1. Handover Problems
2. Refactor/ReEngineer

Chapter 9 - Software Evolution

MI Qing (Lecturer)

Telephone: 15210503242

Email: miqing@bjut.edu.cn

Office: 410, Information Building

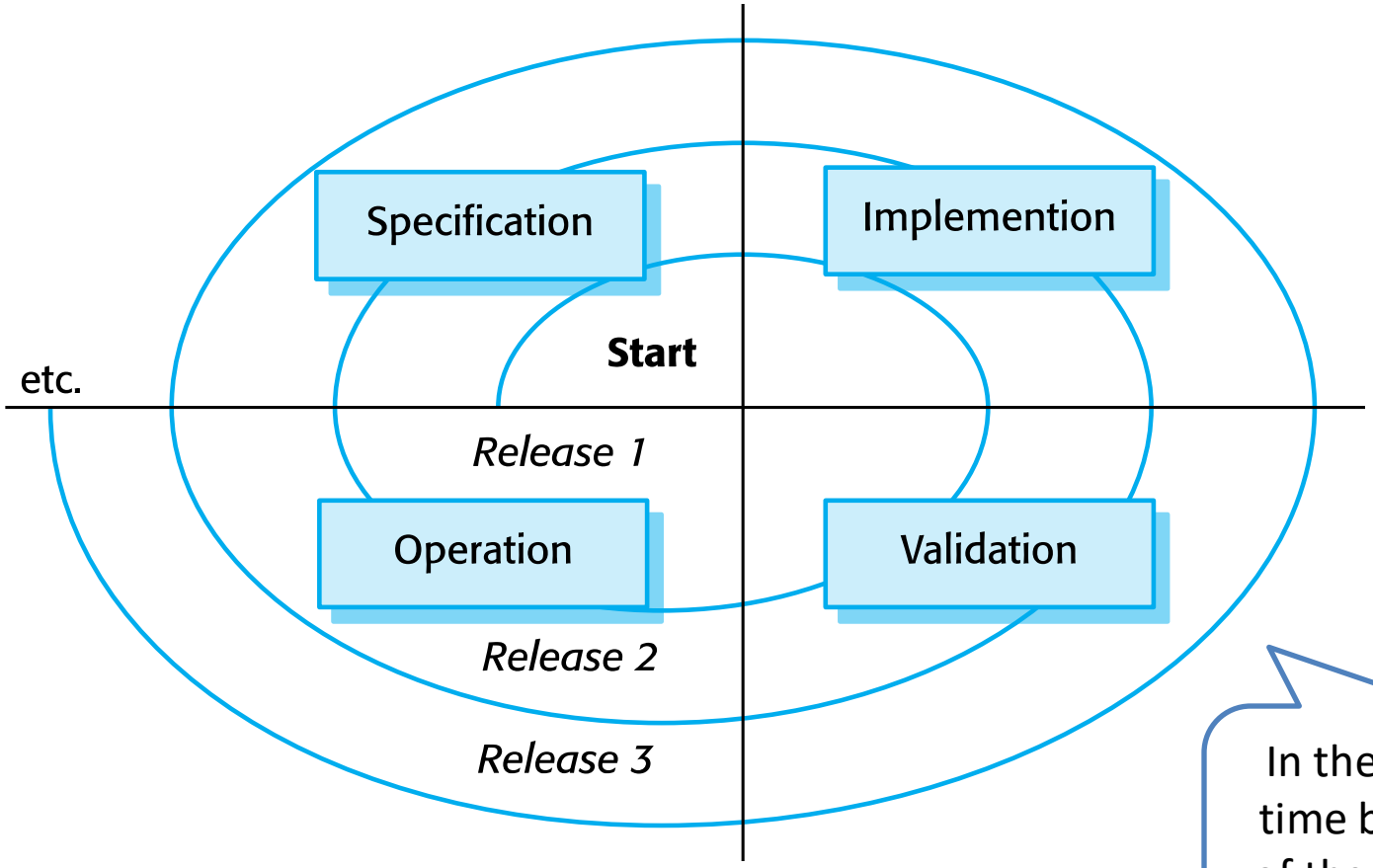
Topics Covered

- ✧ Evolution processes
- ✧ Legacy systems
- ✧ Software maintenance

Software Change

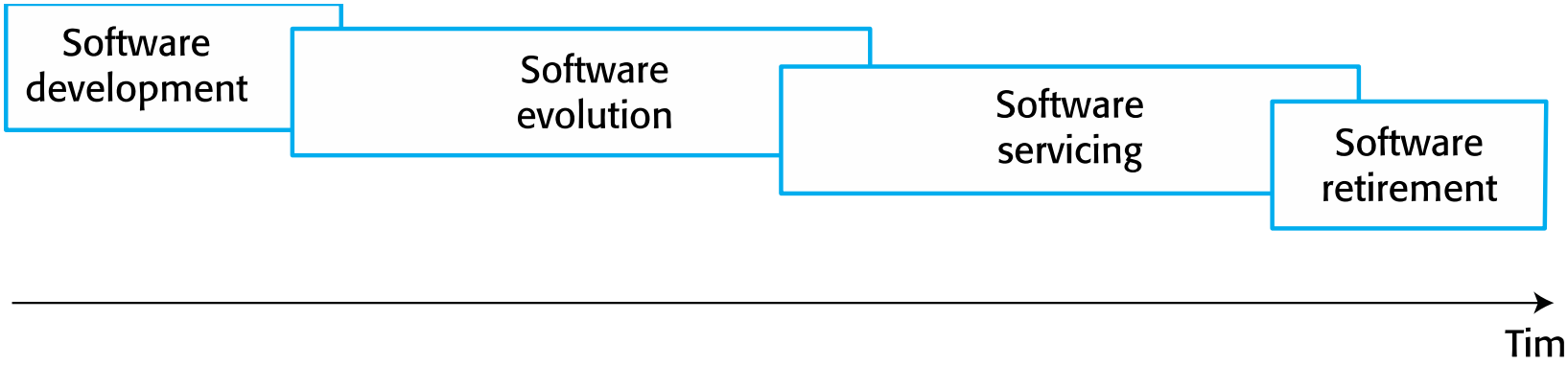
- Software change is inevitable:
  - New requirements emerge when the software is used;
  - The business environment changes;
  - Errors must be repaired;
  - New computers and equipment is added to the system;
  - The performance or reliability of the system may have to be improved.
- A key problem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

Spiral Model of Development and Evolution



In the last 10 years, the time between iterations of the spiral has reduced dramatically.

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 (e.g. bug fixes). No new functionality is added.

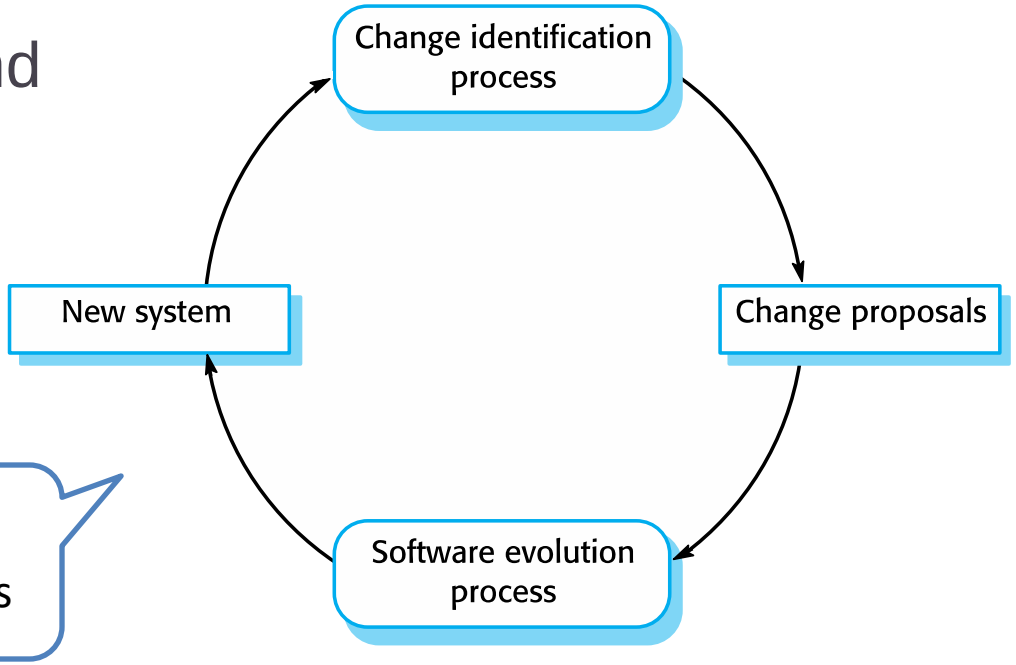
Retirement

The software may still be used but no further changes are made to it.

Evolution Processes

Evolution Processes

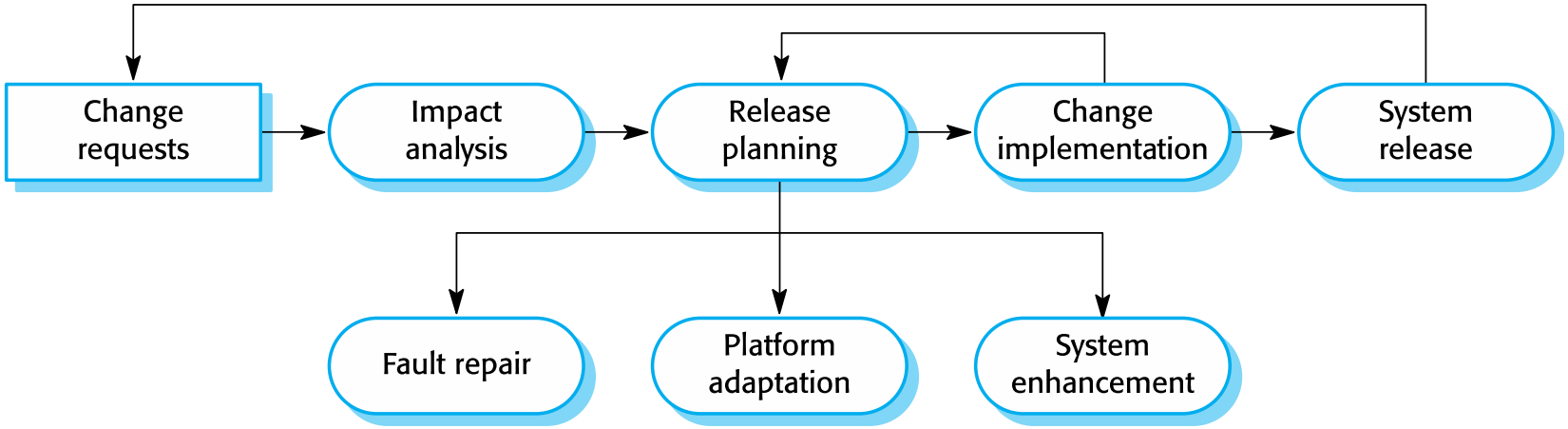
- Proposals for change are the driver for system evolution.
  - Should be linked with components that are affected by the change, thus allowing the cost and impact of the change to be estimated.
- Change identification and evolution continues throughout the system lifetime.



Change identification and evolution processes

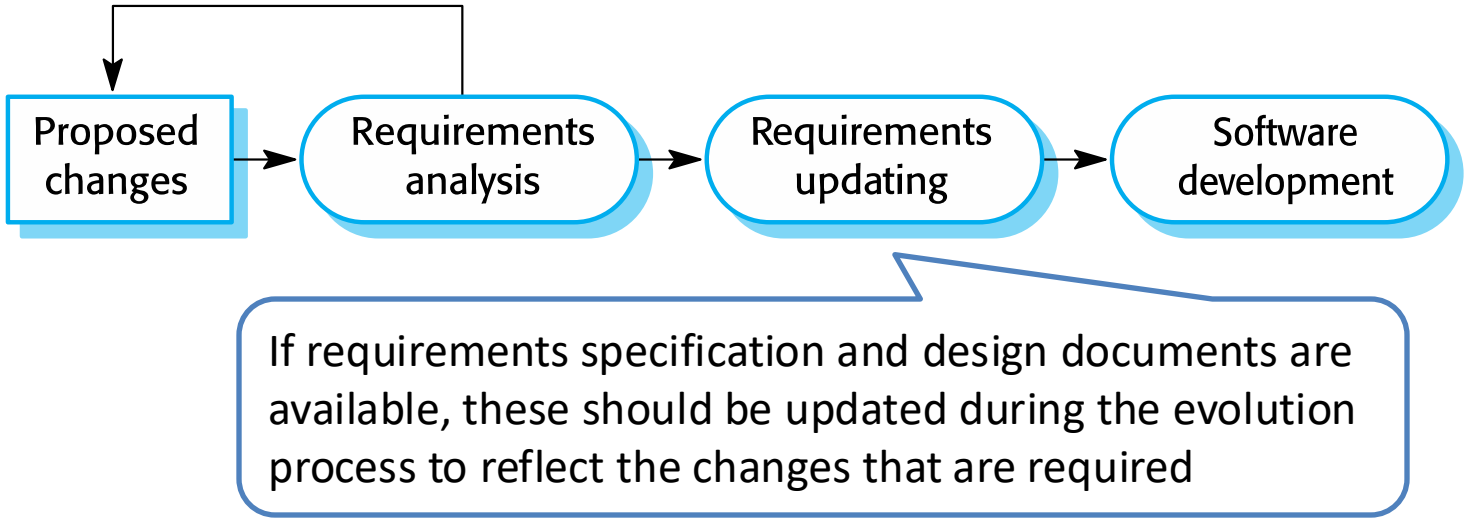
Software Evolution Process

- If the proposed changes are accepted, a new release of the system is planned.
  - During release planning, all proposed changes (fault repair, adaptation, and new functionality) are considered.
  - A decision is then made on which changes to implement in the next version of the system. The changes are implemented and validated, and a new version of the system is released.



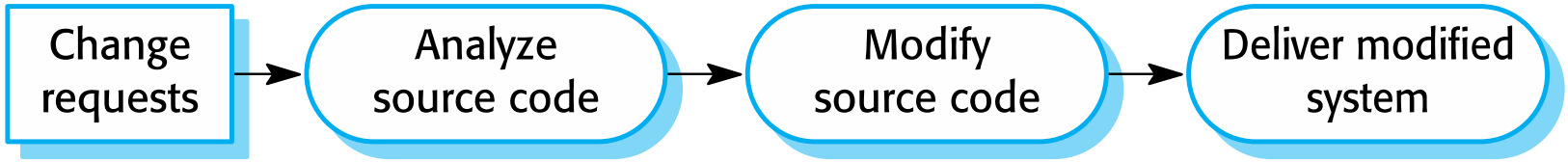
Change Implementation

- In situations where development and evolution are integrated, change implementation is simply an iteration of the development process.
  - The only difference between initial development and evolution is that customer feedback after delivery has to be considered when planning new releases of an application.



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).



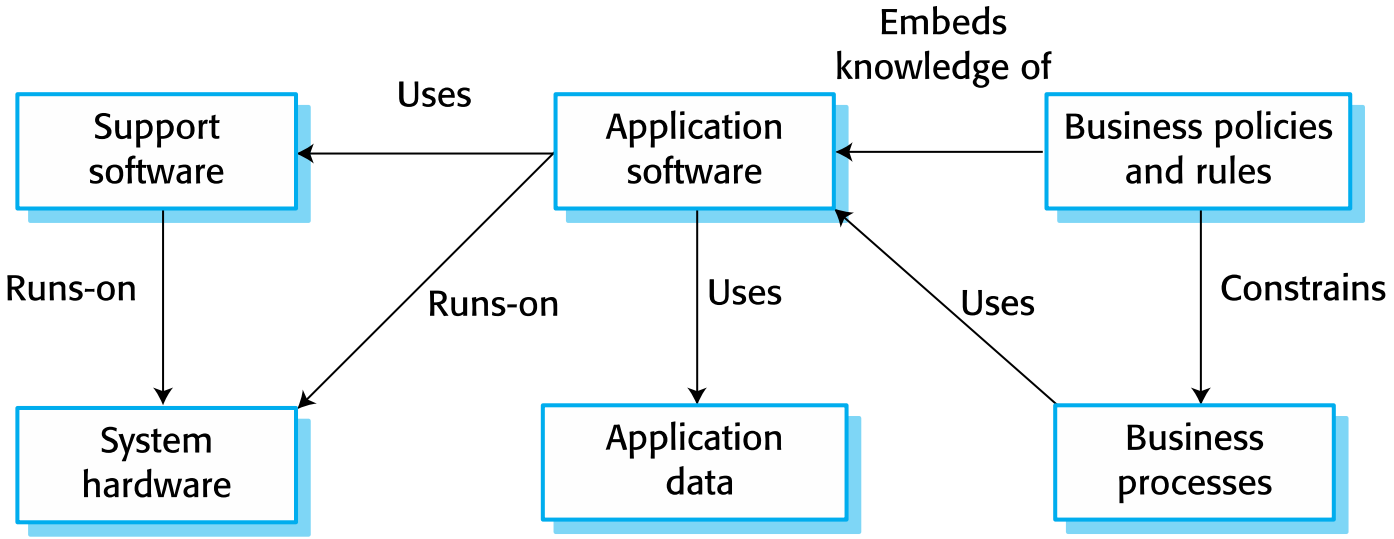
Handover Problems

- Where the development team have used an agile approach but the evolution team is unfamiliar with agile methods and prefer a plan-based approach.
  - The evolution team may expect detailed documentation to support evolution and this is not produced in agile processes.
- Where a plan-based approach has been used for development but the evolution team prefer to use agile methods.
  - The evolution team may have to start from scratch developing automated tests and the code in the system may not have been refactored and simplified as is expected in agile development.

Legacy Systems

Legacy Systems

- Legacy systems are older systems that rely on languages and technology that are no longer used for new systems development.
  - Legacy systems are not just software systems but are broader socio-technical systems that include hardware, software, libraries and other supporting software and business processes.



Legacy System Components

- System hardware
  - Legacy systems may have been written for hardware that is no longer available.
- Support software
  - The legacy system may rely on a range of support software, which may be obsolete or unsupported.
- Application software
  - The application system that provides the business services is usually made up of a number of application programs.

Legacy System Components

✧ Application data

- These are data processed by the application system. They may be inconsistent, duplicated or held in different databases.

✧ Business processes

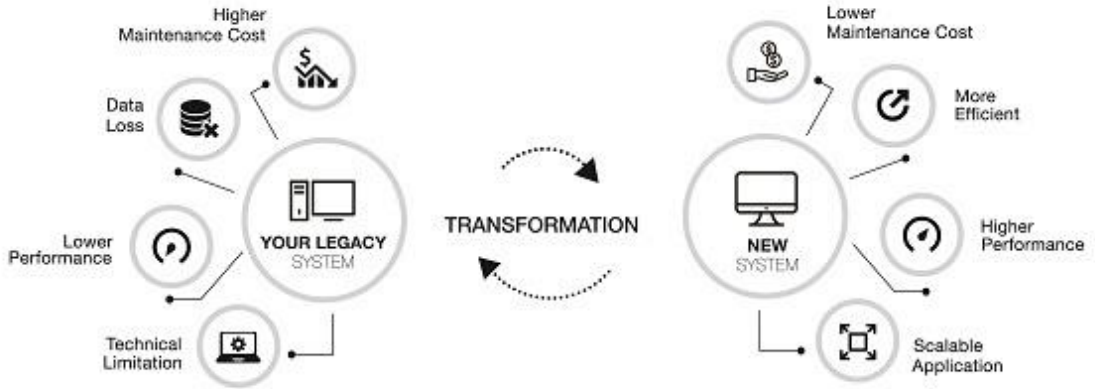
- These are processes that are used in the business to achieve some business objective.
- Business processes may be designed around a legacy system and constrained by the functionality that it provides.

✧ Business policies and rules

- These are definitions of how the business should be carried out and constraints on the business. Use of the legacy application system may be embedded in these policies and rules.

Legacy System Replacement

- Legacy system replacement is risky and expensive so businesses continue to use these systems.
- System replacement is risky for a number of reasons:
  - Lack of complete system specification
  - Tight integration of system and business processes
  - Undocumented business rules embedded in the legacy system
  - New software development may be late and/or over budget



Legacy System Change

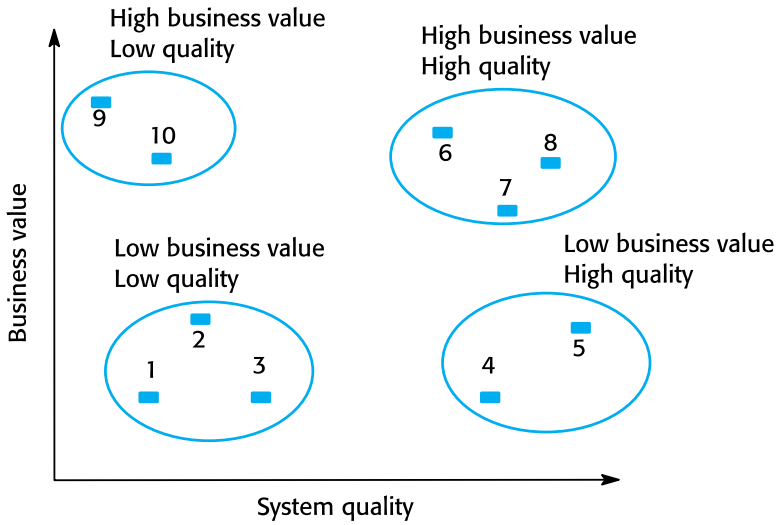
- Legacy systems are expensive to change for a number of reasons:
  - No consistent programming style
  - Use of obsolete programming languages with few people available with these language skills
  - Inadequate system documentation
  - System structure degradation
  - Program optimizations may make them hard to understand
  - Data errors, duplication and inconsistency
- At some stage, the costs of managing and maintaining the legacy system become so high that it has to be replaced with a new 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
  - Scrap completely or maintain.
- High-quality, high business value
  - Continue in operation using normal system maintenance.



Business Value Assessment

- The use of the system
  - If systems are only used occasionally or by a small number of people, they may have a low business value.
- The business processes that are supported
  - A system may have a low business value if it forces the use of inefficient business processes.
- System dependability
  - If a system is not dependable and the problems directly affect business customers, the system has a low business value.
- The system outputs
  - If the business depends on system outputs, then the system has a high business value.

System Quality Assessment

Environment assessment

How effective is the system's environment and how expensive is it to maintain?

Application assessment

What is the quality of the application software system?

Factors used in environment assessment

Factor	Questions
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 system restarts?
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 there are high costs associated with this support, it may be worth considering system replacement.
Maintenance costs	What are the costs of hardware maintenance and support software licences? 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? Is hardware emulation required?

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?

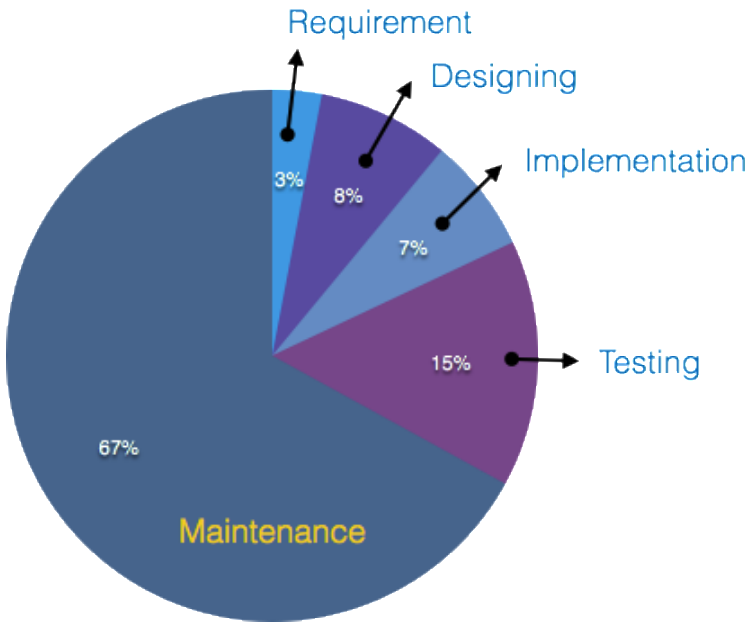
System Measurement

- You may collect quantitative data to make an assessment of the quality of the application system:
  - The number of system change requests
    - The higher this accumulated value, the lower the quality of the system.
  - The number of different user interfaces used by the system
    - The more interfaces, the more likely it is that there will be inconsistencies and redundancies in these interfaces.
  - The volume of data used by the system
    - As the volume of data (number of files, size of database, etc.) processed by the system increases, so too do the inconsistencies and errors in that data.

Software Maintenance

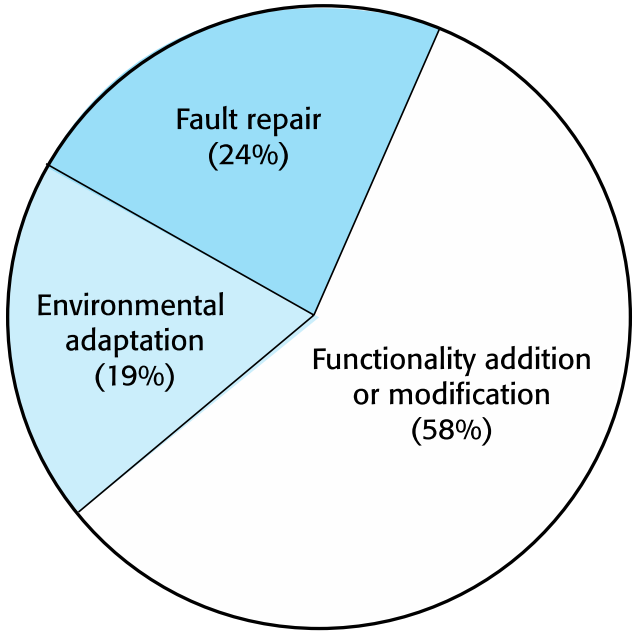
Software Maintenance

- Modifying a program after it has been put into use.
- 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

- Fault repairs
  - Changing a system to fix bugs/vulnerabilities and correct deficiencies in the way meets its requirements.
- Environmental adaptation
  - Changing a system so that it operates in a different environment (computer, OS, etc.) from its initial implementation.
- Functionality addition and modification
  - Modifying the system to satisfy new requirements.

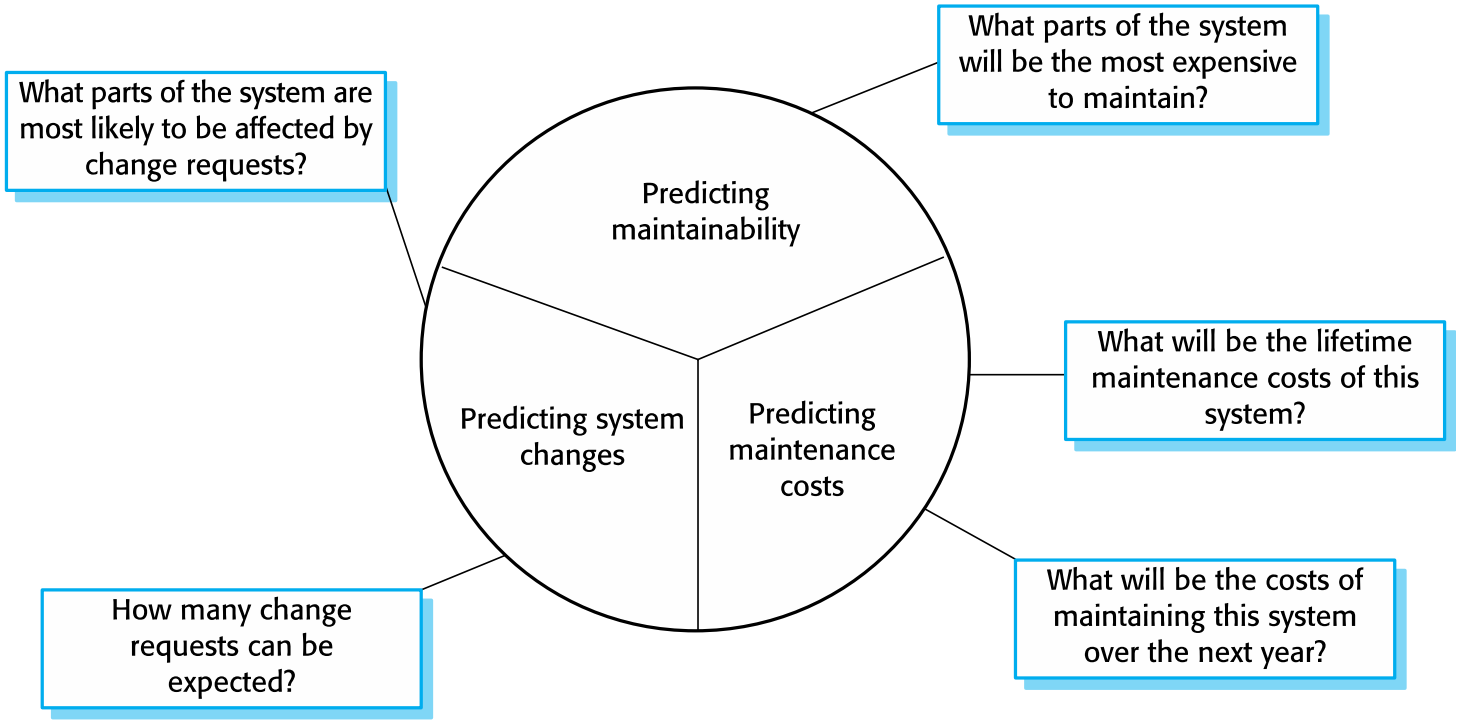


Maintenance Costs

- It is usually more expensive to add new features to a system during maintenance than it is to add the same features during development.
  - A new team has to understand the programs being maintained.
  - Separating maintenance and development means there is no incentive for the development team to write maintainable software.
  - Program maintenance work is unpopular. Maintenance staff are often inexperienced and have limited domain knowledge.
  - As programs age, their structure degrades and they become harder to change.

Maintenance Prediction

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



Complexity Metrics

- Predictions of maintainability can be made by assessing the complexity of system components.
- Studies have shown that most maintenance effort is spent on a relatively small number of system components.
- Complexity depends on:
  - Complexity of control structures;
  - Complexity of data structures;
  - Object, method (procedure) and module size.

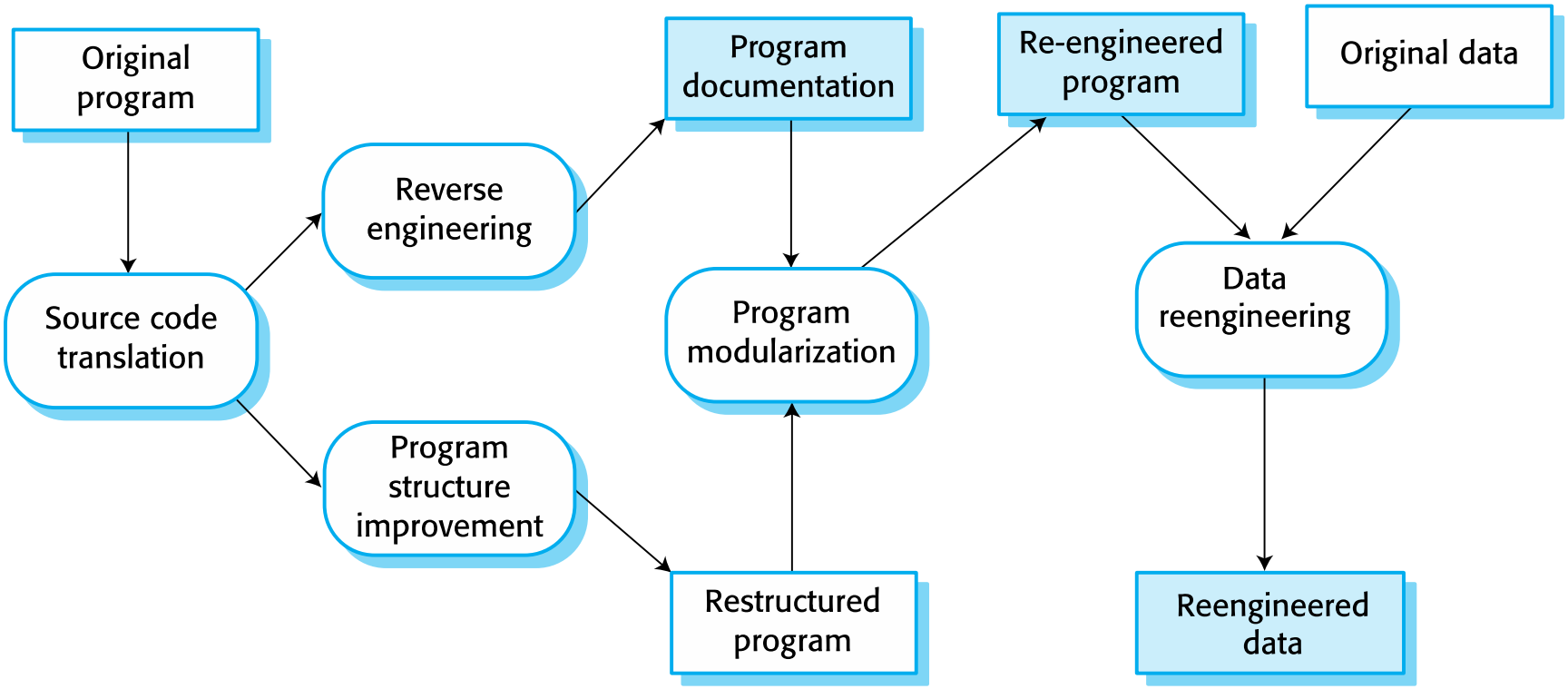
Process Metrics

- After a system has been put into service, you may be able to use process data to help predict maintainability. Examples of process metrics that can be used for assessing maintainability are:
  - Number of requests for corrective maintenance;
  - Average time required for impact analysis;
  - Average time taken to implement a change request;
  - Number of outstanding change requests.
- If any or all of these is increasing, this may indicate a decline in maintainability.

Software Reengineering

- Restructuring or rewriting 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.
  - Reengineering involves adding effort to make them easier to maintain.
- Advantages:
  - 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.

Reengineering Process

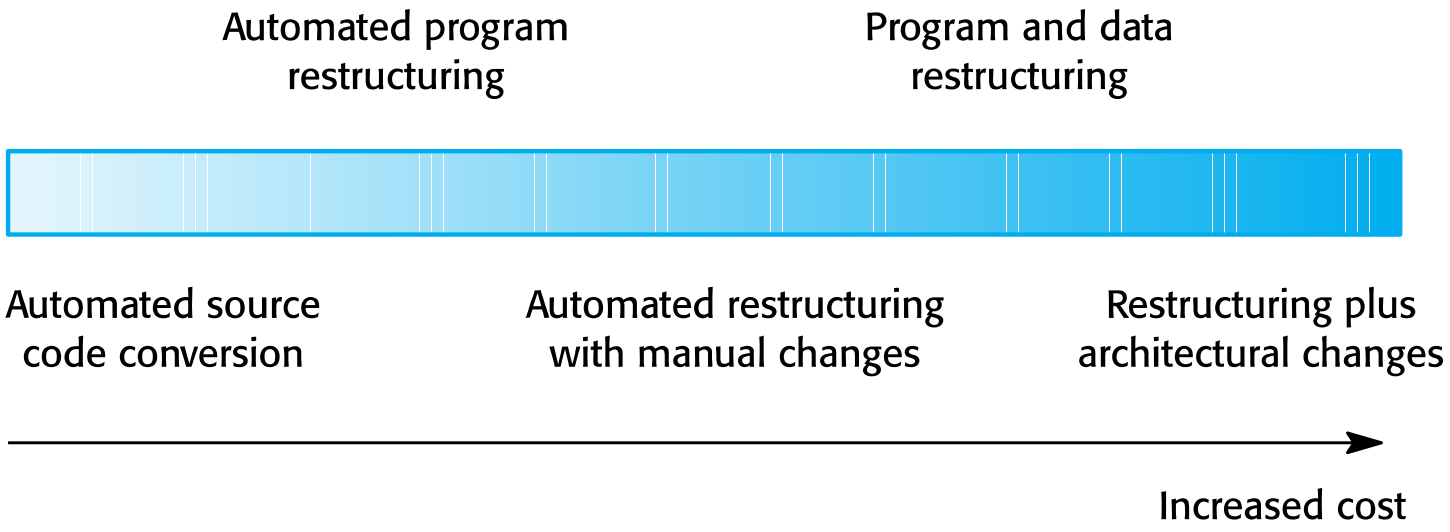


Reengineering Process Activities

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

Reengineering Approaches

- The costs of reengineering obviously depend on the extent of the work that is carried out. There is a spectrum of possible approaches to reengineering. Costs increase from left to right.



Refactoring

- Refactoring is the process of making improvements to a program to slow down degradation through change.
- You can think of refactoring as ‘preventative maintenance’ that reduces the problems of future change.
- Refactoring involves modifying a program to improve its structure, reduce its complexity or make it easier to understand.
- When you refactor a program, you should not add functionality but rather concentrate on program improvement.

Refactoring VS 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.

Summary

Key Points

- Software development and evolution can be thought of as an integrated, iterative process that can be represented using a spiral model.
- The process of software evolution is driven by requests for changes and includes change impact analysis, release planning and change implementation.
- Legacy systems are older software systems, developed using obsolete software and hardware technologies, that remain useful for a business.
  - 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.

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.