CHAPTER 9

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1 - 20

\*\*Software Change:\*\*

- Inevitable due to new requirements, business environment changes, errors, new hardware, and performance improvements.

- Organizations face challenges in implementing and managing change in existing software systems.

\*\*Importance of Evolution:\*\*

- Organizations heavily invest in software systems, critical business assets.

- Regular changes and updates are necessary to maintain the value of these assets.

- Majority of software budget in large companies is allocated to evolving existing software rather than developing new.

\*\*Evolution Processes:\*\*

- \*\*Evolution:\*\*

- Ongoing changes in a software system's operational use to implement new requirements.

- \*\*Servicing:\*\*

- Changes to keep the software operational (bug fixes and adaptations) with no new functionality.

- \*\*Phase-out:\*\*

- Software still in use but no further changes.

\*\*Evolution Processes (continued):\*\*

- Depend on software type, development processes, and the skills of those involved.

- Change proposals drive system evolution.

- Identification and evolution processes continue throughout the system's lifetime.

\*\*Change Implementation:\*\*

- Iterative development process for designing, implementing, and testing revisions.

- Initial stage may involve program understanding, especially if original developers are not involved.

\*\*Urgent Change Requests:\*\*

- Implemented without going through all software engineering stages in emergencies (e.g., system faults, unexpected effects from environment changes, urgent business changes).

- Emergency repair process diagram provided.

\*\*Agile Methods and Evolution:\*\*

- Agile methods support incremental development, making the transition from development to evolution seamless.

- Automated regression testing is valuable during system changes.

- Changes expressed as additional user stories.

\*\*Handover Problems:\*\*

- Issues when the development team uses agile, but the evolution team prefers plan-based (or vice versa).

- Problems may arise due to documentation expectations, testing approaches, and code complexity.

\*\*Program Evolution Dynamics:\*\*

- Study of the processes of system change.

- Lehman and Belady proposed 'laws' applicable to large systems developed by large organizations.

- Laws are more like sensible observations, and their applicability to other software systems is uncertain.

\*\*Lehman’s Laws:\*\*

- \*\*Continuing Change:\*\*

- Programs in a real-world environment must change to remain useful.

- \*\*Increasing Complexity:\*\*

- Evolving programs tend to become more complex; extra resources are needed for preservation and simplification.

- \*\*Large Program Evolution:\*\*

- System attributes like size, time between releases, and reported errors are approximately invariant for each release.

- \*\*Organizational Stability:\*\*

- Over a program’s lifetime, the rate of development is approximately constant.

- \*\*Conservation of Familiarity:\*\*

- Incremental change in each release is approximately constant.

- \*\*Continuing Growth:\*\*

- Functionality must continually increase to maintain user satisfaction.

- \*\*Declining Quality:\*\*

- Quality declines without modification to reflect changes in the operational environment.

- \*\*Feedback System:\*\*

- Evolution processes incorporate feedback systems for significant product improvement.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*21-40

\*\*Applicability of Lehman’s Laws:\*\*

- Generally applicable to large, tailored systems developed by large organizations.

- Uncertain modification for shrink-wrapped software, systems with COTS components, small organizations, and medium-sized systems.

\*\*Key Points:\*\*

- Software development and evolution are integrated, iterative processes represented by a spiral model.

- For custom systems, maintenance costs usually exceed development costs.

- Software evolution is driven by change requests, involving change impact analysis, release planning, and change implementation.

- Lehman’s laws describe insights from long-term studies of system evolution.

\*\*Software Maintenance:\*\*

- Modifying a program after deployment, typically for custom software.

- Generic software products evolve to create new versions.

- Maintenance involves modifying existing components and adding new ones without major architectural changes.

- Types: fault repair, adaptation to a different environment, and adding/modifying functionality.

\*\*Maintenance Effort Distribution:\*\*

- Diagram provided, showing how maintenance efforts are distributed.

\*\*Maintenance Costs:\*\*

- Usually greater than development costs (2\* to 100\* depending on the application).

- Affected by both technical and non-technical factors.

- Increases with software maintenance, making further maintenance difficult.

- Ageing software can have high support costs.

\*\*Maintenance Cost Factors:\*\*

- Team stability, contractual responsibility, staff skills, program age, and structure.

- Maintenance staff often inexperienced and lack domain knowledge.

- As programs age, their structure degrades, making them harder to understand and change.

\*\*Maintenance Prediction:\*\*

- Concerned with assessing which parts of the system may cause problems and have high maintenance costs.

- Change acceptance depends on maintainability.

- Implementing changes degrades the system, reducing maintainability.

- Maintenance costs depend on the number of changes, and costs of change depend on maintainability.

\*\*Maintenance Prediction (continued):\*\*

- Diagram provided, illustrating the maintenance prediction process.

\*\*Change Prediction:\*\*

- Predicting the number of changes requires an understanding of the system-environment relationship.

- Tightly coupled systems require changes whenever the environment changes.

- Influencing factors: number and complexity of system interfaces, volatile requirements, and business processes.

\*\*Complexity Metrics:\*\*

- Predicting maintainability by assessing component complexity.

- Most maintenance effort spent on a small number of components.

- Complexity depends on control structures, data structures, object/method/module size.

\*\*Process Metrics:\*\*

- Used to assess maintainability.

- Metrics include requests for corrective maintenance, impact analysis time, implementation time, and outstanding change requests.

- Increasing metrics may indicate a decline in maintainability.

\*\*System Re-engineering:\*\*

- Restructuring or rewriting part or all of a legacy system without changing functionality.

- Applicable when some but not all subsystems require frequent maintenance.

- Involves adding effort to make subsystems easier to maintain.

\*\*Advantages of Reengineering:\*\*

- Reduced risk compared to new software development.

- Lower cost compared to developing new software.

\*\*Reengineering Process Activities:\*\*

- Source code translation, reverse engineering, program structure improvement, program modularization, and data reengineering.

- Diagram provided illustrating the reengineering process.

\*\*Reengineering Approaches:\*\*

- Different approaches based on source code transformation, reverse engineering, and program transformation.

\*\*Reengineering Cost Factors:\*\*

- Quality of software to be reengineered, tool support availability, extent of required data conversion, and availability of expert staff.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 41 – 57

\*\*Preventative Maintenance by Refactoring:\*\*

- Refactoring is making improvements to a program to slow down degradation through change.

- Considered as 'preventative maintenance' to reduce future change problems.

- Involves modifying a program to enhance structure, reduce complexity, and improve understandability.

- Concentrates on program improvement without adding functionality.

\*\*Refactoring and Reengineering:\*\*

- Reengineering occurs after a system has been maintained, using automated tools to create a more maintainable system.

- Refactoring is continuous improvement throughout development, avoiding structure and code degradation.

\*\*'Bad Smells' in Program Code:\*\*

- Identifies issues in program code:

- Duplicate code

- Long methods

- Switch (case) statements

- Data clumping

- Speculative generality

- Strategies to address these issues are provided.

\*\*Legacy System Management:\*\*

- Strategies for organizations relying on legacy systems:

- Scrap the system

- Continue maintaining

- Transform through reengineering

- Replace with a new system

- Strategy depends on system quality and business value.

\*\*Legacy System Categories:\*\*

- Categories based on quality and business value:

- Low quality, low business value (scrap)

- Low quality, high business value (reengineer/replace)

- High quality, low business value (replace/scrap/maintain)

- High quality, high business value (continue maintenance)

\*\*Business Value Assessment:\*\*

- Assessment considers different viewpoints from stakeholders.

- Issues in assessment include system use, supported business processes, system dependability, and system outputs.

\*\*System Quality Assessment:\*\*

- Assessment includes business process assessment, environment assessment, and application assessment.

- Business process assessment considers process model, adaptation, relationships, and legacy software support.

- Environmental assessment factors include supplier stability, failure rate, age, performance, support requirements, maintenance costs, interoperability, understandability, documentation, data, and programming language.

- Quantitative data may be collected to assess system quality.

\*\*Key Points:\*\*

- Three types of software maintenance: bug fixing, adapting to a new environment, and implementing new/changed requirements.

- Software reengineering involves restructuring and re-documenting to improve understandability.

- Refactoring, a form of preventative maintenance, involves making changes while preserving functionality.

- Assessing the business value and application quality helps decide whether to replace, transform, or maintain a legacy system.