CS5030

Software Evolution

Learning objectives

- On completing this lecture and associated reading, you should
  - Understand the reasons why software has to evolve over time
  - Be familiar with software evolution lifecycle
  - Be aware of the concerns regarding managing legacy systems
  - Understand the concepts and processes of software maintenance, software re-engineering and refactoring

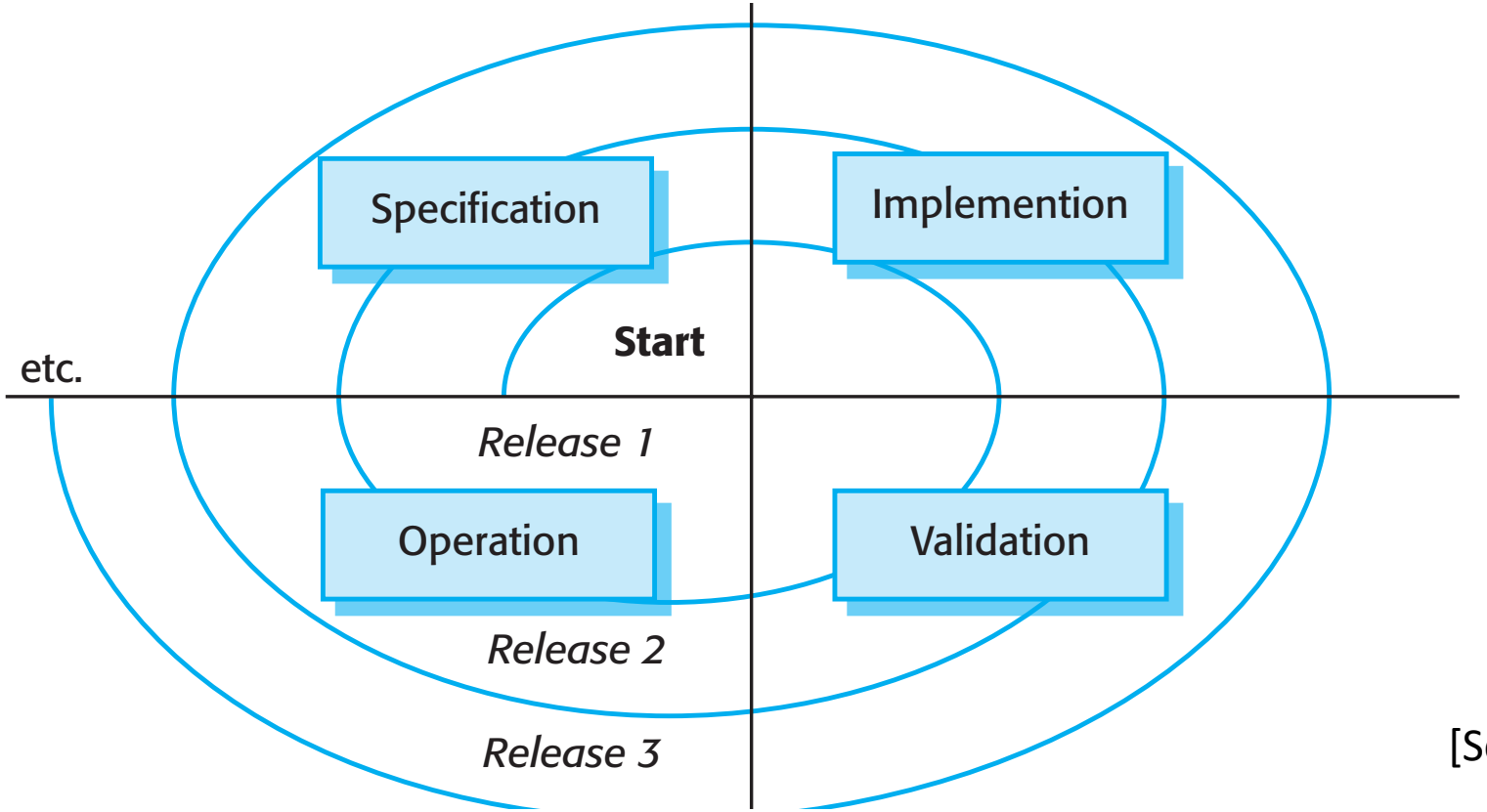
Software change

- Most software systems are expected to be long-lasting
- Software change is inevitable
  - New requirements emerge when the software is used
  - The business environment changes
  - Errors must be repaired
  - New hardware or infrastructure is added to the system
  - The performance or reliability of the system may have to be improved
- Changes can lead to degradation of software quality and reduced understanding of the system
- A key problem for all organisations is implementing and managing change to their existing software systems

Importance of change

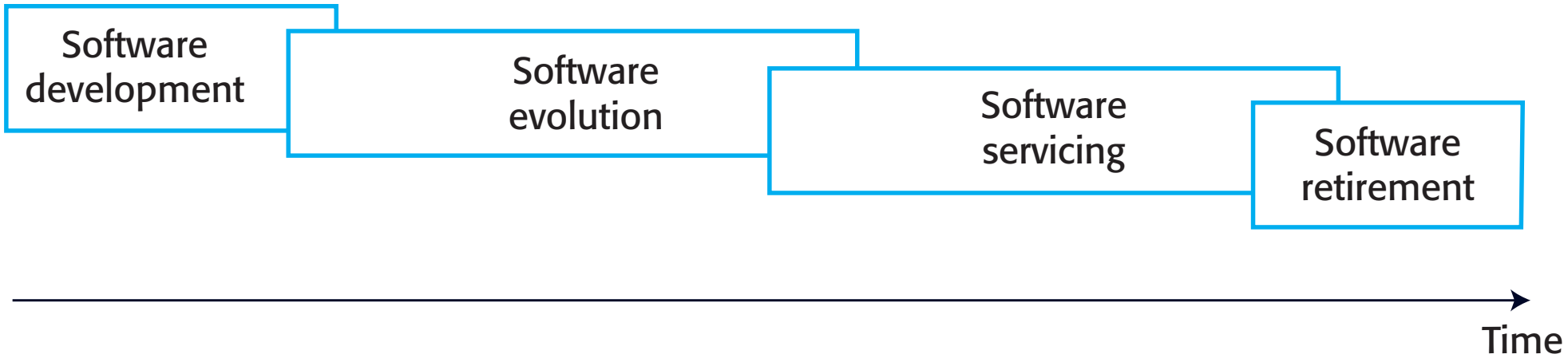
- Organisations have huge investments in their software systems
  - Critical business assets
- They must be kept updated so they maintain their value to the business
- Software budget in large companies
  - More is used for changing and evolving existing software rather than developing new software

A spiral model of development and evolution



[Sommerville, 2016]

Software evolution lifecycle



[Sommerville, 2016]

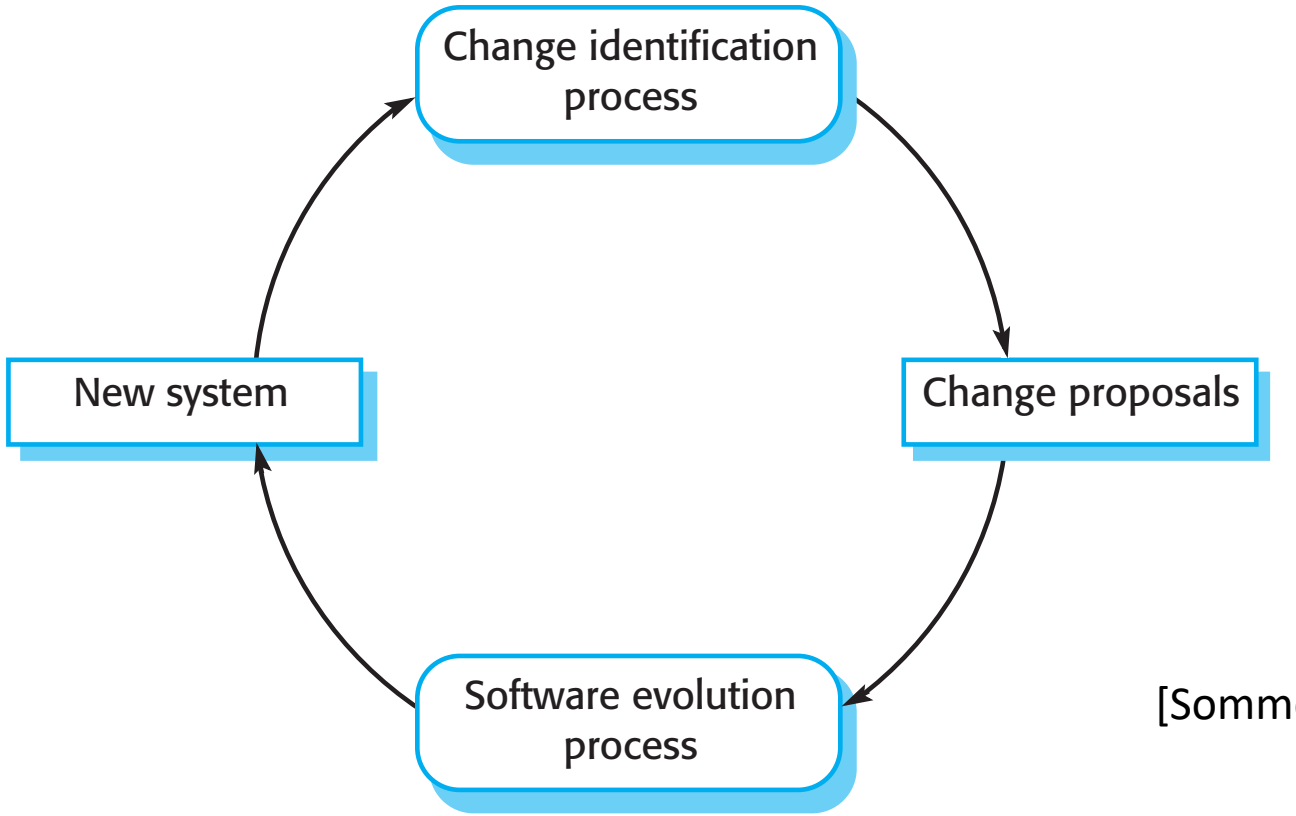
Evolution lifecycle

- Evolution
  - The system is in operational use and is evolving as new requirements are proposed and implemented in the system
- Servicing
  - The system remains useful but the only changes made are those required to keep it operational
    - Bug fixes and adapting to reflect the operational environment
  - No new functionality is added
- Phase-out
  - The system may still be used but no further changes are made to it

Evolution processes

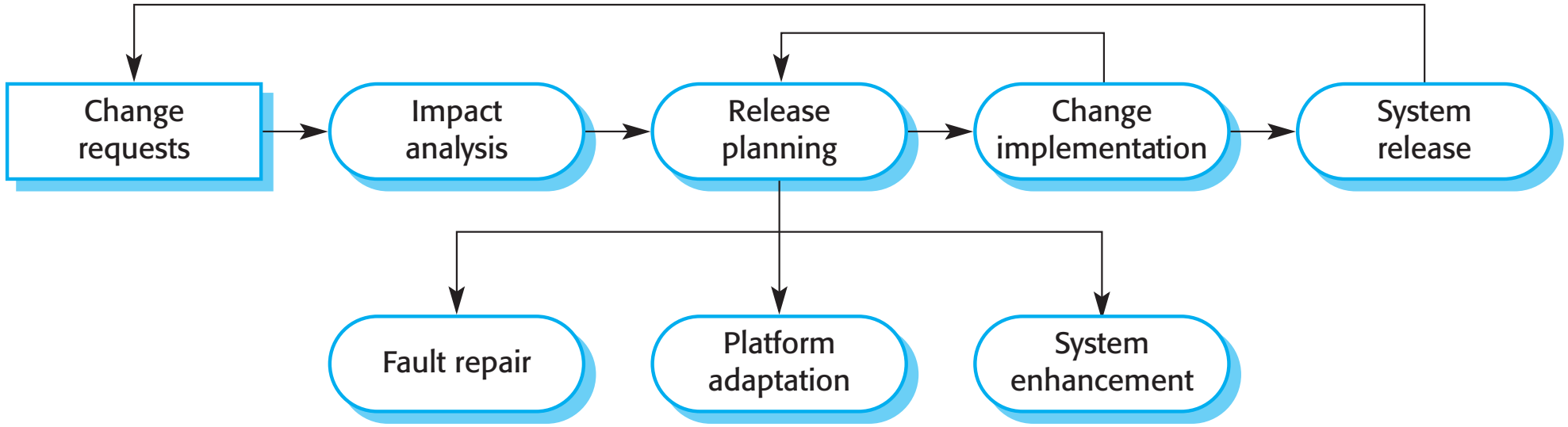
- Software evolution processes depend on
  - The type of software being maintained
  - The development processes used
  - The skills and experience of the people involved
- Proposals for change are the driver for system evolution
  - Should be linked to components that are affected by the change
  - Cost and impact of the change can then be estimated
- Change identification and evolution will continue throughout the system lifetime

Evolution processes



[Sommerville, 2016]

Software evolution process – a general model



[Sommerville, 2016]

Agile development and evolution

- Agile methods use incremental development so the transition from development to evolution is seamless
  - 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

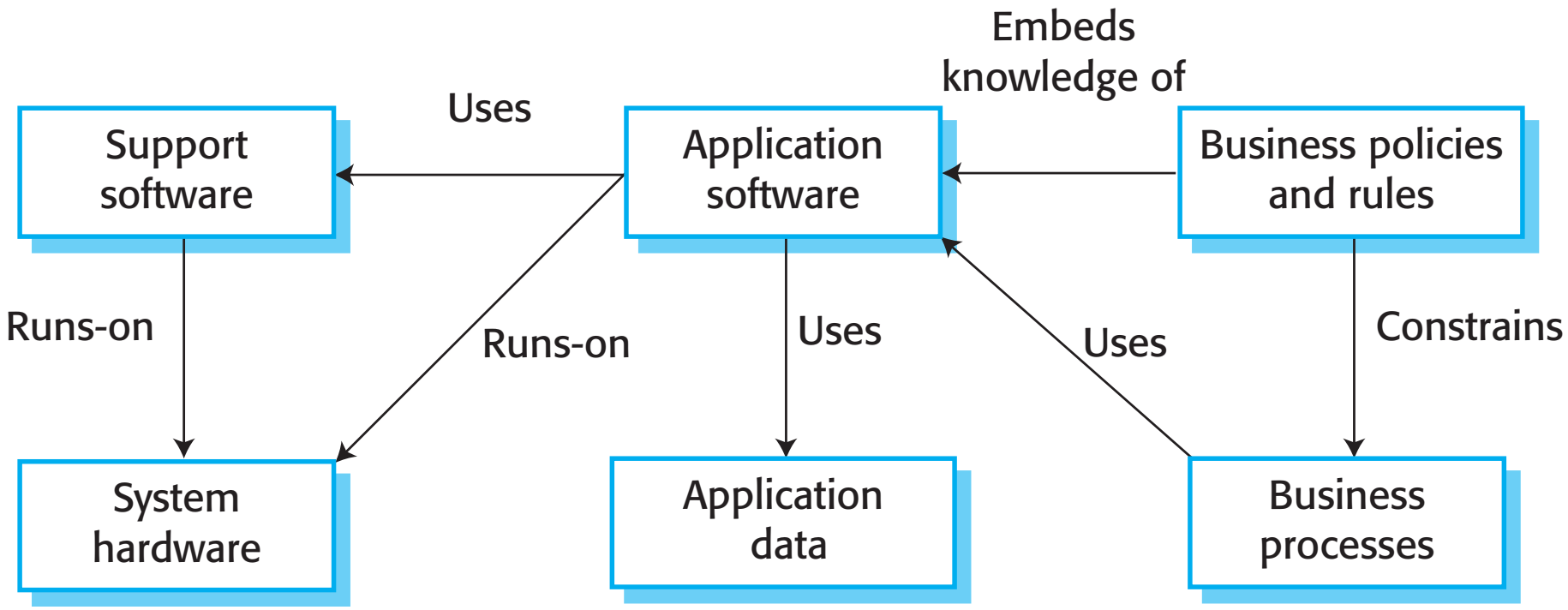
Handover problems

- Mismatch between development and evolution approaches
- When the development team has used an agile approach but the evolution team prefers a plan-based approach
  - The evolution team may expect detailed documentation to support evolution and this is not produced in agile processes
- When a plan-based approach has been used for development but the evolution team prefers 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 expected in agile development

Legacy systems

- Older systems that rely on languages and technologies that are no longer used for new systems development
- Legacy software may
  - Be dependent on older hardware, such as mainframe computers
  - Have associated legacy processes and procedures
- Not just software systems
  - Broader socio-technical systems that include hardware, software, libraries and other supporting software and business processes

Elements of legacy systems



[Sommerville, 2016]

Replacing legacy systems

- 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

Changing legacy systems

- Legacy systems may be 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 optimisations may make software hard to understand
  - Data errors, duplication and inconsistency

Managing legacy systems

- 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

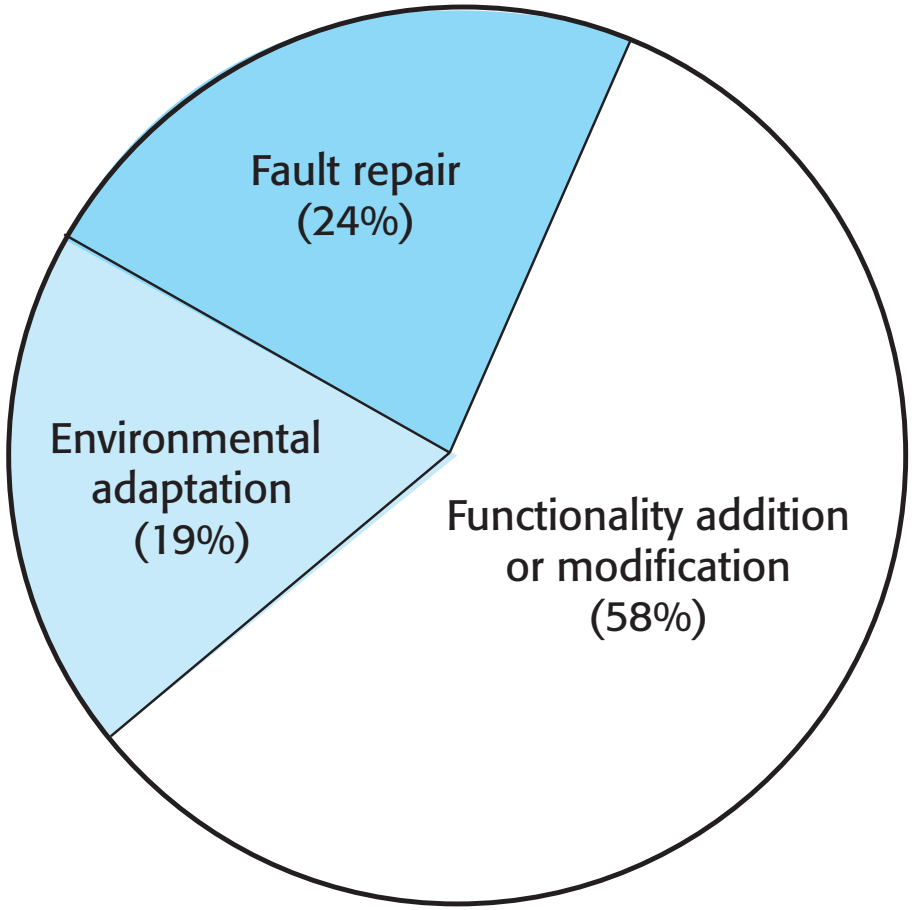
Software maintenance

- 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

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

Maintenance effort



[Sommerville, 2016]

Maintenance costs

- Usually greater than development costs
- Affected by both technical and non-technical factors
- Increases as software is maintained
  - Maintenance corrupts the software structure and makes further maintenance more difficult
- Ageing software can have high support costs
  - Old languages, compilers etc.
  - Availability of personnel with knowledge and skills

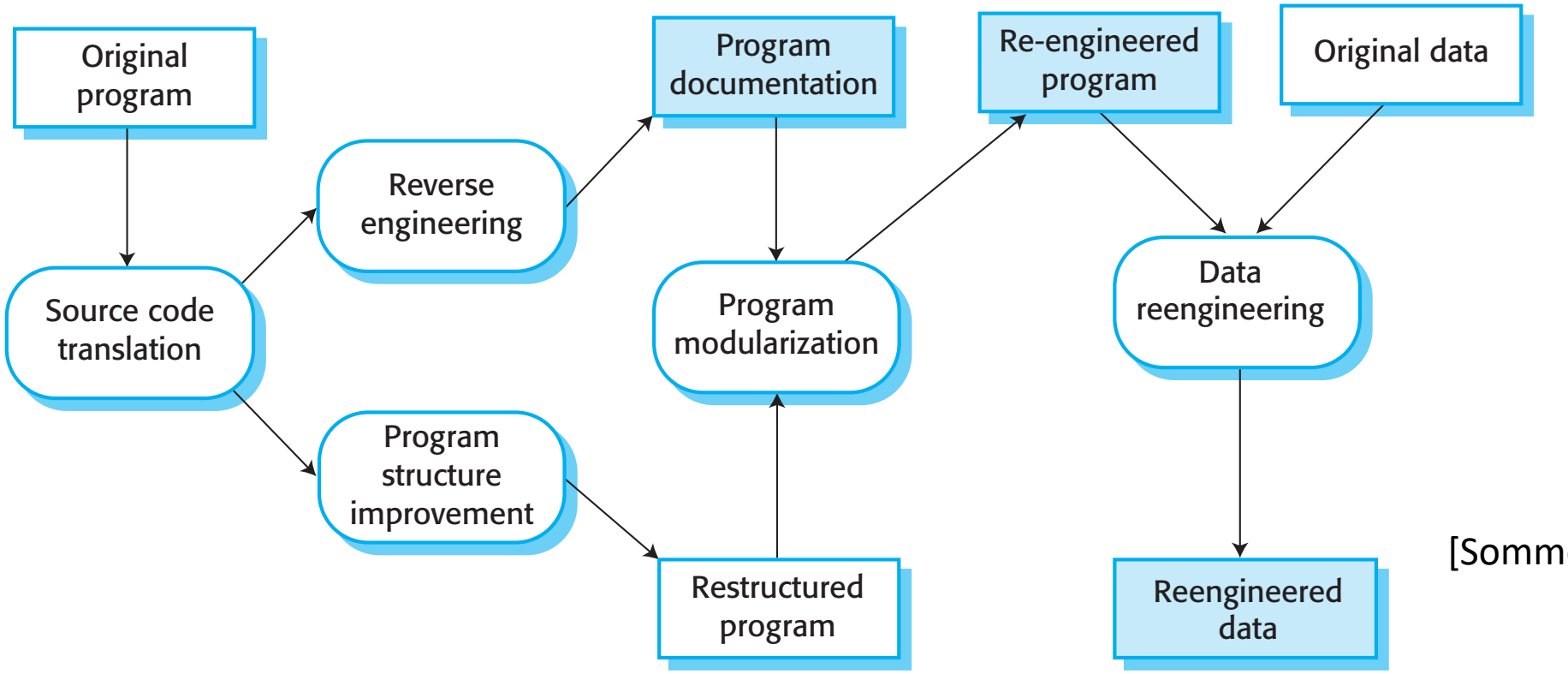
Software re-engineering

- Restructuring or rewriting part or all of an existing 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

Benefits of re-engineering

- Reduced risk
  - There is 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

Re-engineering process



[Sommerville, 2016]

Factors affecting re-engineering costs

- The quality of the software to be re-engineered
- The tool support available for re-engineering
- The extent of the data conversion required
- The availability of expert staff for re-engineering
  - This can be a problem with old systems based on technologies that are no longer widely used

Refactoring

- The process of making improvements to a program to slow down degradation through change
- Can be thought of as ‘preventative maintenance’ that reduces the problems of future change
- Involves modifying a program to
  - improve its structure
  - reduce its complexity or
  - make it easier to understand
- During refactoring, the focus is on software improvement rather than adding new functionality

Refactoring vs re-engineering

- Re-engineering takes place after a system has been maintained for some time and when maintenance costs are increasing
  - Automated tools are used 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 degradation of structure and code that increases the costs and difficulties of maintaining a system

Examples of bad smells in software

- Duplicate code
- Long methods
- Data clumping
- Speculative generality

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, software maintenance costs usually exceed software development costs
- Legacy systems are older software systems, developed using obsolete software and hardware technologies, that remain useful for a business
- It is often cheaper and less risky to maintain a legacy system than to develop a replacement system using modern technology

Key points

- There are 3 types of software maintenance: 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