ICT 2402 Software Engineering

Software Evolution

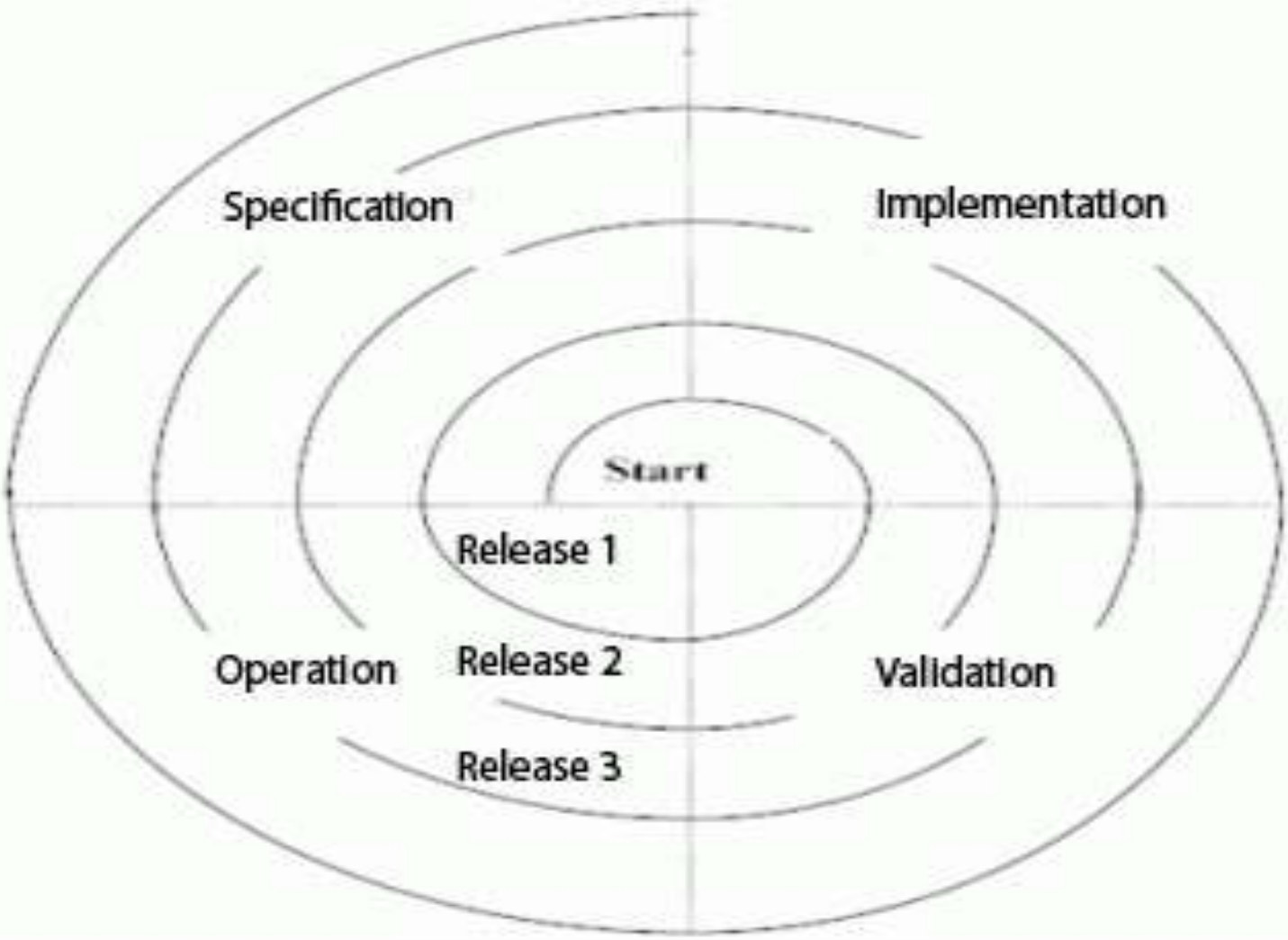
Topics covered

- Program evolution dynamics
- Software maintenance
- Evolution process
- Legacy system evolution
- Configurations management

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 organizational is implementing and managing change to their existing software systems.

Spiral model of development and evolution



Program evolution dynamics

- Program evolution dynamics is the study of the processes of system change.
- After major empirical studies, Lehman and Belady proposed that there were a number of 'laws' which applied to all systems as they evolved.
- There are sensible observations rather than laws. They are applicable to large systems developed by large organizations Perhaps less applicable in other cases.

Lehman's laws

Law	Description
Continuing change	A program that is used in a real-world environment necessarily must change or become progressively less useful in that environment.
Increasing complexity	As an evolving program changes, its structure tends to become more complex. Extra resources must be devoted to preserving and simplifying the structure.
Large program evolution	Program evolution is a self-regulating process. System attributes such as size, time between releases and the number of reported errors is approximately invariant for each system release.
Organisational stability	Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development.
Conservation of familiarity	Over the lifetime of a system, the incremental change in each release is approximately constant.
Continuing growth	The functionality offered by systems has to continually increase to maintain user satisfaction.
Declining quality	The quality of systems will appear to be declining unless they are adapted to changes in their operational environment.
Feedback system	Evolution processes incorporate multi-agent, multi-loop feedback systems and you have to treat them as feedback systems to achieve significant product improvement.

Applicability of Lehman's laws

- Lehman's laws seem to be generally applicable to large, tailored systems developed by large organizations
  - Confirmed in more recent work by Lehman on the FEAST project.
- It is not clear how they should be modified for
  - Shrink-wrapped software products;
  - Systems that incorporate a significant number of COTS components;
  - Small organizations;
  - Medium sized systems.

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.

Software maintenance activities

- Scheduled maintenance activities:
  - Adaptive
  - Preventive
  - Perfective
  - Corrective
- Unscheduled corrective maintenance – to deal with an error that must be dealt with immediately – emergency maintenance
- Effort needed depends on type, scope, effect, severity and priority of the maintenance

Adaptive maintenance

- Activities involve adapting the software to new operating environment, new software or hardware paradigm or technology, new type of user interface
- Driven by management based on market demand
- Typically require new requirements and interface specifications
- Scheduled activities

Corrective maintenance

- Diagnosing software errors and making changes to fix (correct) them
- Found by users after deployment or found by testers but were not fixed prior to deployment
- Unscheduled – if error is of high impact and critical and need to be attended for immediately
- Can be caused by a requirement, design, coding or testing error or caused by previous maintenance
- Documentation must be updated after maintenance
- Coding errors are the simplest to fix

Perfective maintenance

- Activities involve changes leading to improving on both functional and non-functional requirements
  - Enhancing performance, usability, …
- For example, modifying the UI to optimize on interactions; modifying the query processing algorithm and code; modifying the compiler optimization algorithm and code
- Scheduled maintenance activities initiated by external users or internally by developers

Preventive maintenance

- Activities involve software changes to improve future maintainability and reduce future discovery of errors
- Could be a large scale scheduled process: restructuring, re-engineering, design recovery …
- Could be part of a software evolution plan
- 75% are adaptive and preventive
- 20% corrective, 5% perfective

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.

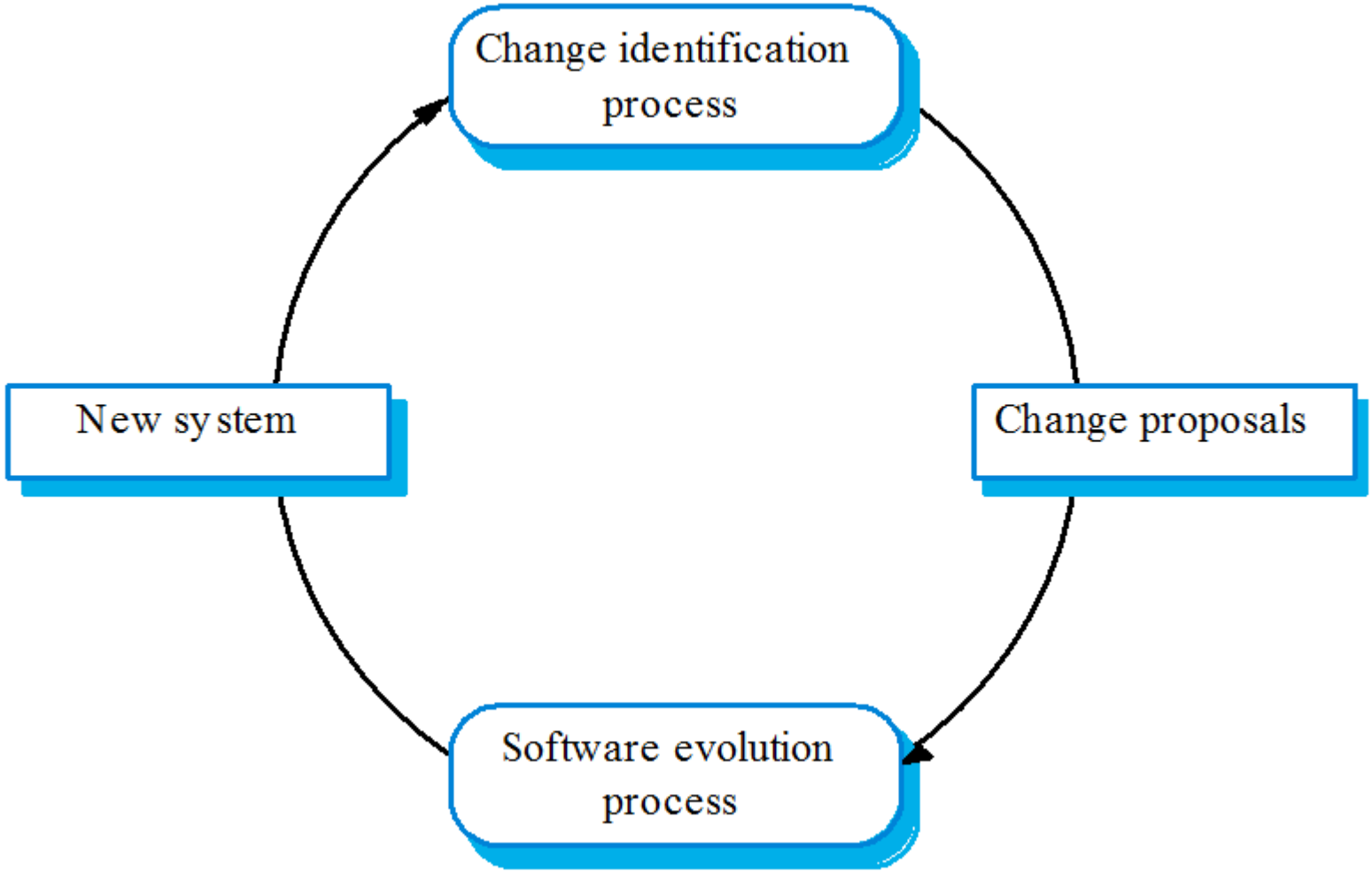
Maintenance prediction

- 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.

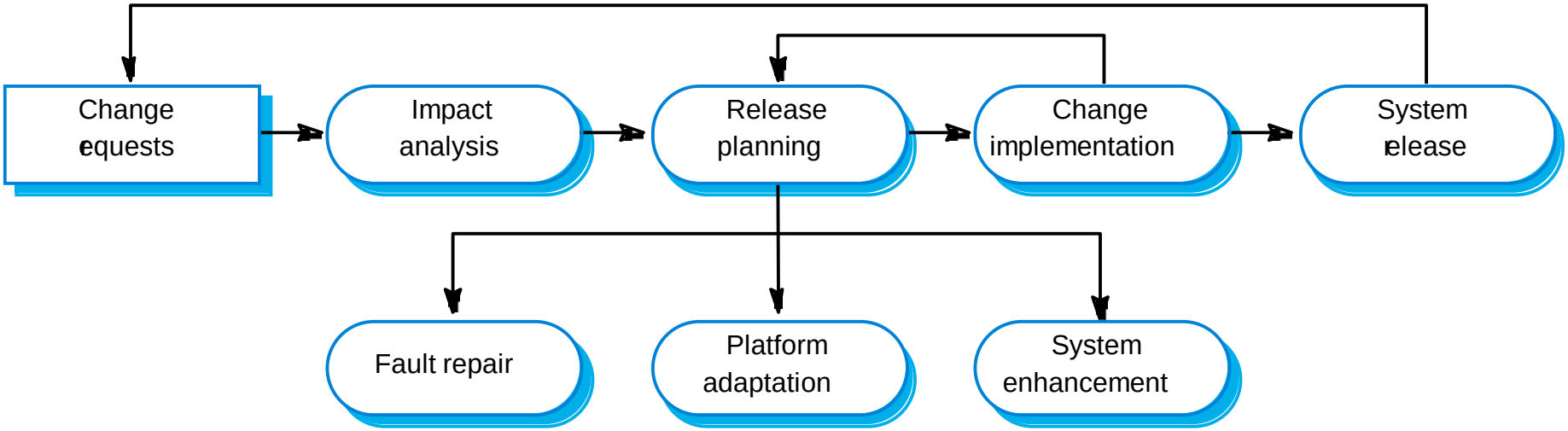
Evolution processes

- 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. Change identification and evolution continue throughout the system lifetime.

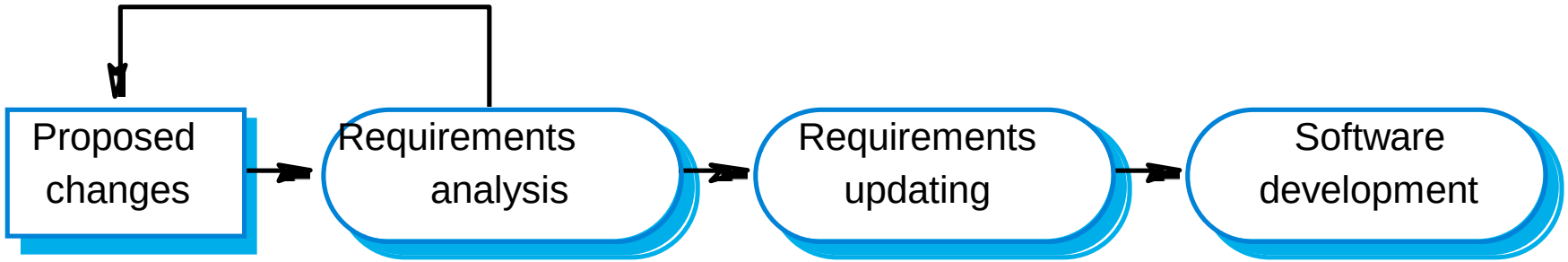
Change identification and evolution



The system evolution process



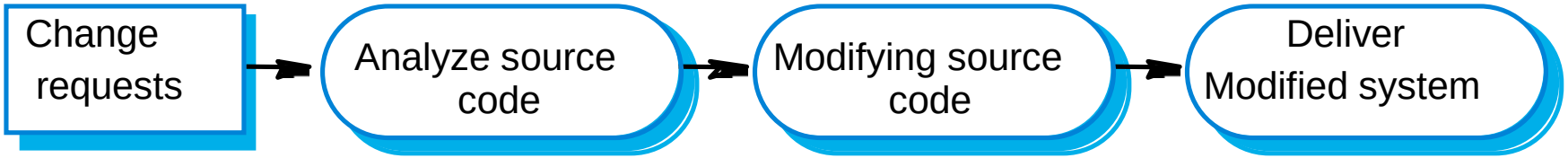
Change implementation



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;
  - 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).

Emergency repair



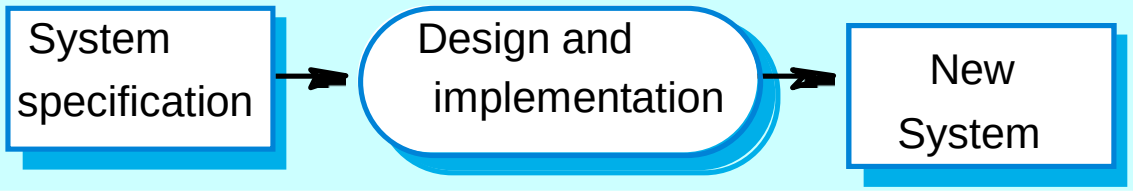
System re-engineering

- 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.

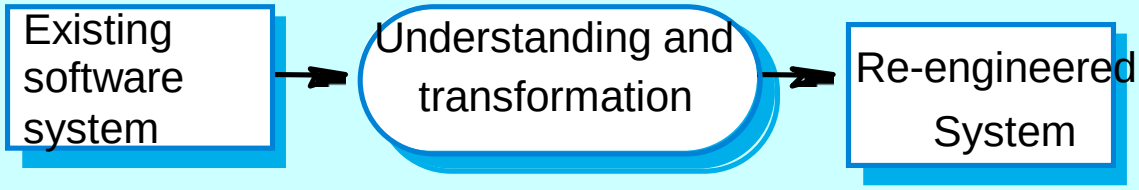
Advantages of re-engineering

- 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.

Forward vs. re-engineering

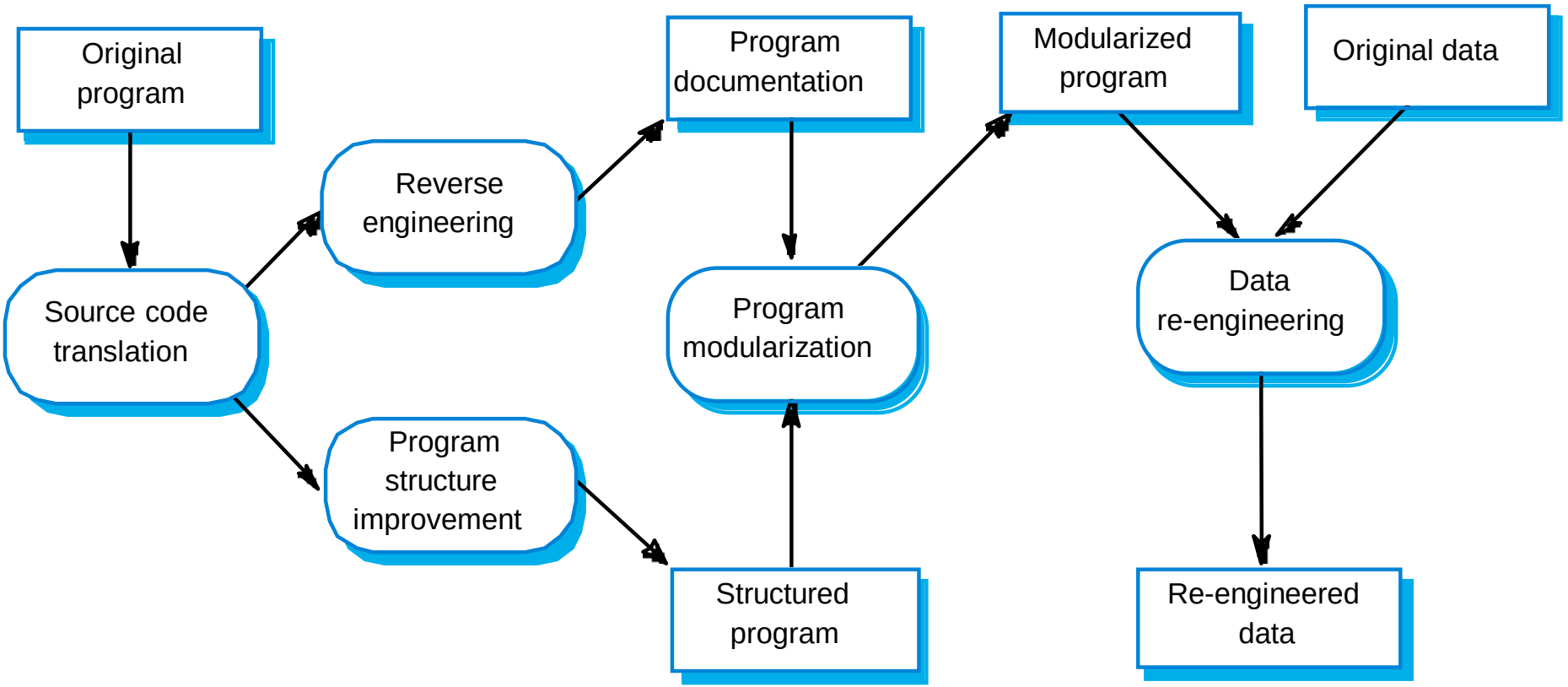


Forward Engineering



Re-engineering process

The re-engineering process



Re-engineering 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.

Legacy system evolution

- Organizations 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.

Software configuration management

- New versions of software systems are created as they change:
  - For different machines/OS;
  - Offering different functionality;
  - Tailored for particular user requirements.
- Configuration management is concerned with managing evolving software systems:
  - System change is a team activity;
  - CM aims to control the costs and effort involved in making changes to a system.

Configuration management

- Involves the development and application of procedures and standards to manage an evolving software product.
- CM may be seen as part of a more general quality management process.
- When released to CM, software systems are sometimes called baselines as they are a starting point for further development.

Questions?