Chapter 9 – Software Evolution

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.

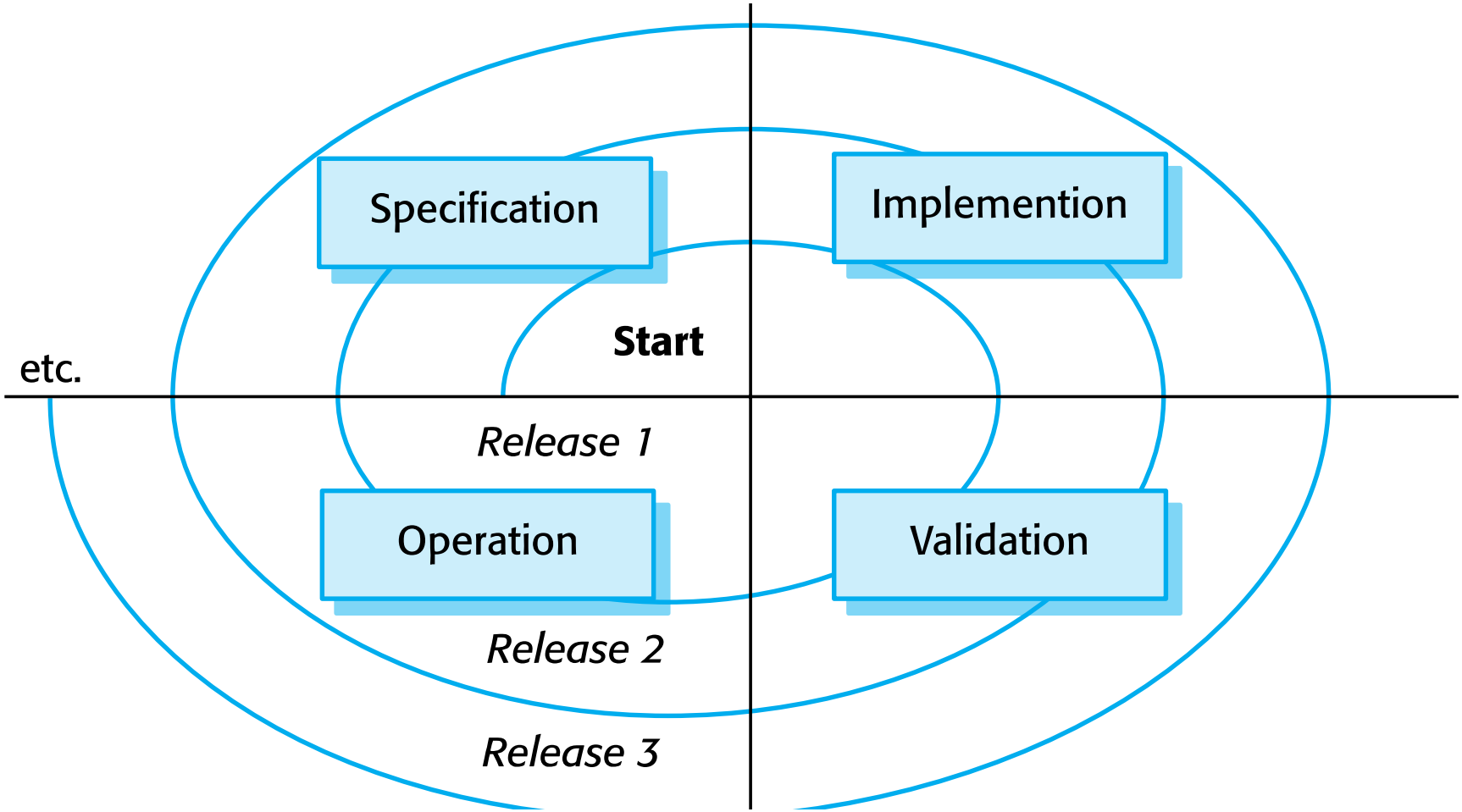
Importance of evolution

- 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.
  - An informal industry poll suggests that 85~90% of organizational software costs are evolution costs.
  - Other surveys suggest that 2/3 software costs are evolution costs.

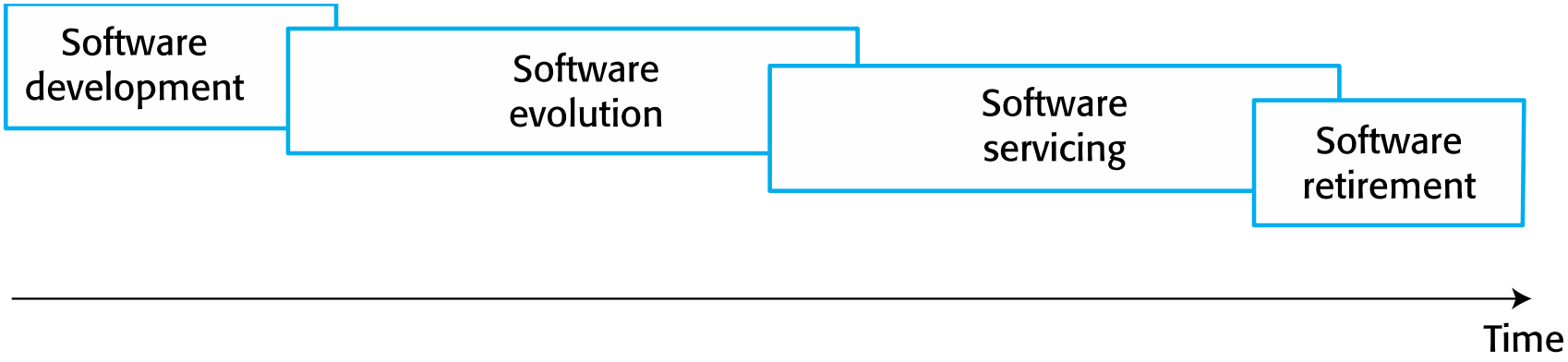
Importance of evolution

- The evolution of a system can rarely be considered in isolation
  - Changes to the environment lead to system change that may then trigger further environmental changes
- The evolution of software
  - For most packaged software products, a single organization is responsible for both the initial software development and evolution of the software
  - For customer software, a different approach is commonly used. A software company develops software for a customer and the customer's own development staff then take over and are responsible for the software evolution
- When the transition from development to evolution is **not seamless**, the process of changing the software after delivery is often called '**software maintenance**'

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 reaches a transition point where significant changes, implementing new requirements, become less and less cost effective), 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

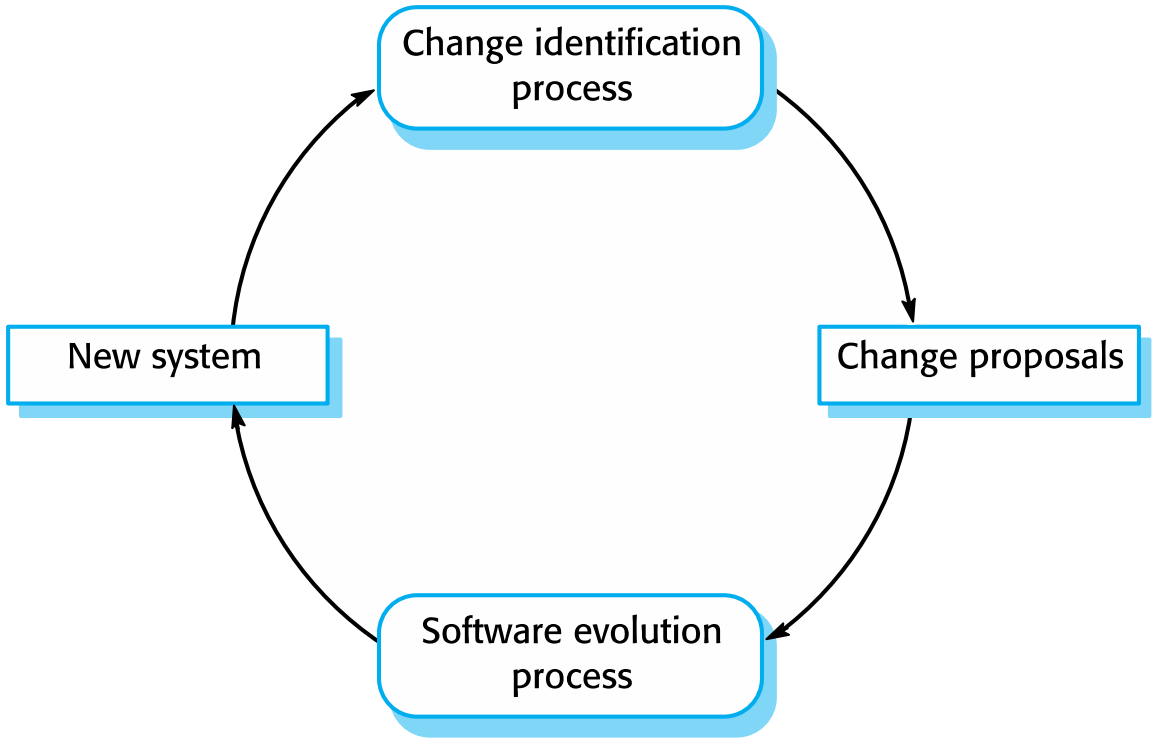
- The software may **still be used but no further changes** are made to it. Users have to work around any problem that they discover.

Evolution processes

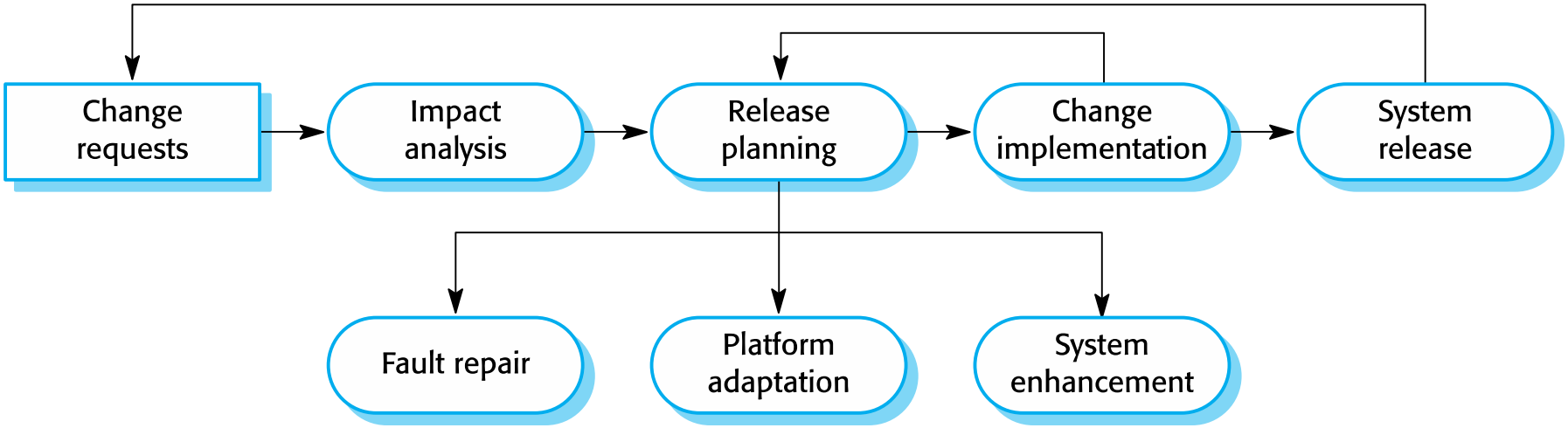
Evolution processes

- 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



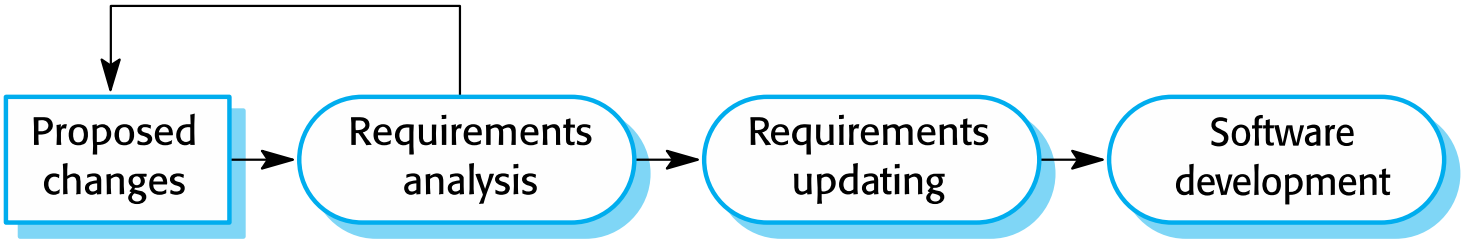
The software evolution process



Change implementation

- 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.

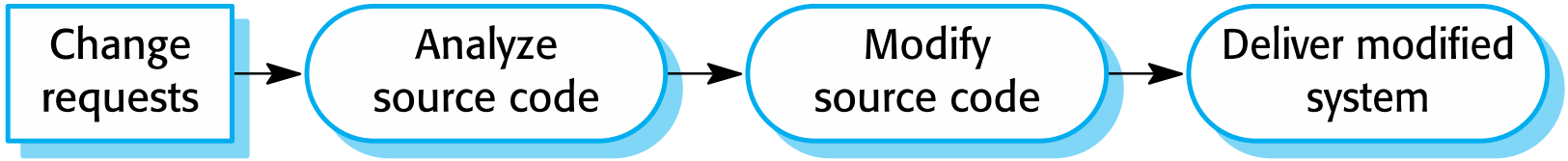
Change implementation



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 that disrupt normal operation;
  - If there are business changes that require a very rapid response (e.g. the release of a competing product).
- Emergency system repairs usually have to be completed as quickly as possible and a quick and workable solution is chosen rather than the best solution as far as system structure is concern.
  - Accelerate the process of software ageing and increase future maintenance cost
  - In practice, re-implement the emergency repairs to improve the software is almost inevitable to have a low priority and is often forgotten and, if further system changes are made, it then becomes unrealistic to redo the emergency repairs

The emergency repair process



Emergency fixes take priority over documentation and eventually the original change is forgotten and the system documentation and code are never realigned

Agile methods and evolution

- Agile methods are based on incremental development so the transition from development to evolution is a seamless one.
  - Evolution is simply a continuation of the development process based on frequent system releases.
- Automated regression testing is particularly valuable when changes are made to a system.
- Changes may be expressed as additional user stories.
- Problems may arise in situations in which there is a handover from a development team to a separate team responsible for evolution.

Handover problems

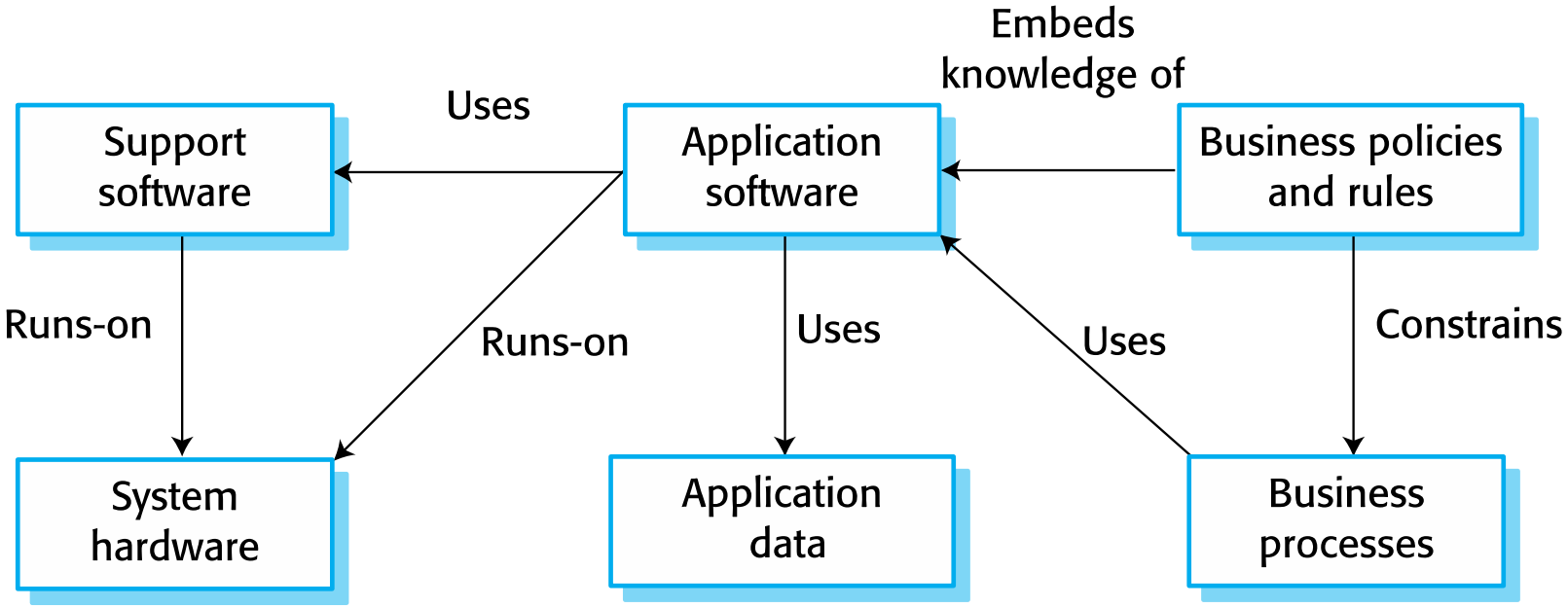
- Where the development team have used an agile approach but the evolution team is unfamiliar with agile methods and prefer a plan-based approach.
  - The evolution team may expect detailed documentation to support evolution and this is not produced in agile processes.
- Where a plan-based approach has 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. Some reengineering may be required to improve the code before it can be used in an agile development process

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 software may be dependent on older hardware, such as mainframe computers and may have associated legacy processes and procedures.
- 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.

The elements of a legacy system



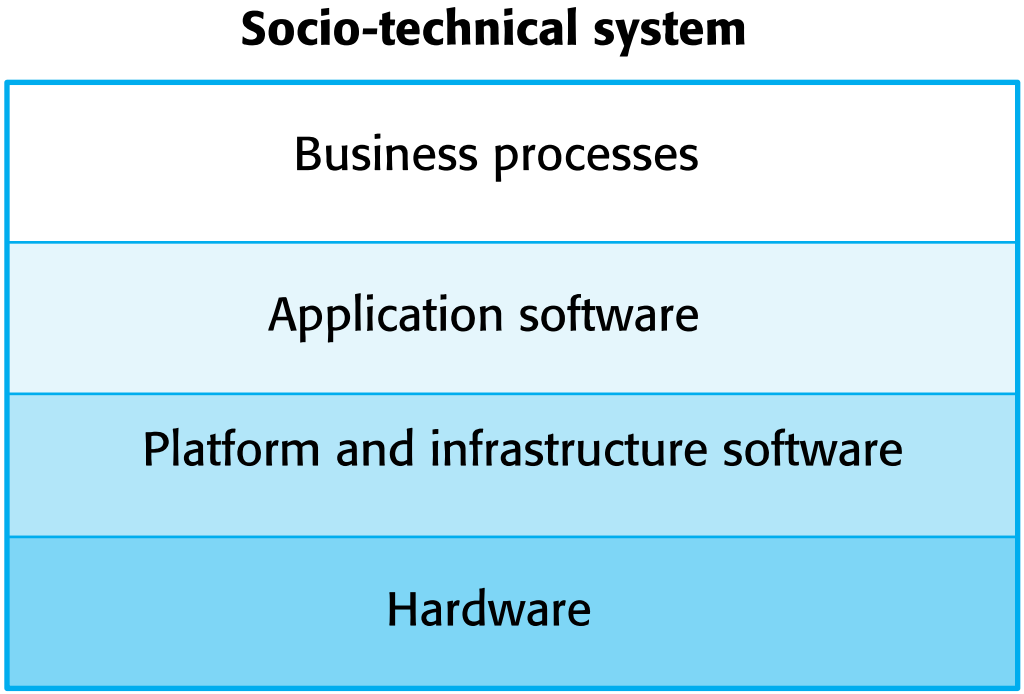
Legacy system components

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

Legacy system components

- *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 layers



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

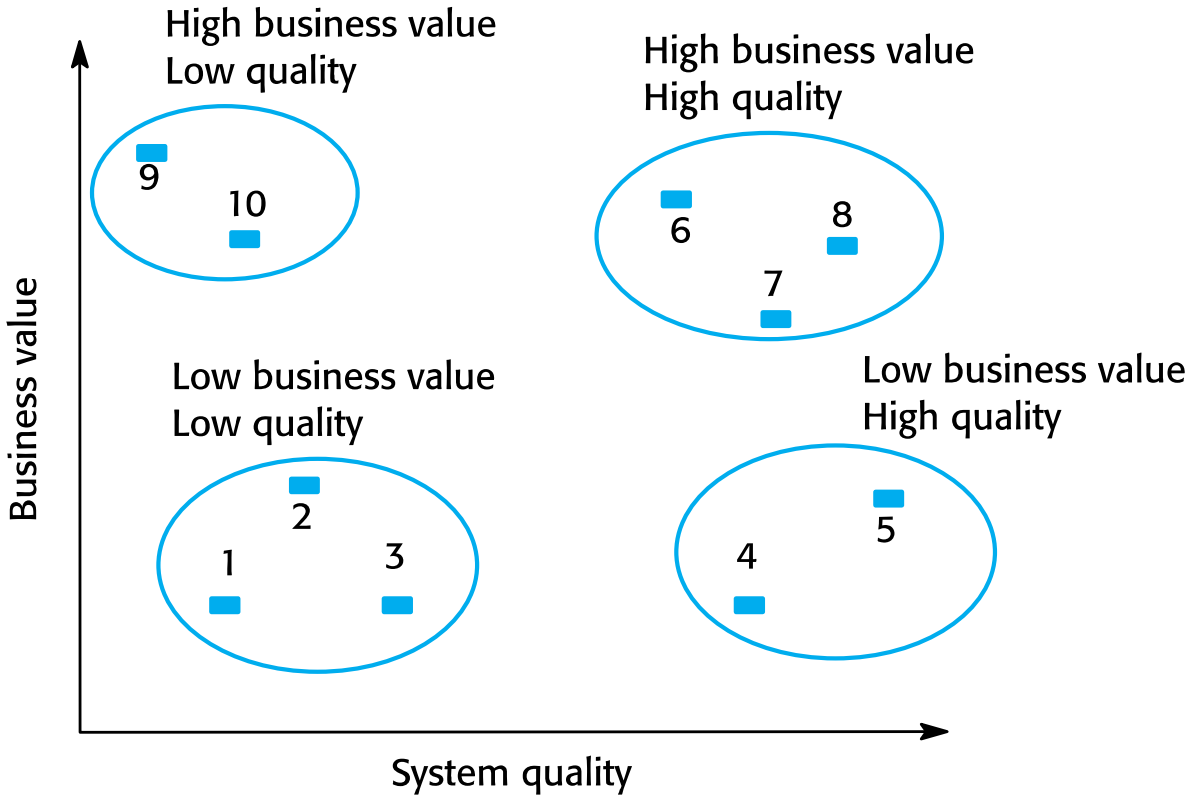
Legacy system change

- 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

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.
- These options are not exclusive. When a system is composed of several programs, different options may be applied to each program.
- 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.
- High-quality, low-business value
  - Replace with COTS, scrap completely or maintain.
- High-quality, high business value
  - Continue in operation using normal system maintenance.

Business value assessment

- Assessment should take different viewpoints into account
  - System end-users;
  - Business customers;
  - Line managers;
  - IT managers;
  - Senior managers.
- Interview different stakeholders and collate results.

Issues in 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

- Business process assessment
  - How well does the business process support the current goals of the business?
- Environment assessment
  - How effective is the system's environment and how expensive is it to maintain?
  - The environment includes the hardware and all associated support software (compilers, development environments, etc)
  - Measure the system and its maintaining process, such as the costs of maintaining the system hardware and support software, the number of hardware faults that occur over some time period, and the frequency of patches and fixes applied to the system support software
- Application assessment
  - What is the quality of the application software system?

Business process assessment

- Use a viewpoint-oriented approach and seek answers from system stakeholders
  - Is there a defined process model and is it followed?
  - Do different parts of the organisation use different processes for the same function?
  - How has the process been adapted?
  - What are the relationships with other business processes and are these necessary?
  - Is the process effectively supported by the legacy application software?
- Example - a travel ordering system may have a low business value because of the widespread use of web-based ordering.

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?

Factors used in environment assessment

Factor	Questions
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?

Factors used in application assessment

Factor	Questions
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.
  - Cleaning up old data is a very expensive and time-consuming process
- The decisions about the quality of the application system can also be based on organizational & political considerations

Software maintenance

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.

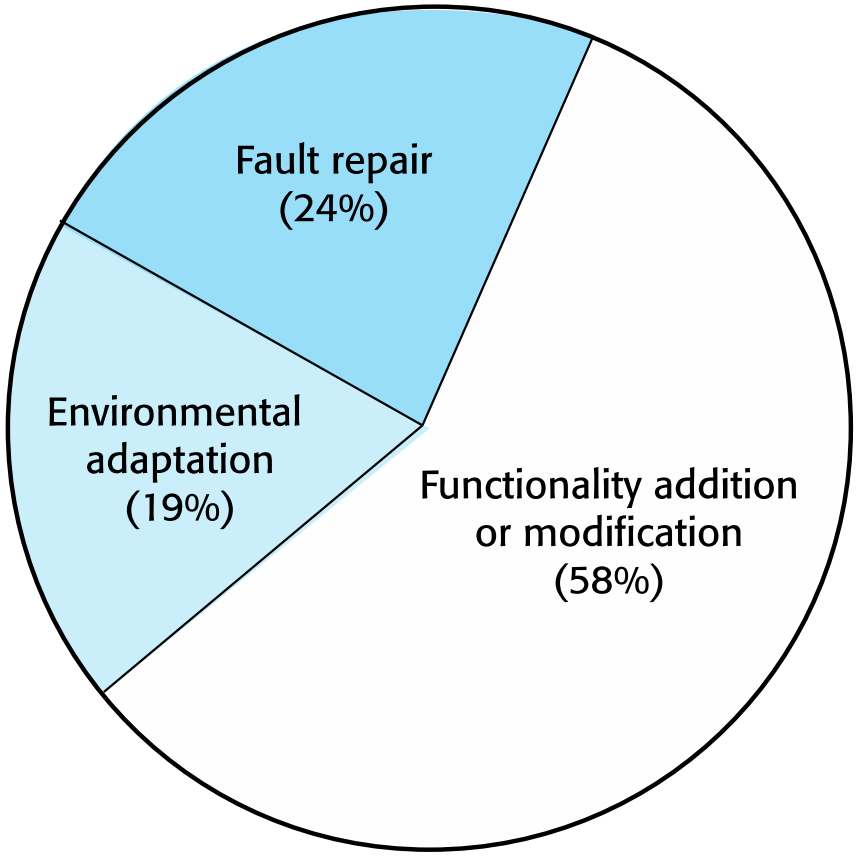
Types of maintenance

- ✧ Maintenance to repair software faults
  - Changing a system to correct deficiencies in the way meets its requirements.
  - **Corrective** 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.
  - **Adaptive** maintenance
- ✧ Maintenance to add to or modify the system's functionality
  - Modifying the system to satisfy new requirements.
  - **Perfective** maintenance

Types of maintenance

- Fault repairs
  - Changing a system to fix bugs/vulnerabilities and correct deficiencies in the way meets its requirements.
  - **Corrective** maintenance
- Environmental adaptation
  - 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.
  - **Adaptive** maintenance
- Functionality addition and modification
  - Modifying the system to satisfy new requirements.
  - **Perfective** maintenance

Maintenance effort distribution

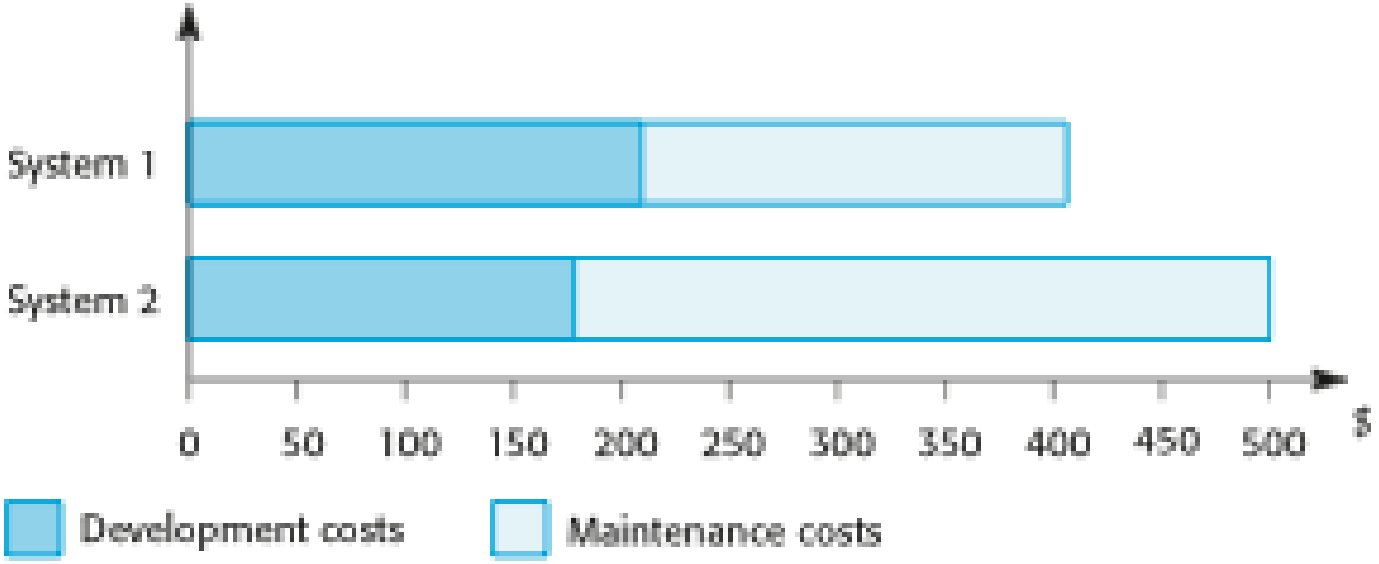


Maintenance costs

- 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.).

Development and maintenance costs

System 1, extra development costs of $25,000 are invested in making the system more maintainable. This results in a savings of $100,000 in maintenance costs over the lifetime of the system.



A percentage increase in development costs results in a comparable percentage decrease in overall system costs. However, in plan-based development, the reality is that additional investment in code improvement is rarely made during development because companies are reluctant to spend money for unknown future return.

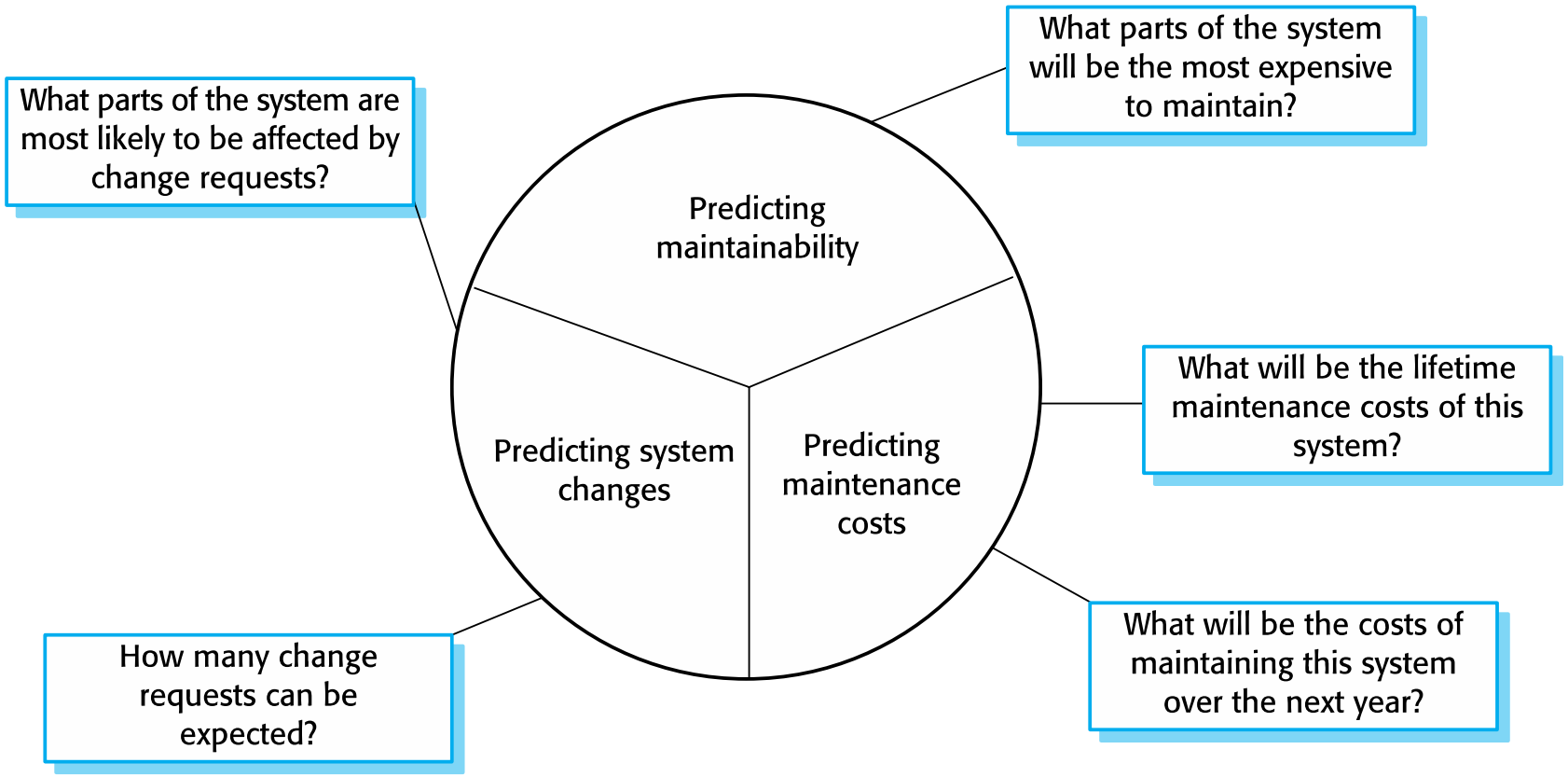
Maintenance cost factors

- It is usually more expensive to add functionality after a system is in operation than it is to implement the same functionality during development
  - 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 (Program maintenance work is unpopular)
    - 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.
    - Can be improved with reengineering techniques, architectural transforming, and refactoring.

Maintenance prediction

- Maintenance prediction is concerned with assessing what system changes might be proposed and 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 requirements reflecting organizational polices & procedures are more volatile vs. stable domain characteristics
  - The business processes where the system is used.
    - The more business processed that use a system, the more the demands for system change

Complexity metrics

- Predictions of maintainability can be made by assessing the complexity of system components.
  - The more complex a system or component, the more expensive it is to maintain
- Studies have shown that most maintenance effort is spent on a relatively small number of system components.
  - Replace complex system components with simpler alternatives
- 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, the process metrics may be used to assess maintainability
  - 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.
- You use predicted information about change requests and predictions about system maintainability to predict maintenance costs
  - The maintenance effort can be based on the effort to understand and update existing code and the effort to develop the new code

System re-engineering

- To make legacy software system easier to maintain, you can reengineer these system to improve their structure and understandability
  - 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**.
  - Involve redocumenting the system, refactoring the system architecture, translating programs to a modern programming language, and modifying and updating the structure and values of the system's data

Advantages of reengineering

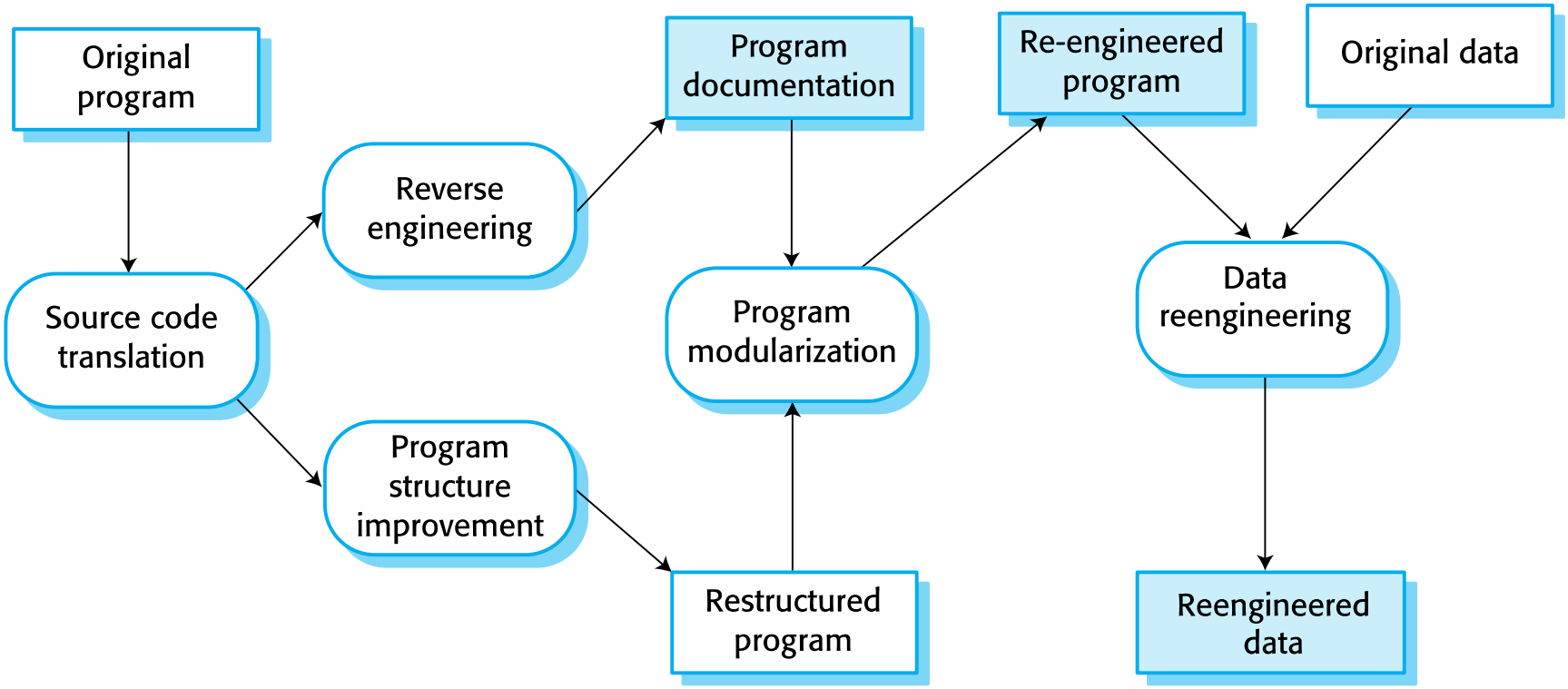
✧ Reduced risk

- There is a high risk in new software development. There may be development problems, staffing problems and specification problems.
- Delays in introducing the new system may cause the lose of business or incursion of extra costs

✧ Reduced cost

- The cost of re-engineering is often significantly less than the costs of developing new software.

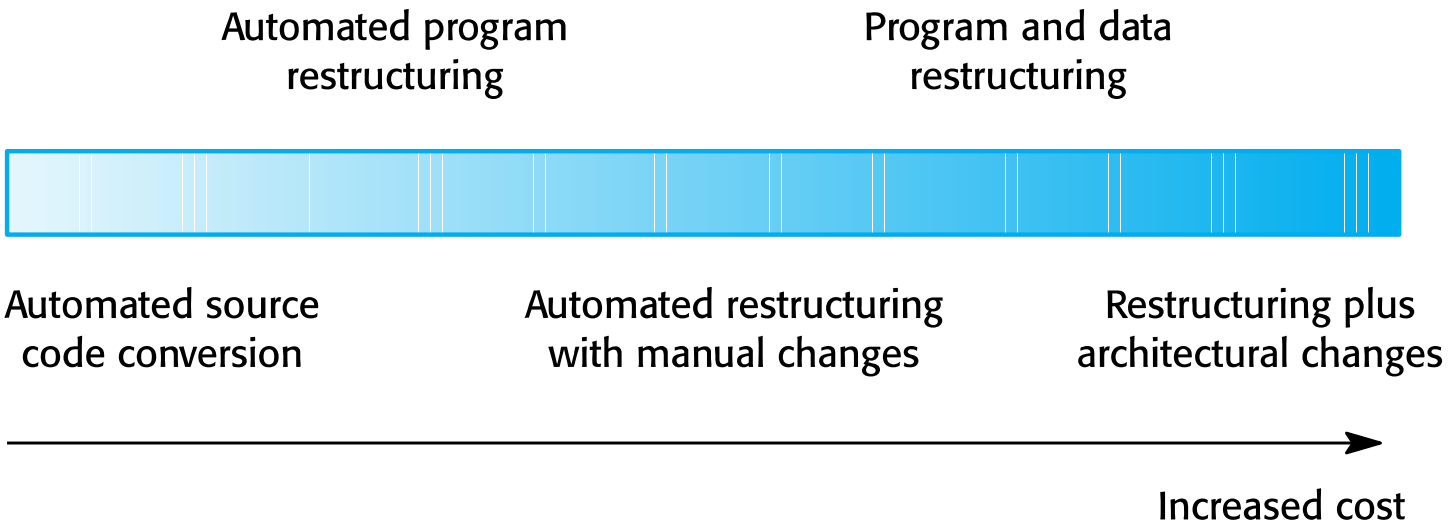
The reengineering process



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 or manually for understandability;
- ✧ Program modularisation
  - Reorganise the program structure; (manual process)
- ✧ Data reengineering
  - Clean-up and restructure system data. (may mean redefining database schema and converting existing database to the new structure)

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.

Preventative maintenance by refactoring

- Refactoring is the process of making improvements to a program to slow down degradation through change.
  - Developing software to make it more maintainable is cost effective
- 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 and reengineering

- Both are intended to make software easier to understand and change
- 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.

‘Bad smells’ in program code

✧ Duplicate code

- The same or very similar code may be included at different places in a program. This can be removed and implemented as a single method or function that is called as required.

✧ Long methods

- If a method is too long, it should be redesigned as a number of shorter methods.

✧ Switch (case) statements

- These often involve duplication, where the switch depends on the type of a value. The switch statements may be scattered around a program. In object-oriented languages, you can often use polymorphism to achieve the same thing.

‘Bad smells’ in program code

- Data clumping
  - Data clumps occur when the same group of data items (fields in classes, parameters in methods) re-occur in several places in a program. These can often be replaced with an object that encapsulates all of the data.
- Speculative generality (過度臆測未來「不必要」的擴充性)
  - This occurs when developers include generality in a program in case it is required in the future. This can often simply be removed.
- Fowler, in his book “*Refactoring*: *Improving the Design of Existing Code*”, suggests several methods to deal with the bad smells, such as Extract method, Pull-up method, Consolidate conditional expression, using refactoring support IDE tool…

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.
- Legacy systems are older software systems, developed using obsolete software and hardware technologies, that remain useful for a business.

Key points

- It is often cheaper and less risky to maintain a legacy system than to develop a replacement system using modern technology.
- 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.
- There are 3 types of software maintenance, namely bug fixing, modifying software to work in a new environment, and implementing new or changed requirements.

Key points

- **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.