Lecture 14 - Software Evolution

SE 317 - Principles of Software Engineering

Material from : Software Engineering, 9th ed., Sommerville, Pearson
Object-Oriented Software Engineering, Stephen R. Schach, McGraw-Hill

Software Evolution

- Any system has to change after being delivered to the client if it is to remain useful
- Also called postdelivery maintenance

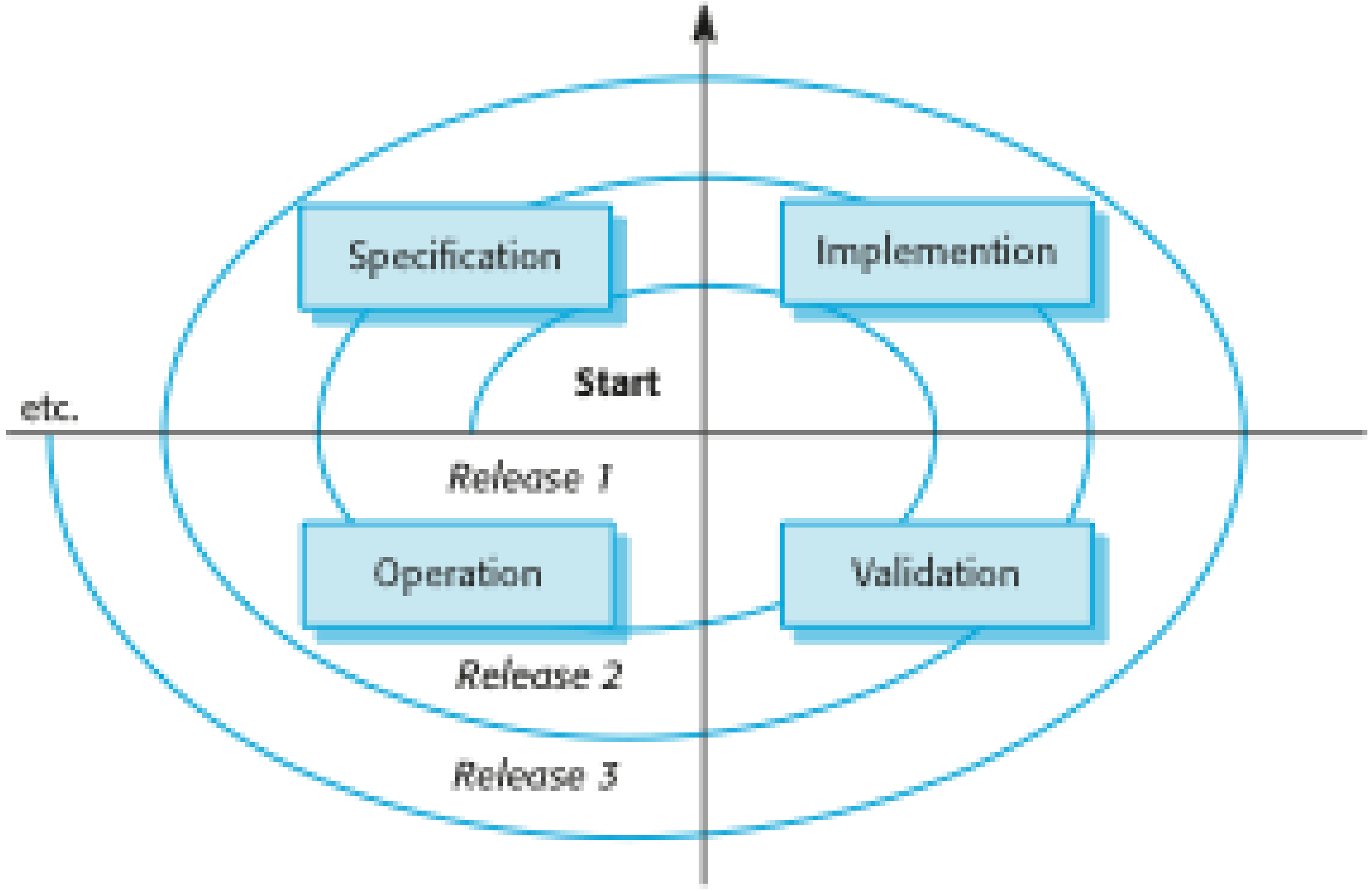
Software Evolution

- May be triggered by
  - Changing business requirements (perfective maintenance)
  - Reports of software defects (corrective maintenance)
  - Changes to other systems in a software system's environment (adaptive maintenance)

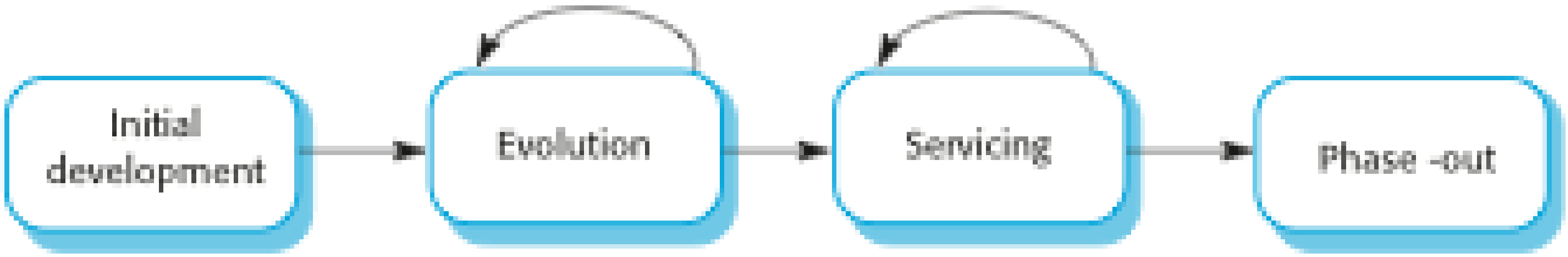
Software Evolution

- Useful software systems often have a very long lifetime
- Large military or infrastructure systems, such as air traffic control systems, may have a lifetime of 30 or more years
- Business systems are often more than 10 years old

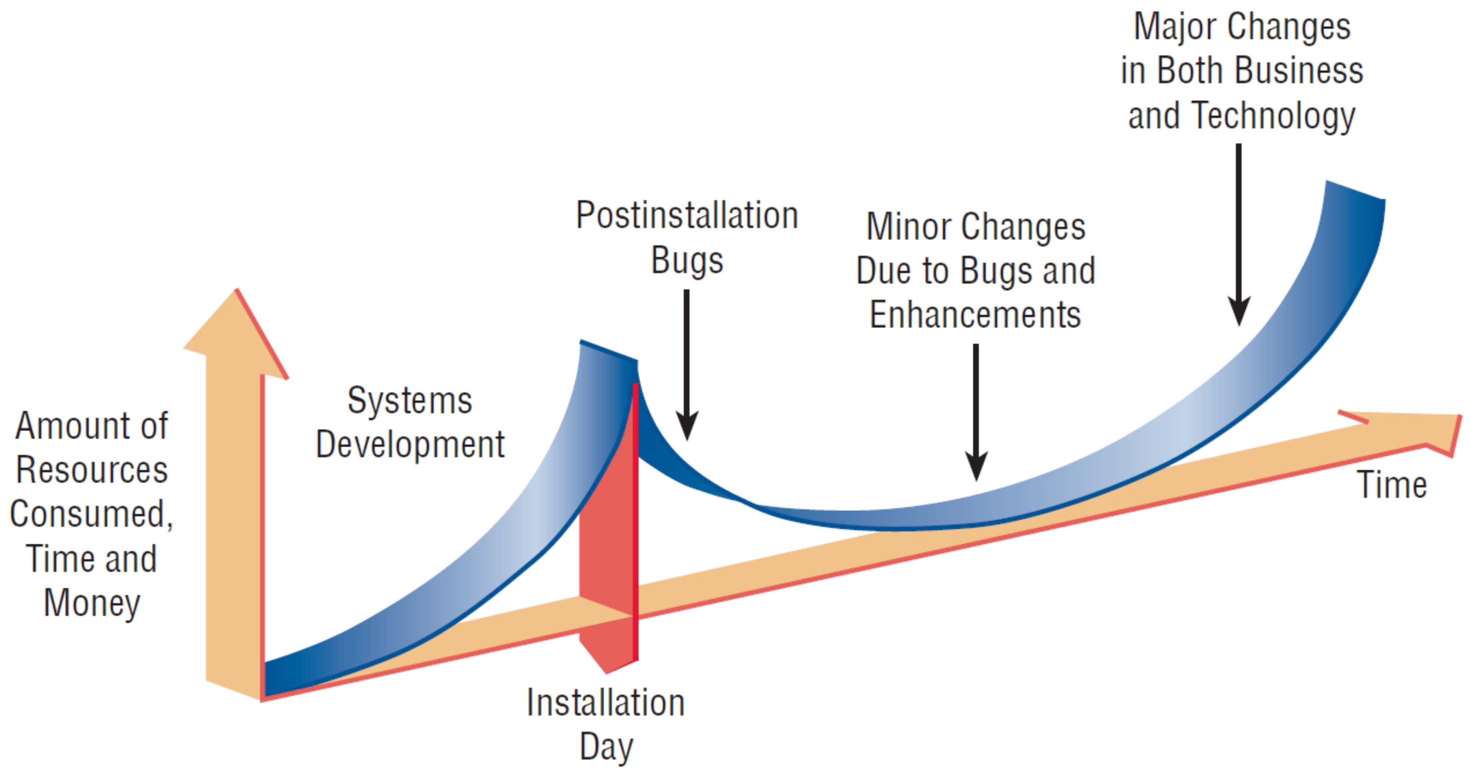
A spiral model of development and evolution



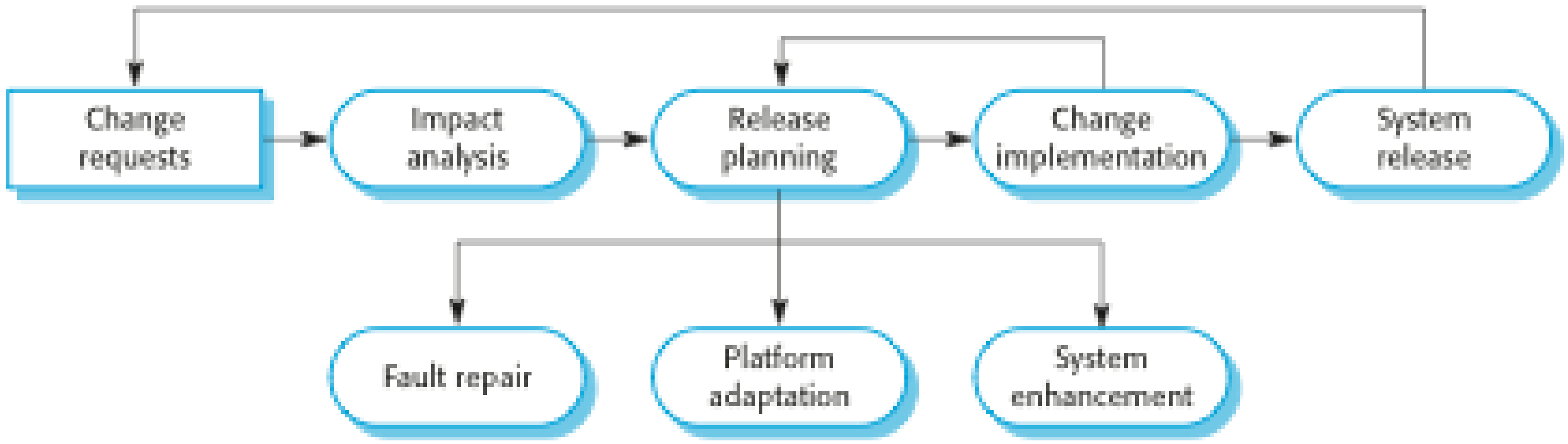
Evolution and servicing



Resource Consumption over the System Life


Major Changes in Both Business and Technology
Postinstallation Bugs
Minor Changes Due to Bugs and Enhancements
Systems Development
Amount of Resources Consumed, Time and Money
Time
Installation Day

The software evolution 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

Software maintenance

- 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

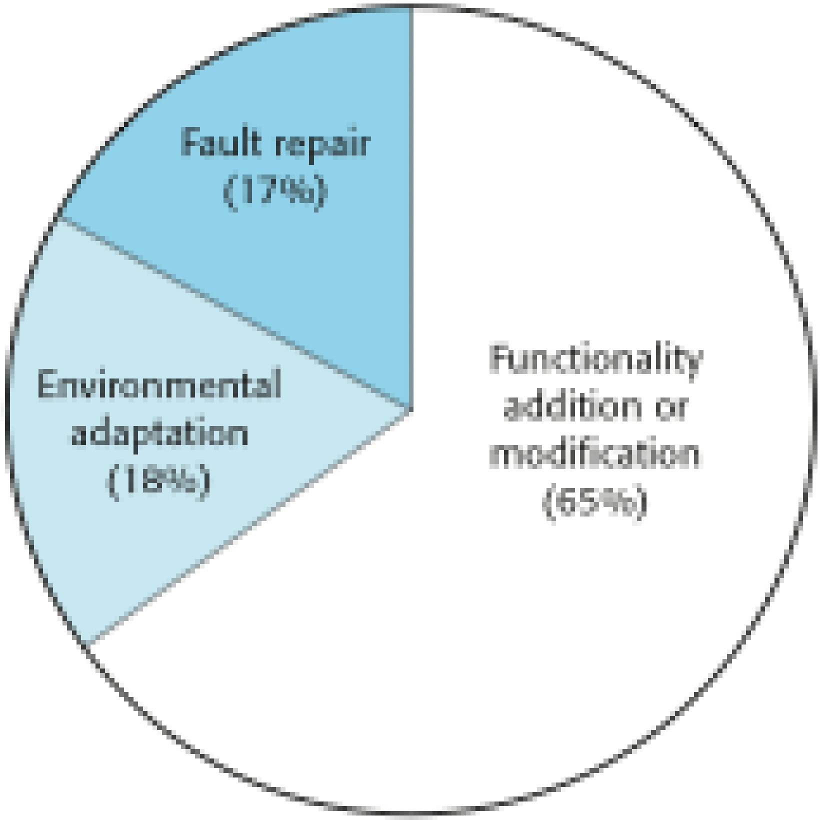
Types of maintenance

- 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

Types of maintenance

- Maintenance to add to or modify the system's functionality
  - Modifying the system to satisfy new requirements

Figure 9.8 Maintenance effort distribution



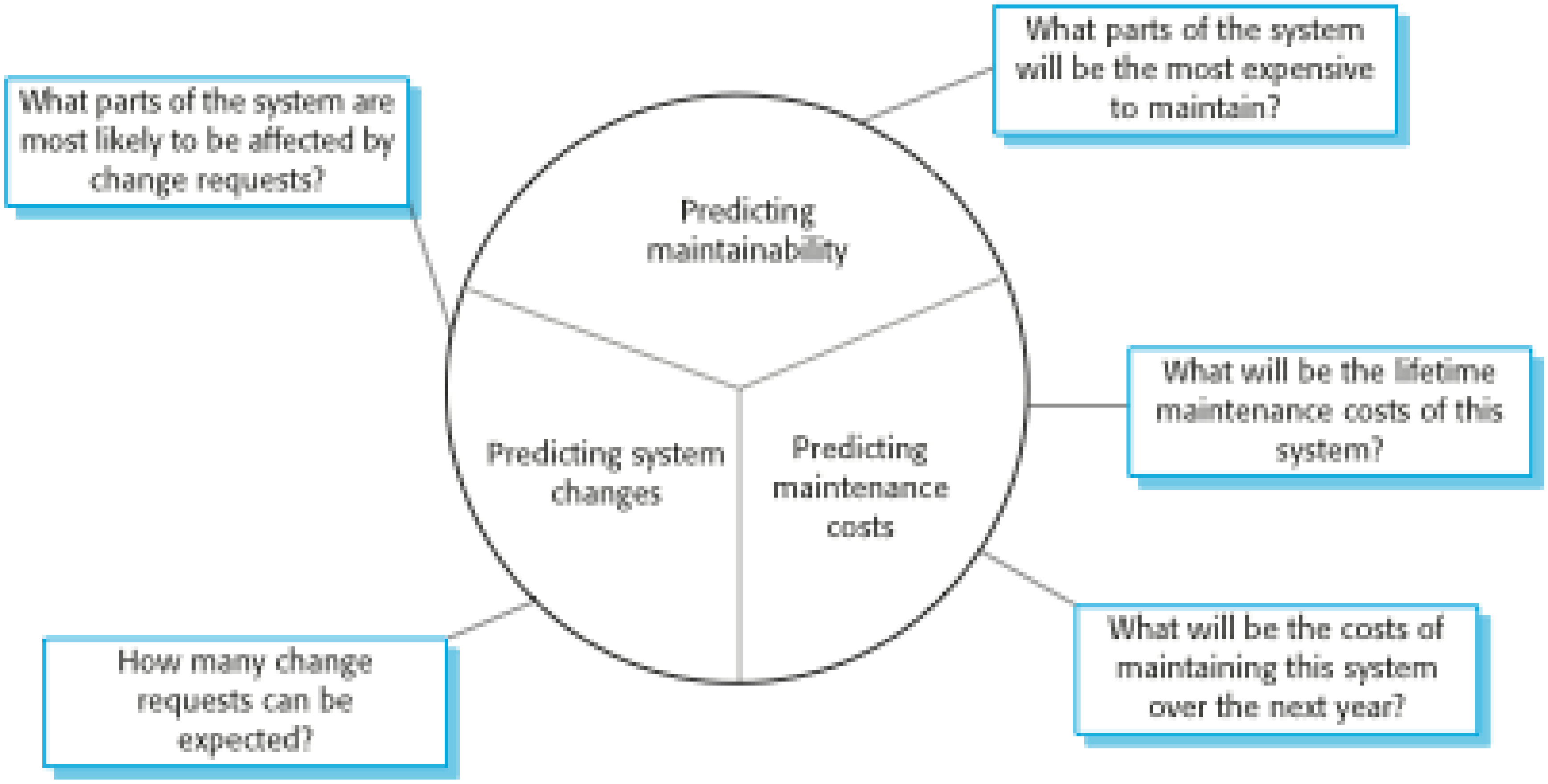
Maintenance costs

- Usually greater than development costs (2* to 100* depending on the application)
- Affected by both technical and non-technical factors

Maintenance costs

- 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.)

Maintenance prediction


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

System re-engineering

- Re-structuring or re-writing part or all of a legacy system without changing its functionality

System re-engineering

- 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

Preventative maintenance by refactoring

- Refactoring is the process of making improvements to a program to slow down degradation through change

Preventative maintenance by refactoring

- You can think of refactoring as ‘preventative maintenance’ that reduces the problems of future change

Refactoring and reengineering

- Re-engineering takes place after a system has been maintained for some time and maintenance costs are increasing

Refactoring and reengineering

- You use automated tools to process and re-engineer a legacy system to create a new system that is more maintainable

Refactoring and reengineering

- Refactoring is a continuous process of improvement throughout the development and evolution process

Refactoring and reengineering

- It is intended to avoid the structure and code degradation that increases the costs and difficulties of maintaining a system

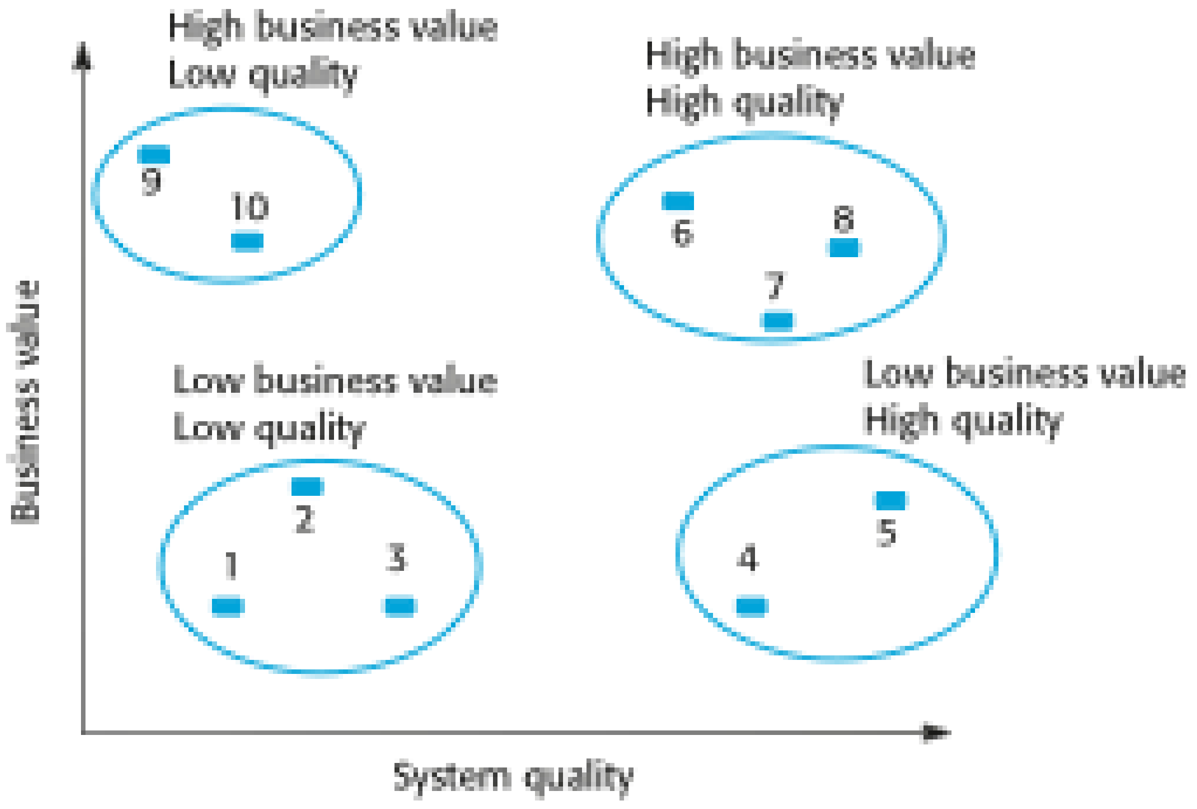
Legacy system management

- 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

Legacy system management

- The strategy chosen should depend on the system quality and its business value

Figure 9.13 An example of a legacy system assessment



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

Legacy system categories

- High-quality, low-business value
  - Replace with COTS, scrap completely or maintain
- High-quality, high business value
  - Continue in operation using normal system maintenance