality, is a common technique for preventative maintenance. By refactoring, developers can reduce the complexity of the codebase, making it easier to understand and modify, thereby reducing the likelihood of future errors and enhancing the overall maintainability of the software.

Maintenance Prediction

Maintenance prediction is the process of anticipating and assessing the potential changes required for a software system. It aims to identify the most likely areas of change and their associated costs, enabling proactive measures to enhance maintainability and reduce overall maintenance expenses.

By predicting changes, you can also assess the overall maintenance costs for a system in a given period and set a budget for maintaining the software. Here are the possible predictions and the questions that these predictions may answer.

Maintenance Prediction Questions

Factor	Questions
Predicting maintainability	What parts of the system will be the most expensive to maintain?
Predicting maintenance costs	What will be the lifetime maintenance costs of this system? What will be the costs of maintaining this system over the next year?
Predicting system changes	How many change requests can be expected? What parts of the system are most likely to be affected by change requests?

Software maintainability process metrics

Process metrics may be used to assess software maintainability; if any or all of these are increasing, this may indicate a decline in maintainability:

1. Number of Requests for Corrective Maintenance: An increase in bug and failure reports may indicate more errors are being introduced than are being repaired, suggesting a decline in maintainability.

2. Average Time Required for Impact Analysis: If the time for impact analysis increases, it implies that more components are affected by changes, indicating a decrease in maintainability.

3. Average Time Taken to Implement a Change Request: This is the time needed to modify the system and its documentation after assessing which components are affected. An increase in this time may suggest a decline in maintainability.

4. Number of Outstanding Change Requests: An increase in this number over time may imply a decline in maintainability.

Software Reengineering

Software reengineering encompasses a set of techniques aimed at improving the structure, understandability, and maintainability of existing software systems, particularly legacy systems. It involves activities such as redocumenting the system, refactoring the codebase, translating programs to modern programming languages, and modifying data structures.

Advantages of Reengineering over Replacement

1. Reduced Risk: Replacing critical software systems poses significant risks, including potential errors in system specifications, development delays, and business disruptions. Reengineering, on the other hand, involves modifying the existing system, minimizing the risk of introducing new errors or disrupting business operations.

2. Reduced Cost: Reengineering is often a more cost-effective approach compared to complete redevelopment. By leveraging the existing system as a foundation, reengineering can significantly reduce development time and effort, leading to lower overall costs.

~ end of Chapter 7 ~