Chapter-9
Software Evolution

Prepared by
Asst. Prof. Bal Krishna Subedi

Topics Covered

- Evolution Process;
- Legacy Systems;
- Software Maintenance

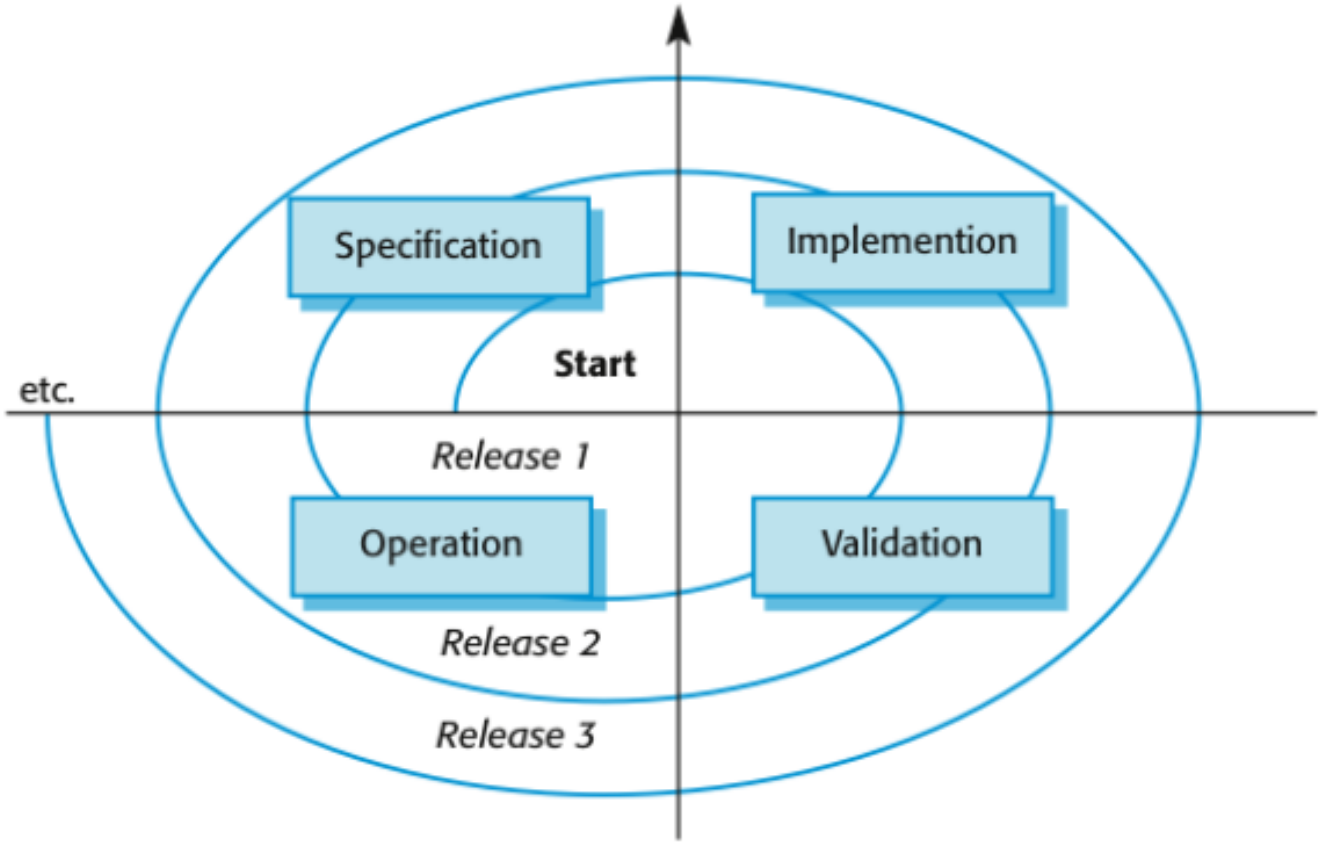
Software change

- Software change is inevitable
  - New requirements emerge when the software is used;
  - The business environment changes;
  - Errors must be repaired;
  - New computers and equipment is added to the system;
  - The performance or reliability of the system may have to be improved.
- A key problem for all organizations is implementing and managing change to their existing software systems.

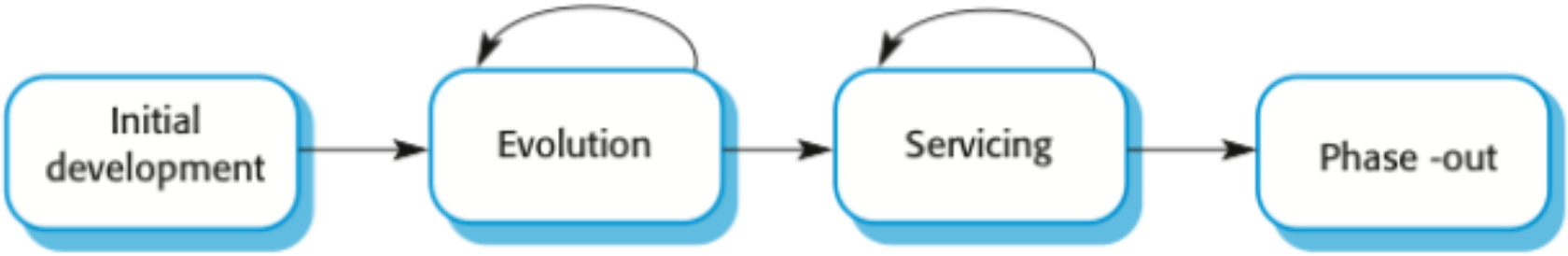
Importance of evolution

- Organisations have huge investments in their software systems - they are critical business assets.
- To maintain the value of these assets to the business, they must be changed and updated.
- The majority of the software budget in large companies is devoted to changing and evolving existing software rather than developing new software.

A spiral model of development and evolution



Evolution and servicing



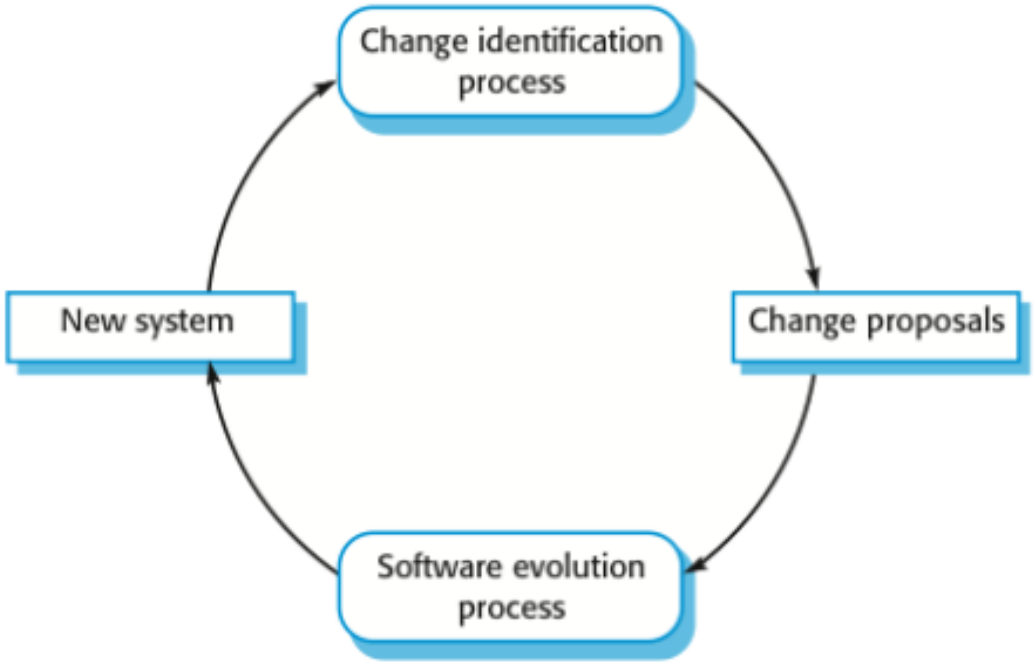
Evolution and servicing

- Evolution
  - The stage in a software system's life cycle where it is in operational use and is evolving as new requirements are proposed and implemented in the system.
- Servicing
  - At this stage, the software remains useful but the only changes made are those required to keep it operational i.e. bug fixes and changes to reflect changes in the software's environment. No new functionality is added.
- Phase-out
  - The software may still be used but no further changes are made to it.

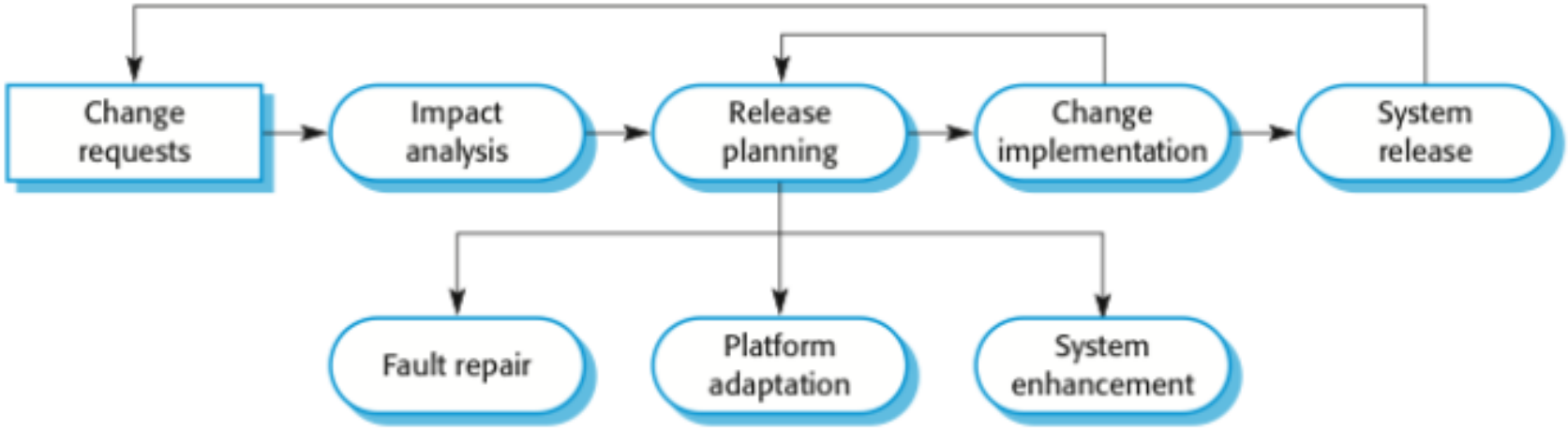
Evolution processes

- Software evolution processes depend on
  - The type of software being maintained;
  - The development processes used;
  - The skills and experience of the people involved.
- Proposals for change are the driver for system evolution.
  - Should be linked with components that are affected by the change, thus allowing the cost and impact of the change to be estimated.
- Change identification and evolution continues throughout the system lifetime.

Change identification and evolution processes



The software evolution process



Change implementation

Change implementation

- Iteration of the development process where the revisions to the system are designed, implemented and tested.
- A critical difference is that the first stage of change implementation may involve program understanding, especially if the original system developers are not responsible for the change implementation.
- During the program understanding phase, you have to understand how the program is structured, how it delivers functionality and how the proposed change might affect the program.

Urgent change requests

- Urgent changes may have to be implemented without going through all stages of the software engineering process
  - If a serious system fault has to be repaired to allow normal operation to continue;
  - If changes to the system's environment (e.g. an OS upgrade) have unexpected effects;
  - If there are business changes that require a very rapid response (e.g. the release of a competing product).

The emergency repair process

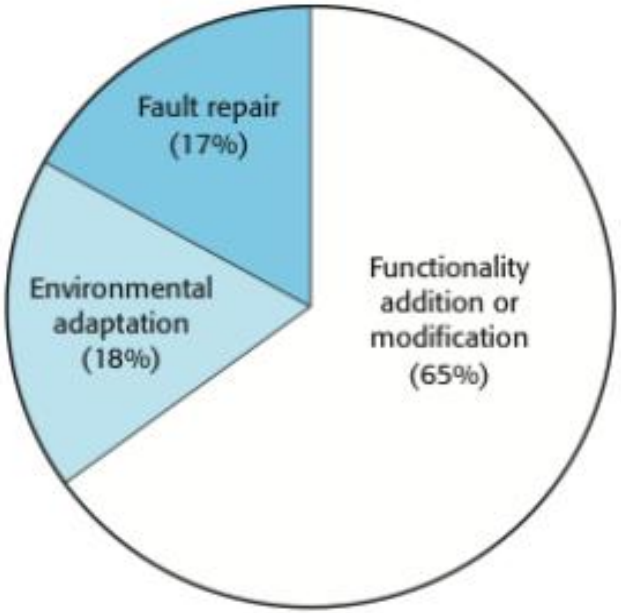
Software maintenance

- Modifying a program after it has been put into use.
- The term is mostly used for changing custom software. Generic software products are said to evolve to create new versions.
- Maintenance does not normally involve major changes to the system's architecture.
- Changes are implemented by modifying existing components and adding new components to the system.

Types of maintenance

- Maintenance to repair software faults
  - Changing a system to correct deficiencies in the way meets its requirements.
- Maintenance to adapt software to a different operating environment
  - Changing a system so that it operates in a different environment (computer, OS, etc.) from its initial implementation.
- Maintenance to add to or modify the system's functionality
  - Modifying the system to satisfy new requirements.

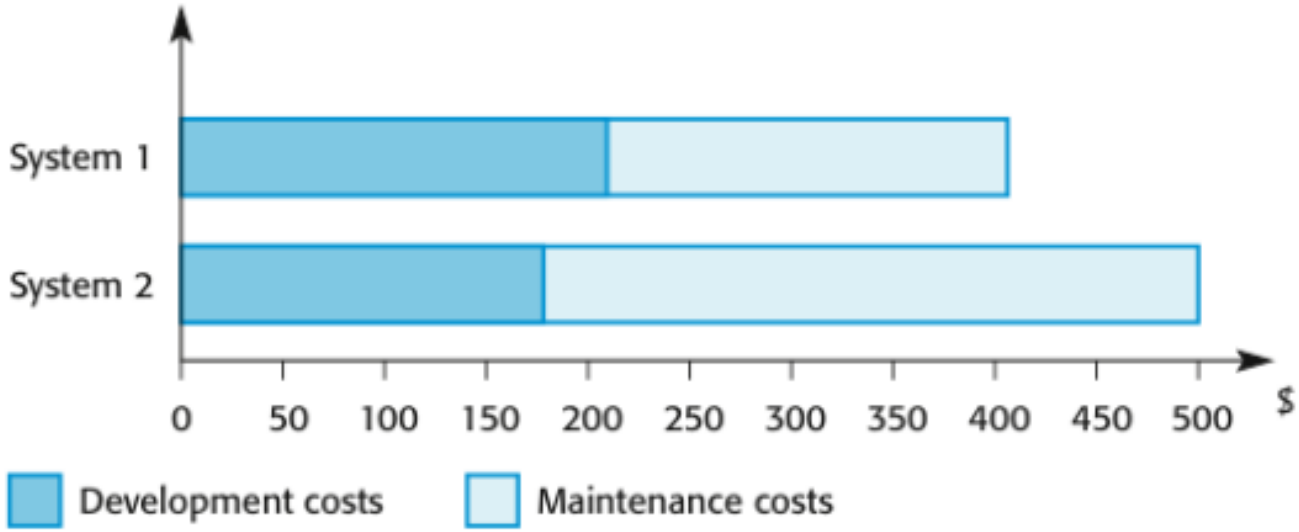
Figure 9.8 Maintenance effort distribution



Maintenance costs

- Usually greater than development costs (2* to 100* depending on the application).
- Affected by both technical and non-technical factors.
- Increases as software is maintained. Maintenance corrupts the software structure so makes further maintenance more difficult.
- Ageing software can have high support costs (e.g. old languages, compilers etc.).

Figure 9.9 Development and maintenance costs



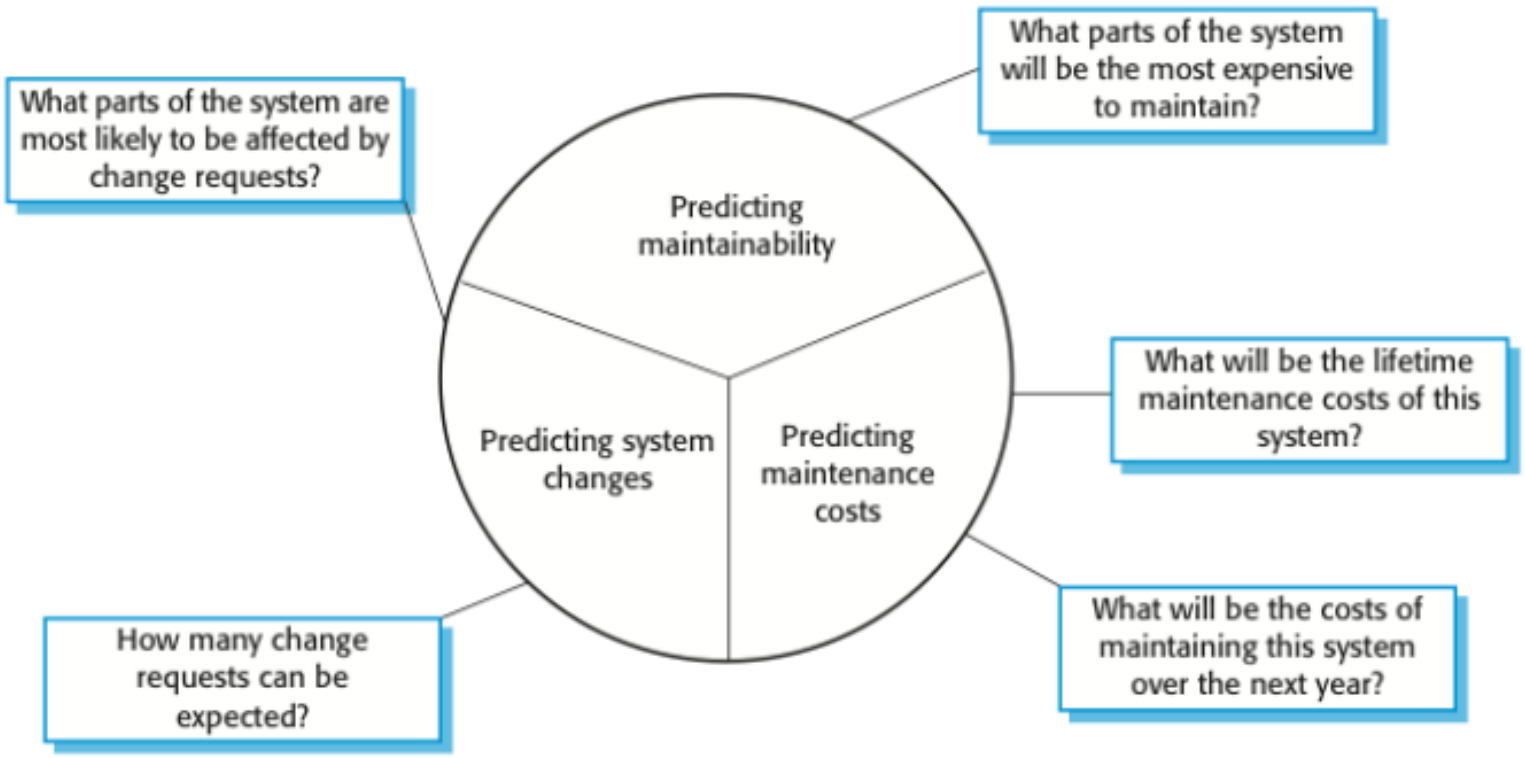
Maintenance cost factors

- Team stability
  - Maintenance costs are reduced if the same staff are involved with them for some time.
- Contractual responsibility
  - The developers of a system may have no contractual responsibility for maintenance so there is no incentive to design for future change.
- Staff skills
  - Maintenance staff are often inexperienced and have limited domain knowledge.
- Program age and structure
  - As programs age, their structure is degraded and they become harder to understand and change.

Maintenance prediction

- Maintenance prediction is concerned with assessing which parts of the system may cause problems and have high maintenance costs
  - 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 depend on maintainability.

Maintenance prediction



Change prediction

- Predicting the number of changes requires and understanding of the relationships between a system and its environment.
- Tightly coupled systems require changes whenever the environment is changed.
- Factors influencing this relationship are
  - Number and complexity of system interfaces;
  - Number of inherently volatile system requirements;
  - The business processes where the system is used.

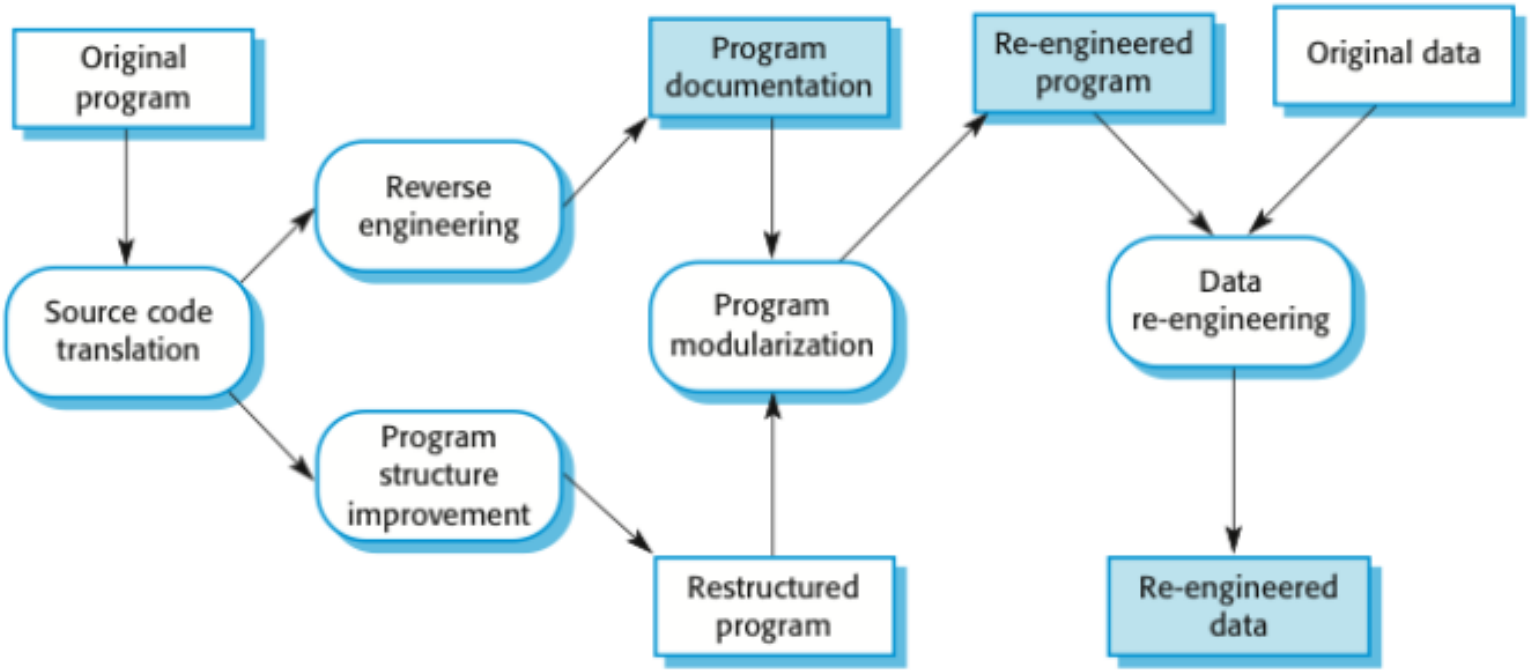
System re-engineering

- Re-structuring or re-writing part or all of a legacy system without changing its functionality.
- Applicable where some but not all sub-systems of a larger system require frequent maintenance.
- Re-engineering involves adding effort to make them easier to maintain. The system may be re-structured and re-documented.

Advantages of reengineering

- Reduced risk
  - There is a high risk in new software development. There may be development problems, staffing problems and specification problems.
- Reduced cost
  - The cost of re-engineering is often significantly less than the costs of developing new software.

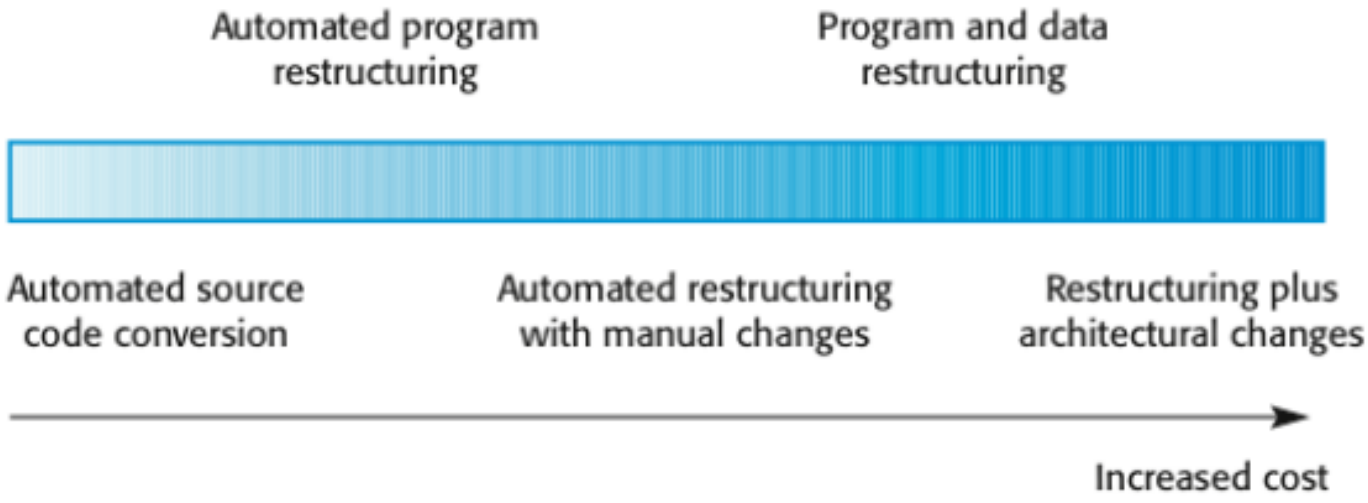
The reengineering process



Reengineering process activities

- Source code translation
  - Convert code to a new language.
- Reverse engineering
  - Analyse the program to understand it;
- Program structure improvement
  - Restructure automatically for understandability;
- Program modularisation
  - Reorganise the program structure;
- Data reengineering
  - Clean-up and restructure system data.

Reengineering approaches



Reengineering cost factors

- The quality of the software to be reengineered.
- The tool support available for reengineering.
- The extent of the data conversion which is required.
- The availability of expert staff for reengineering.
  - This can be a problem with old systems based on technology that is no longer widely used.

Refactoring and reengineering

- Re-engineering takes place after a system has been maintained for some time and maintenance costs are increasing. You use automated tools to process and re-engineer a legacy system to create a new system that is more maintainable.
- Refactoring is a continuous process of improvement throughout the development and evolution process. It is intended to avoid the structure and code degradation that increases the costs and difficulties of maintaining a system.

Legacy system management

- Organisations that rely on legacy systems must choose a strategy for evolving these systems
  - Scrap the system completely and modify business processes so that it is no longer required;
  - Continue maintaining the system;
  - Transform the system by re-engineering to improve its maintainability;
  - Replace the system with a new system.
- The strategy chosen should depend on the system quality and its business value.

Legacy system categories

- Low quality, low business value
  - These systems should be scrapped.
- Low-quality, high-business value
  - These make an important business contribution but are expensive to maintain. Should be re-engineered or replaced if a suitable system is available.
- High-quality, low-business value
  - Replace with COTS, scrap completely or maintain.
- High-quality, high business value
  - Continue in operation using normal system maintenance.

Key points

- Software development and evolution can be thought of as an integrated, iterative process that can be represented using a spiral model.
- For custom systems, the costs of software maintenance usually exceed the software development costs.
- The process of software evolution is driven by requests for changes and includes change impact analysis, release planning and change implementation.
- Lehman's laws, such as the notion that change is continuous, describe a number of insights derived from long-term studies of system evolution.

Key points

- There are 3 types of software maintenance, namely bug fixing, modifying software to work in a new environment, and implementing new or changed requirements.
- Software re-engineering is concerned with re-structuring and re-documenting software to make it easier to understand and change.
- Refactoring, making program changes that preserve functionality, is a form of preventative maintenance.
- The business value of a legacy system and the quality of the application should be assessed to help decide if a system should be replaced, transformed or maintained.

THANK YOU
ANY QUERY?