Martin Glinz Harald Gall

Software Engineering

Wintersemester 2005/06

Kapitel 12

Software Evolution und Reengineering

Universität Zürich
Institut für Informatik

Overview

- Software Evolution
- Software Maintenance, Legacy Systems
- Evolution Processes
- Reengineering

Objectives

- To explain why change is inevitable if software systems are to remain useful
- To discuss software maintenance and maintenance cost factors
- To describe the processes involved in software evolution
- To discuss an approach to assessing evolution strategies for legacy systems

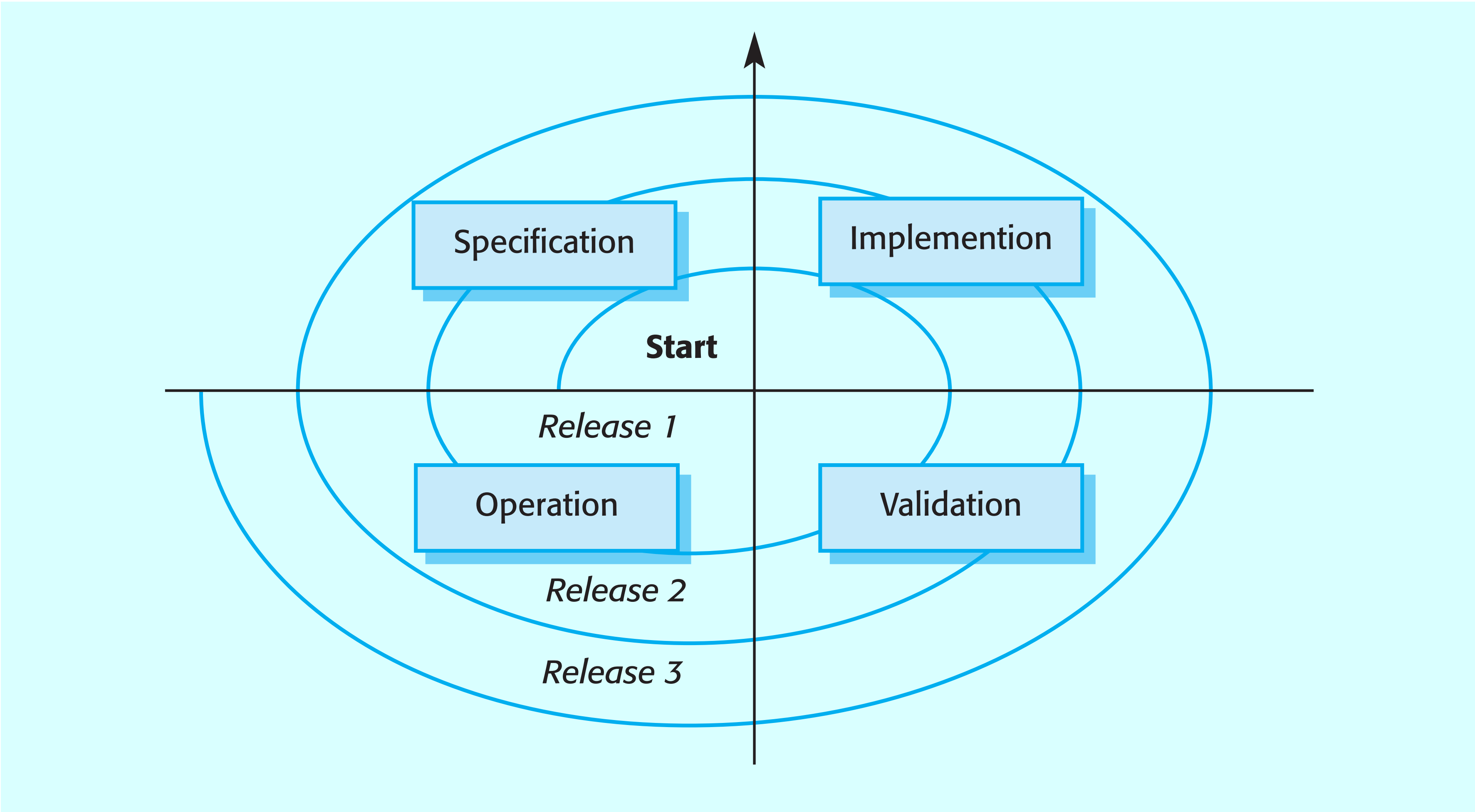
Software change

- 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 organisations is implementing and managing change to their *existing* software systems.

Importance of evolution

- Organizations 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 evolving existing software rather than developing new software.

Spiral model of evolution



Program evolution dynamics

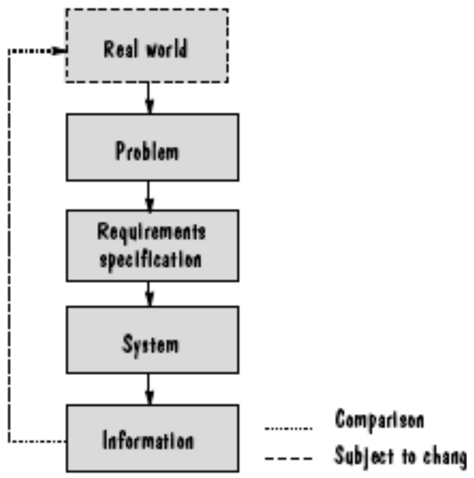
- Program evolution dynamics is the study of the processes of system change.
- After major empirical studies, Lehman and Belady proposed that there were a number of 'laws' which applied to all systems as they evolved.
- There are *sensible observations* rather than laws. They are applicable to large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

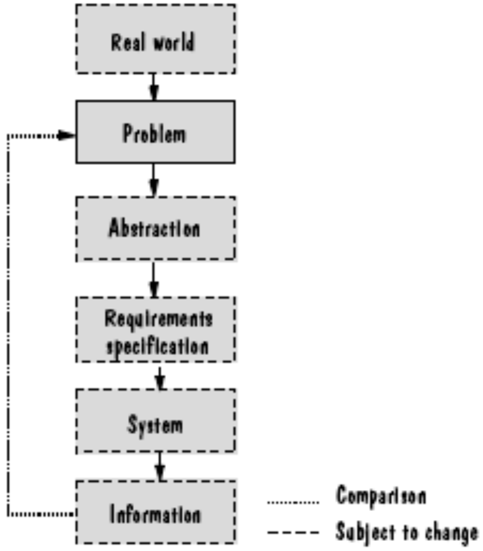
Lehman’s laws

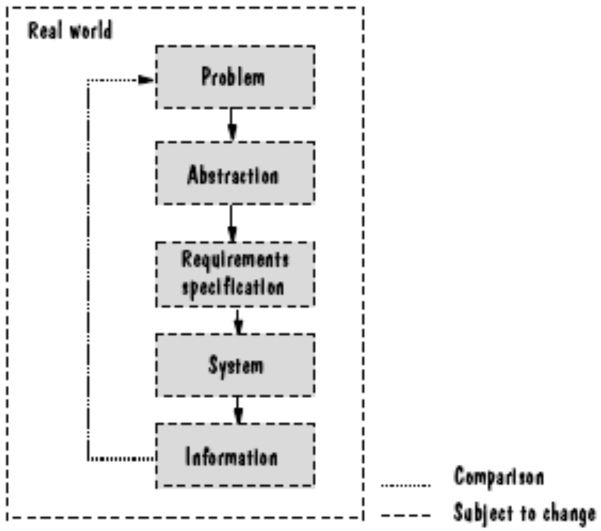
Law	Description
Continuing change	A program that is used in a real-world environment necessarily must change or become progressively less useful in that environment.
Increasing complexity	As an evolving program changes, its structure tends to become more complex. Extra resources must be devoted to preserving 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.

Lehman's system types

- S-system: formally defined, derivable from a specification
- P-system: requirements based on approximate solution to a problem, but real-world remains stable
- E-system: embedded in the real world and changes as the world does







Applicability of Lehman's laws

- Lehman's laws seem to be generally applicable to large, tailored systems developed by large organisations.
  - Confirmed in more recent work by Lehman on the FEAST project (http://www.doc.ic.ac.uk/~mml/feast/).
- It is open how they should be modified for
  - Shrink-wrapped software products;
  - Systems that incorporate a significant number of COTS components;
  - Small organisations;
  - 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.

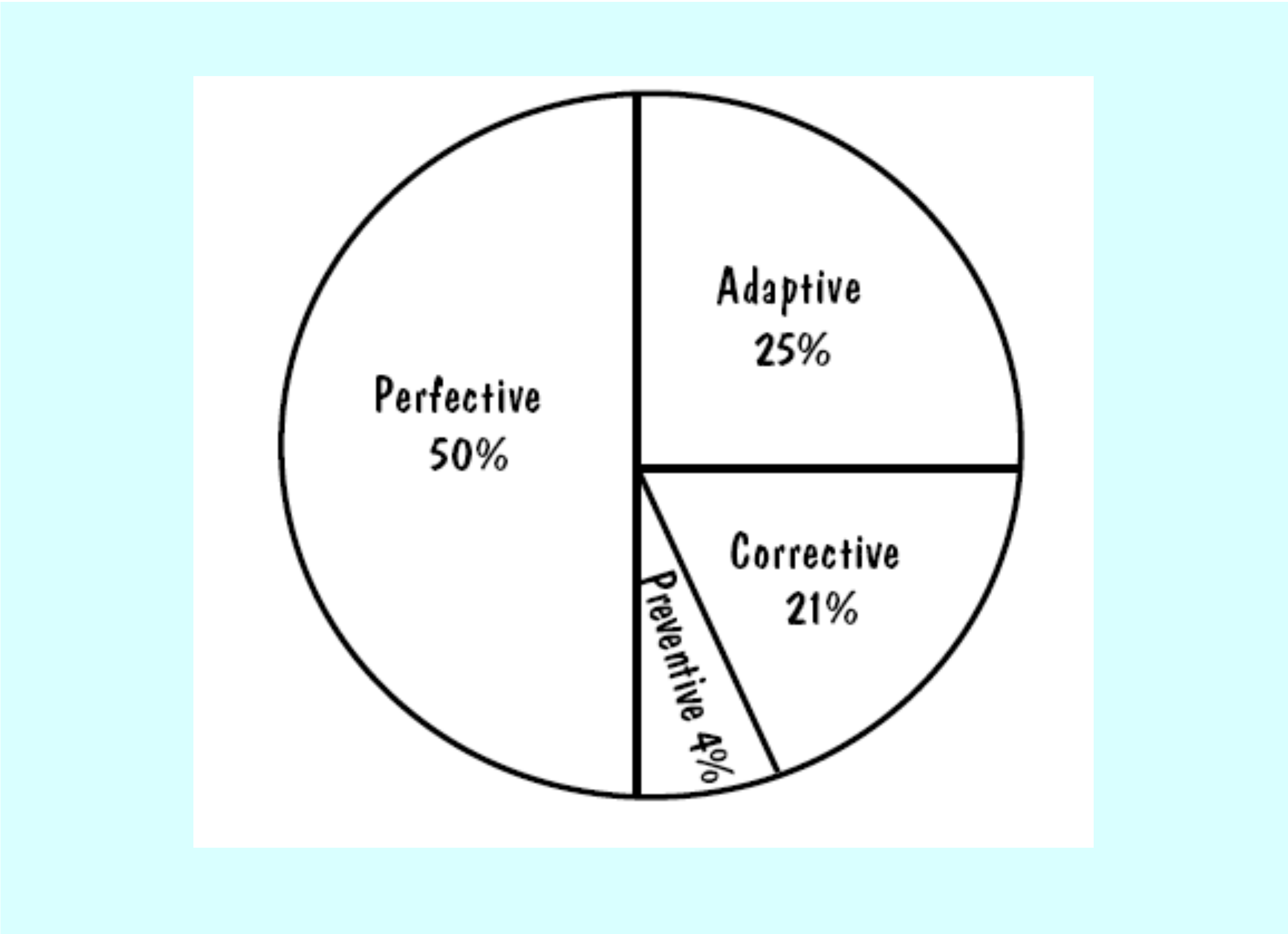
Maintenance is inevitable

- The system requirements are likely to change while the system is being developed because the environment is changing. Therefore a delivered system won't meet its requirements!
- Systems are tightly coupled with their environment. When a system is installed in an environment it changes that environment and therefore changes the system requirements.
- Systems MUST be maintained therefore if they are to remain useful in an environment.

Types of maintenance

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

Maintenance effort



System evolution vs. decline

- Is the cost of maintenance too high?
- Is the system reliability unacceptable?
- Can the system no longer adapt to further change, and within a reasonable amount of time?
- Is system performance still beyond prescribed constraints?
- Are system functions of limited usefulness?
- Can other systems do the same job better, faster or cheaper?
- Is the cost of maintaining the hardware great enough to justify replacing it with cheaper, newer hardware?

Maintenance team responsibilities

- understanding the system
- locating information in system documentation
- keeping system documentation up-to-date
- extending existing functions to accommodate new or changing requirements
- adding new functions to the system
- finding the source of system failures or problems
- locating and correcting faults
- answering questions about the way the system works
- restructuring design and code components
- rewriting design and code components
- deleting design and code components that are no longer useful
- managing changes to the system as they are made

Maintenance problems

- Staff problems
  - Limited understanding
  - Management priorities
  - Morale
- Technical problems
  - Artifacts and paradigms
  - Testing difficulties

Maintainability and Maintenance Costs

Factors affecting maintenance effort

- Application type
- System novelty
- Turnover and maintenance staff ability
- System life span
- Dependence on a changing environment
- Hardware characteristics
- Design quality
- Code quality
- Documentation quality
- Testing quality

Measuring maintainability

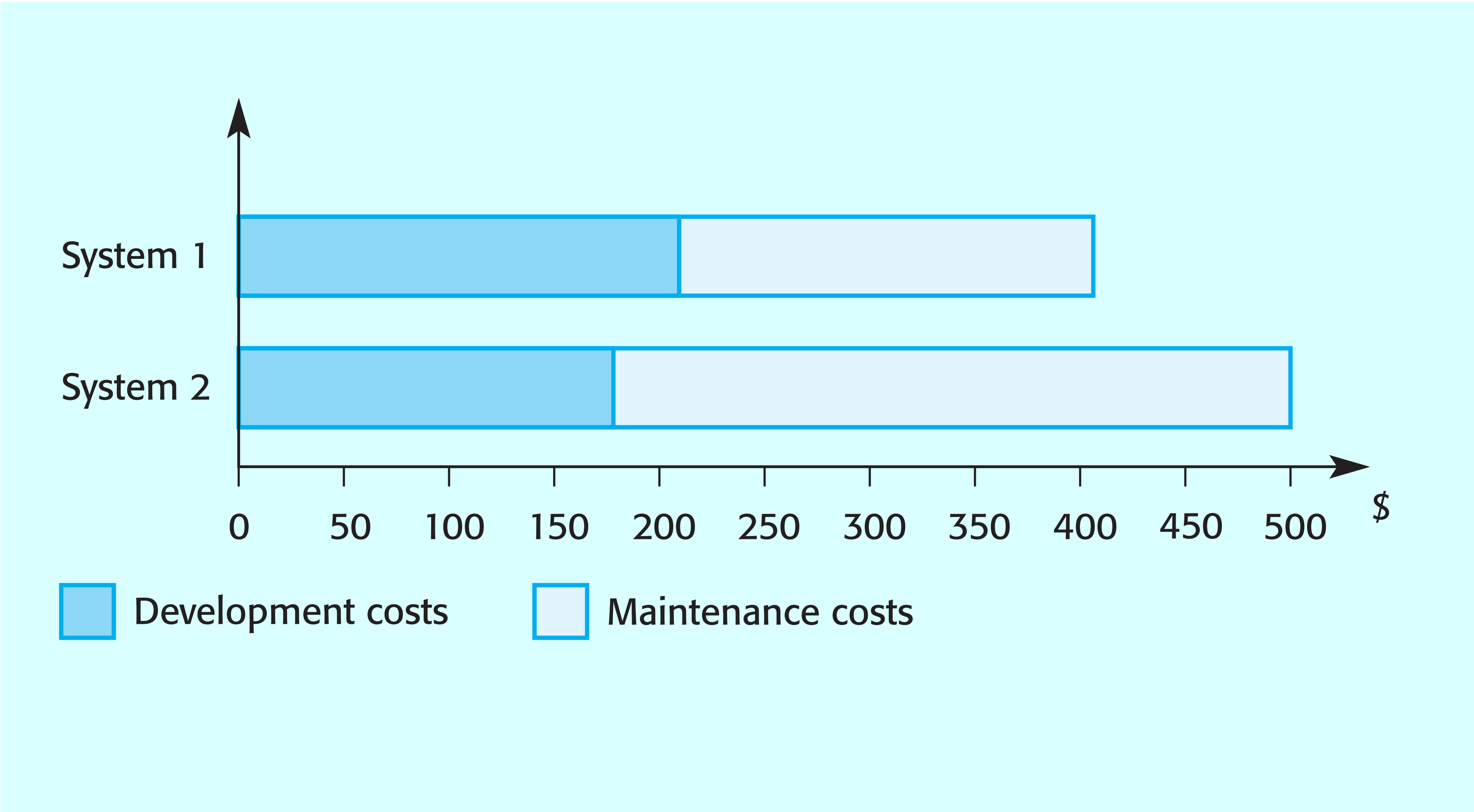
- Necessary data:
  - time at which problem is reported
  - time lost due to administrative delay
  - time required to analyze problem
  - time required to specify which changes are to be made
  - time needed to make the change
  - time needed to test the change
  - time needed to document the change

- Desirable data:
  - ratio of total change implementation time to total number of changes implemented
  - number of unresolved problems
  - time spent on unresolved problems
  - percentage of changes that introduce new faults
  - number of components modified to implement a change

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/maintenance costs



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.

Modeling Maintenance Effort (1)

- Belady and Lehman equation:
  - $M = p + K^{c-d}$
    - M ... total maintenance effort,
    - p ... productive efforts,
    - c ... complexity caused by lack of structured design and documentation,
    - d ... c reduced by d, the degreee to which the maintenance team is familiar with the software
    - K ... empirical constant determined by comparing this model with the effort relationships on actual projects

Modeling Maintenance Effort (2)

- COCOMO II:
  - Size = ASLOC (AA + SU +0.4*DM +0.3*CM + 0.3*IM) /100
    - ASLOC ... number of source lines to be adapted
    - DM ... percentage of design to be modified
    - CM ... percentage of code to be modified
    - IM ... percentage of external code (e.g. reuse code) to be integrated
    - SU ... rating scale representing the amount of software understanding required (Table 11.2)
    - AA ... assessment and assimiliation effort to assess code and make changes (Table 11.3)

COCOMO II - Software Understanding

	Very low	*Low*	*Nominal*	*High*	*Very high*
Structure	Very low cohesion, high coupling, spaghetti code	Moderately low cohesion, high coupling	Reasonably well-structured; some weak areas	High cohesion, low coupling	Strong modularity, information-hiding in data and control structures
Application clarity	No match between program and application world views	Some correlation between program and application	Moderate correlation between program and application	Good correlation between program and application	Clear match between program and application world views
Self-descriptiveness	Obscure code; documentation missing, obscure or obsolete	Some code commentary and headers; some useful documentation	Moderate level of code commentary, headers, documentation	Good code commentary and headers; useful documentation; some weak areas	Self-descriptive code; documentation up-to-date, well-organized, with design rationale
SU increment	**50**	**40**	**30**	**20**	**10**

Table 11.2. COCOMO II rating for software understanding

COCOMO II - Assessment & Assimilation

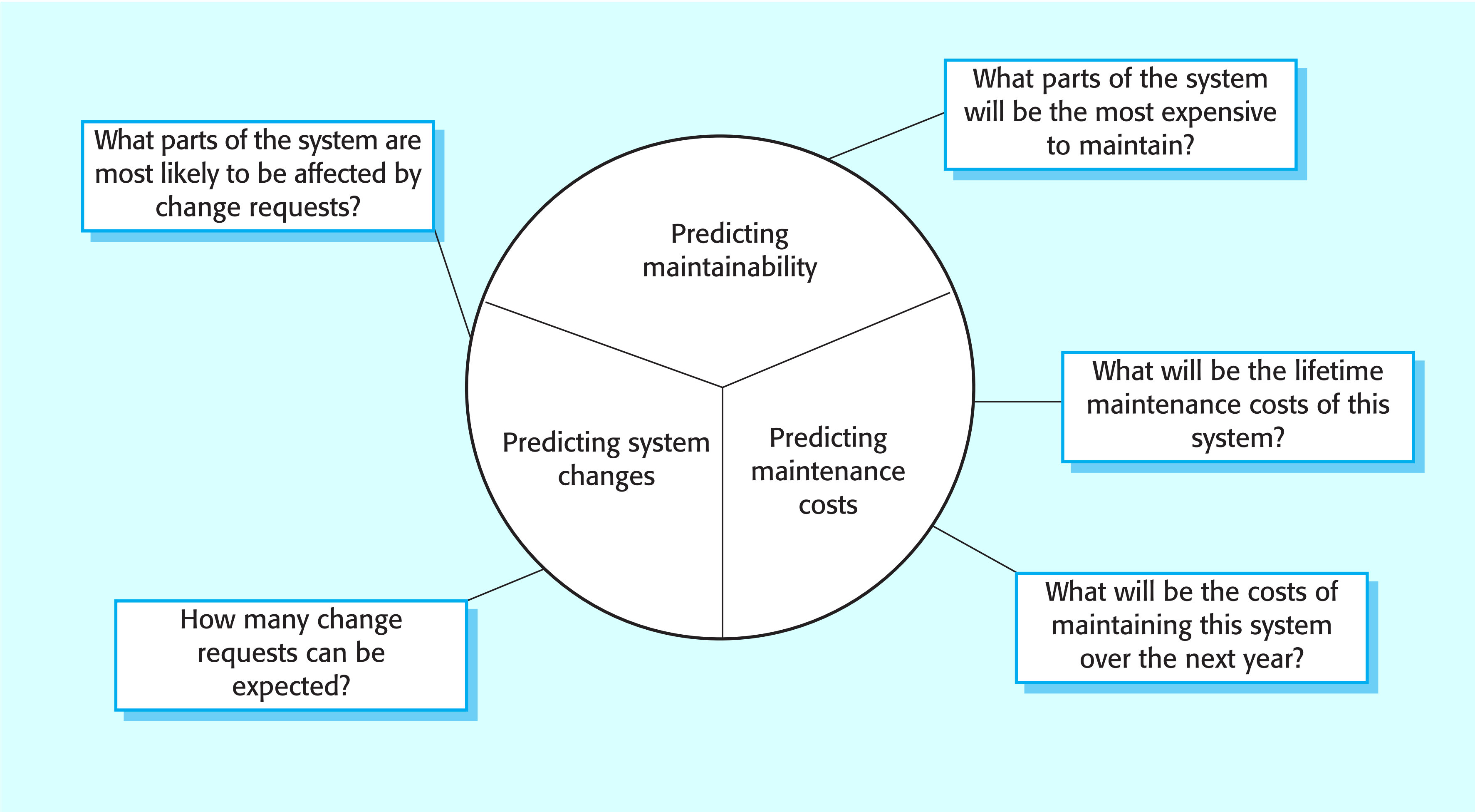
Table 11.3. COCOMO II ratings for assessment and assimilation effort.

Assessment and assimilation increment	*Level of assessment and assimilation effort*
0	None
2	Basic component search and documentation
4	Some component test and evaluation documentation
6	Considerable component test and evaluation documentation
8	Extensive component test and evaluation documentation

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.

Maintenance prediction



Change prediction

- Predicting the number of changes requires an 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 business processes where the system is used.

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.

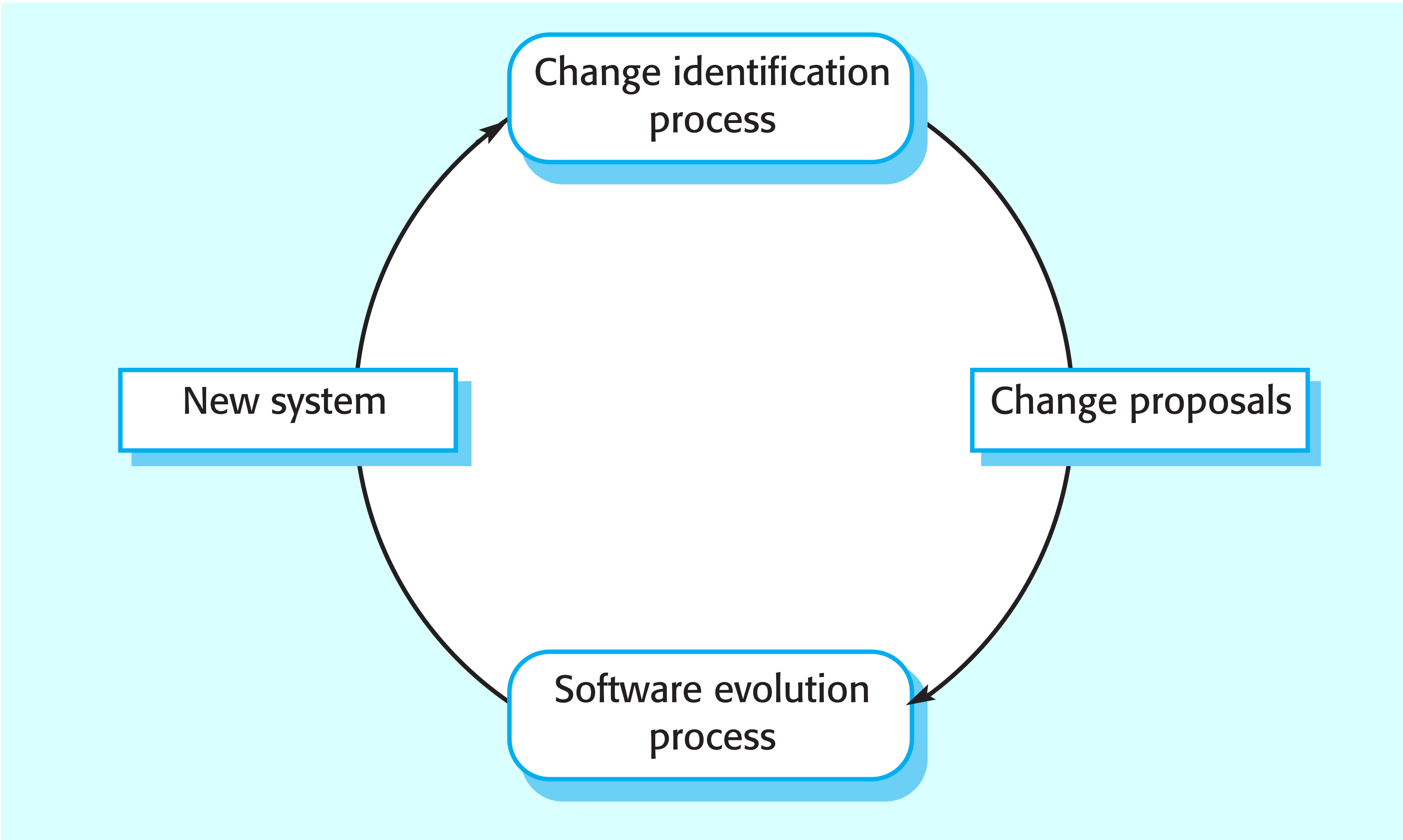
Process metrics

- Process measurements 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 (queued) change requests.
- If any or all of these is increasing, this may indicate a decline in maintainability.

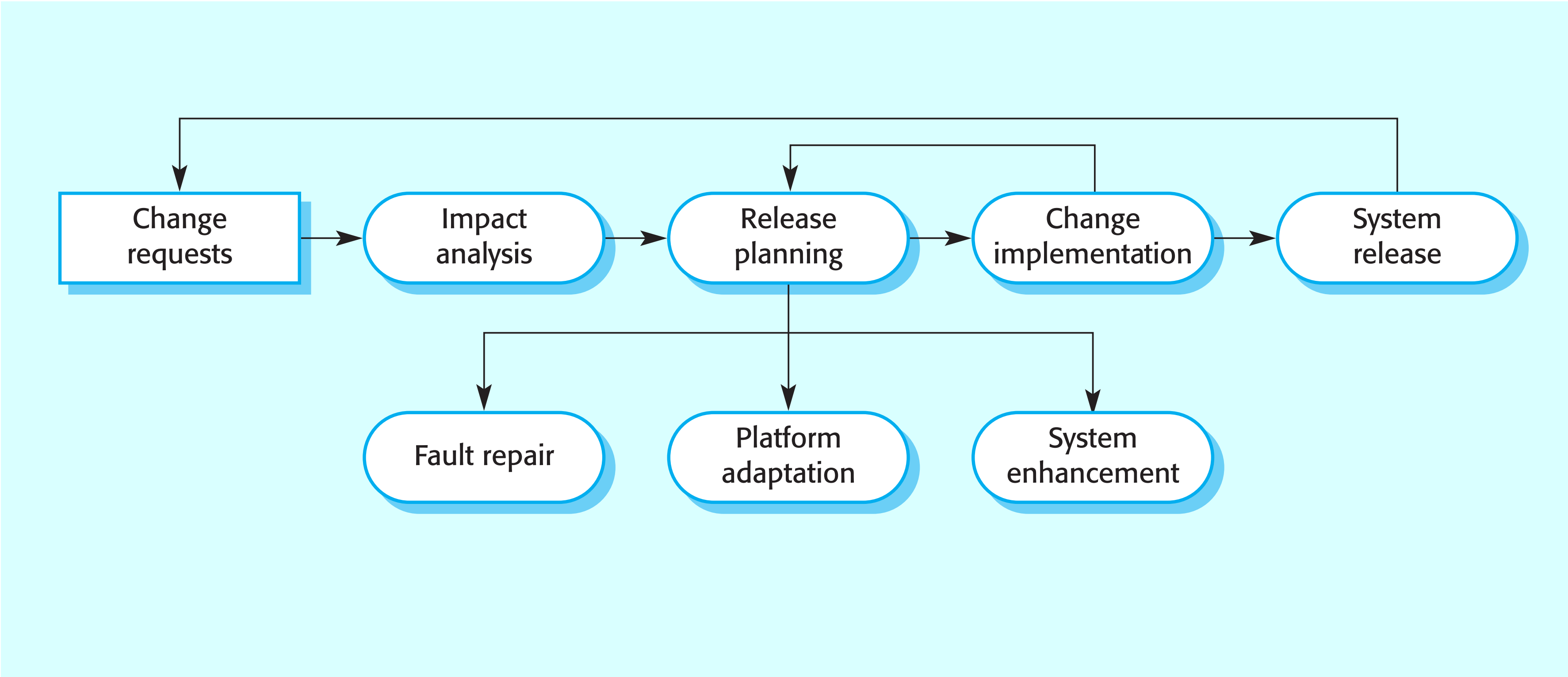
Evolution processes

- 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.

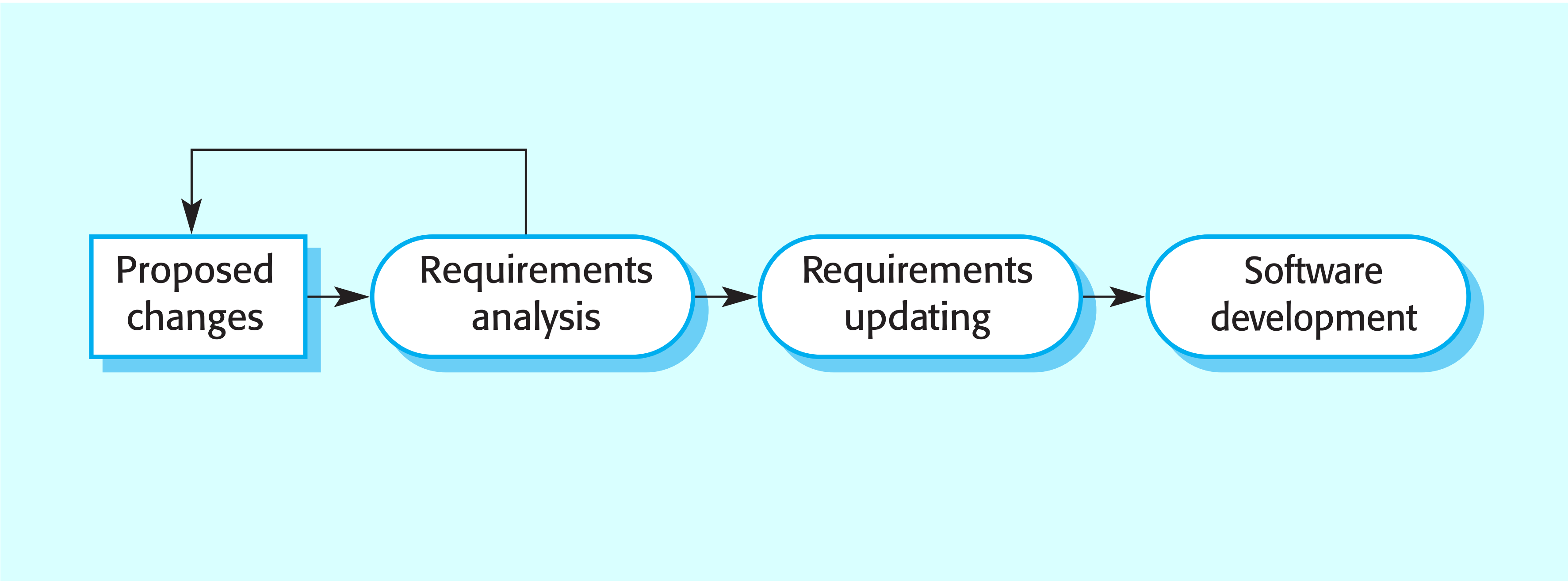
Change identification and evolution



The system evolution process



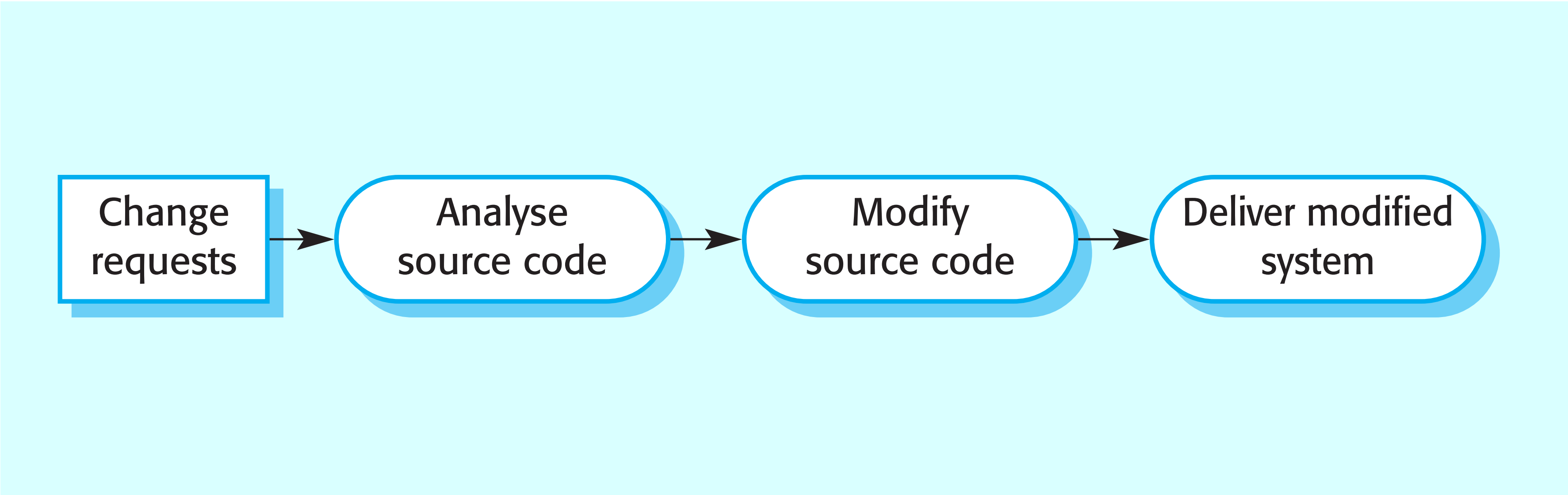
Change implementation



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



Configuration control process

- Problem discovered by or change requested by user/customer/developer, and recorded
- Change reported to the Configuration Control Board (CCB)
  - CCB discusses problem: determines nature of change, who should pay
  - CCB discusses source of problem, scope of change, time to fix; they assign severity/priority and analyst to fix
- Analyst makes change on test copy
- Analyst works with librarian to control installation of change
- Analyst files change report

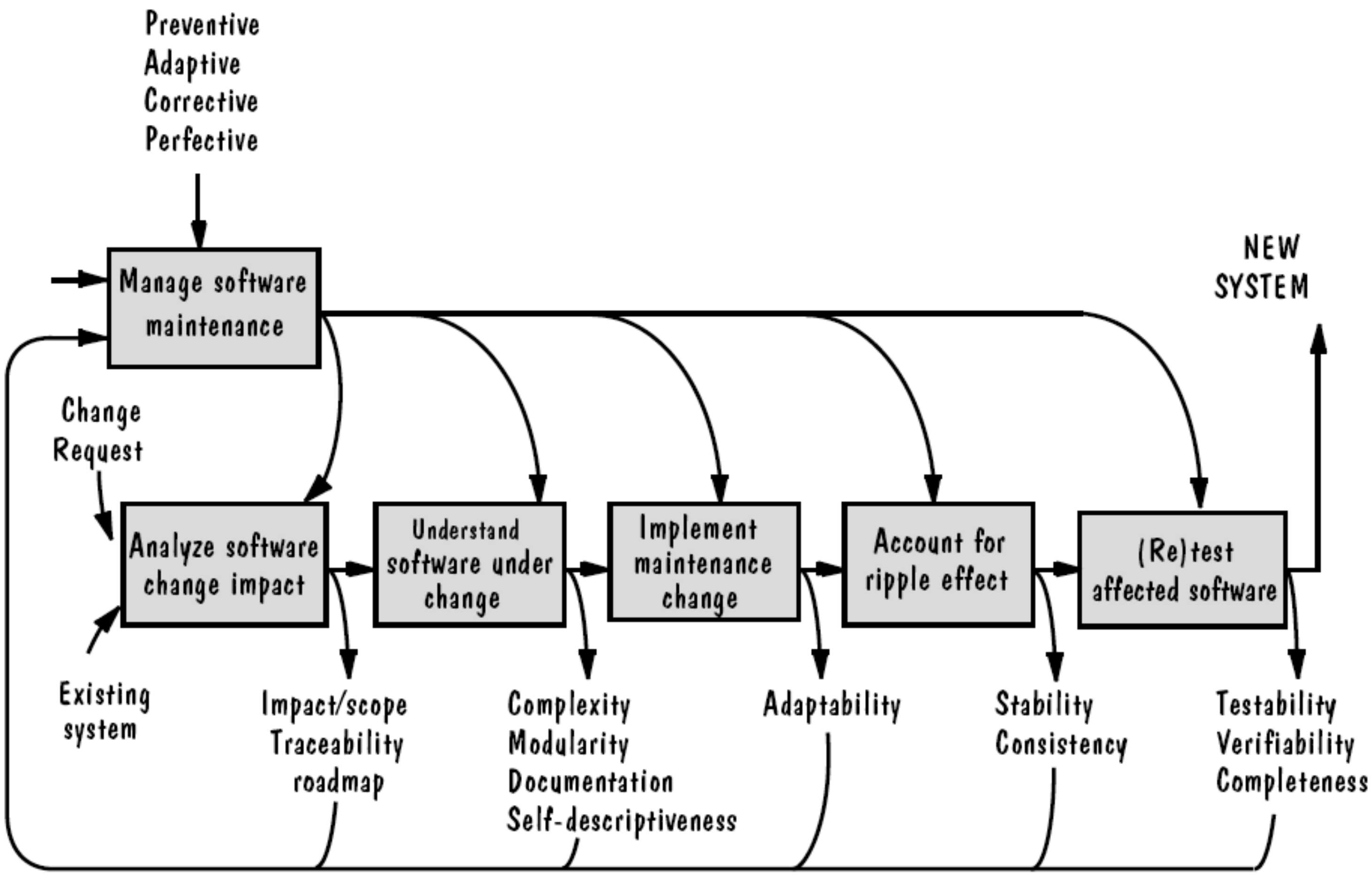
Change control issues

- *Synchronization*: When was the change made?
- *Identification*: Who made the change?
- *Naming*: What components of the system were changed?
- *Authentication*: Was the change made correctly?
- *Authorization*: Who authorized that the change be made?
- *Routing*: Who was notified of the change?
- *Cancellation*: Who can cancel the request for change?
- *Delegation*: Who is responsible for the change?
- *Valuation*: What is the priority of the change?

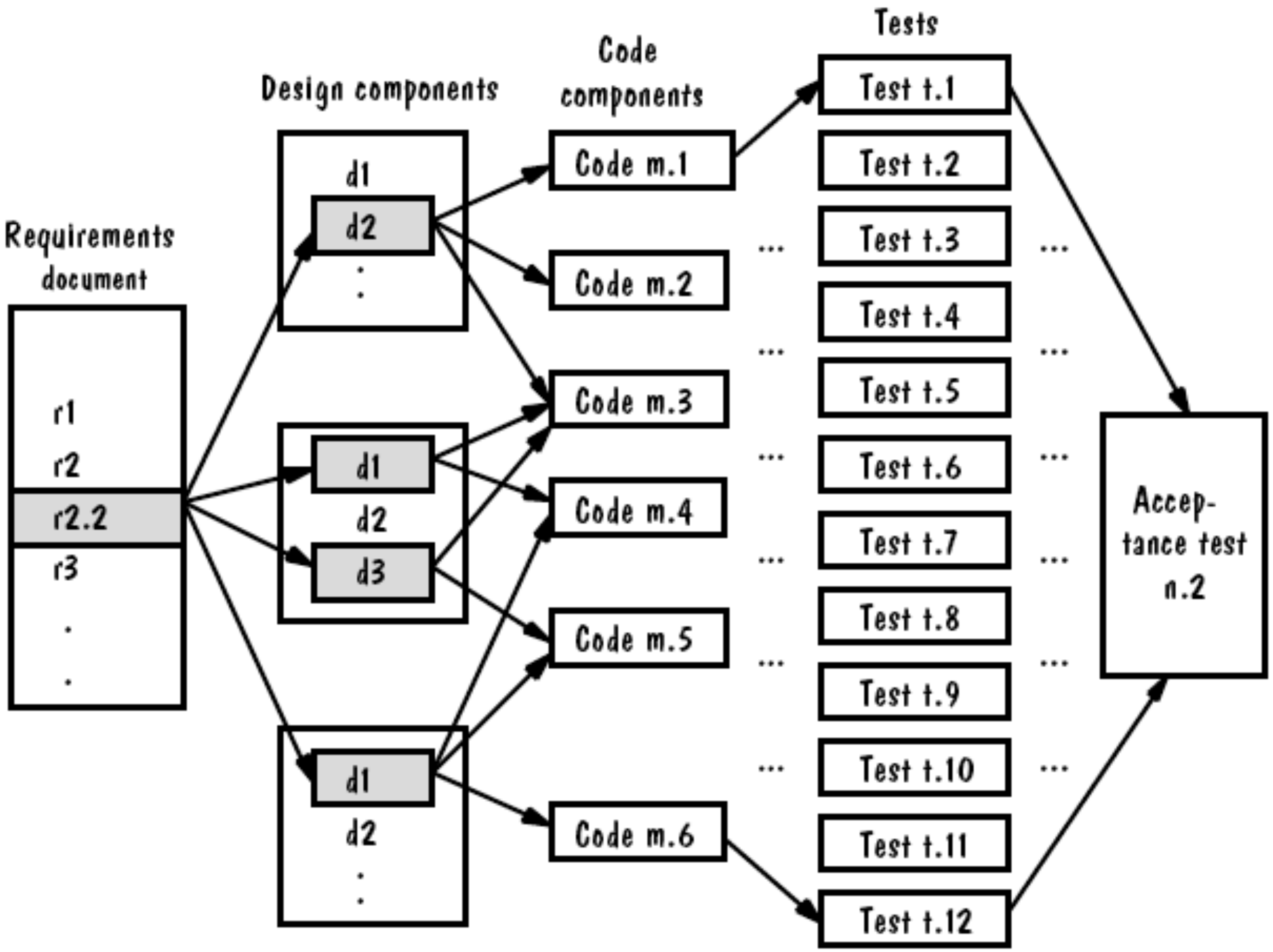
Impact analysis

- Impact analysis is the evaluation of the many risks associated with the change, including estimates of effects on ressources, effort, and schedule.
- Workproduct
  - any development artifact whose change is significant, e.g. requirements, design and code components, test cases, etc.
  - the quality of one can affect the quality of others
- Horizontal traceability
  - relationships of components across collections of workproducts
- Vertical traceability
  - relationships among parts of a workproduct

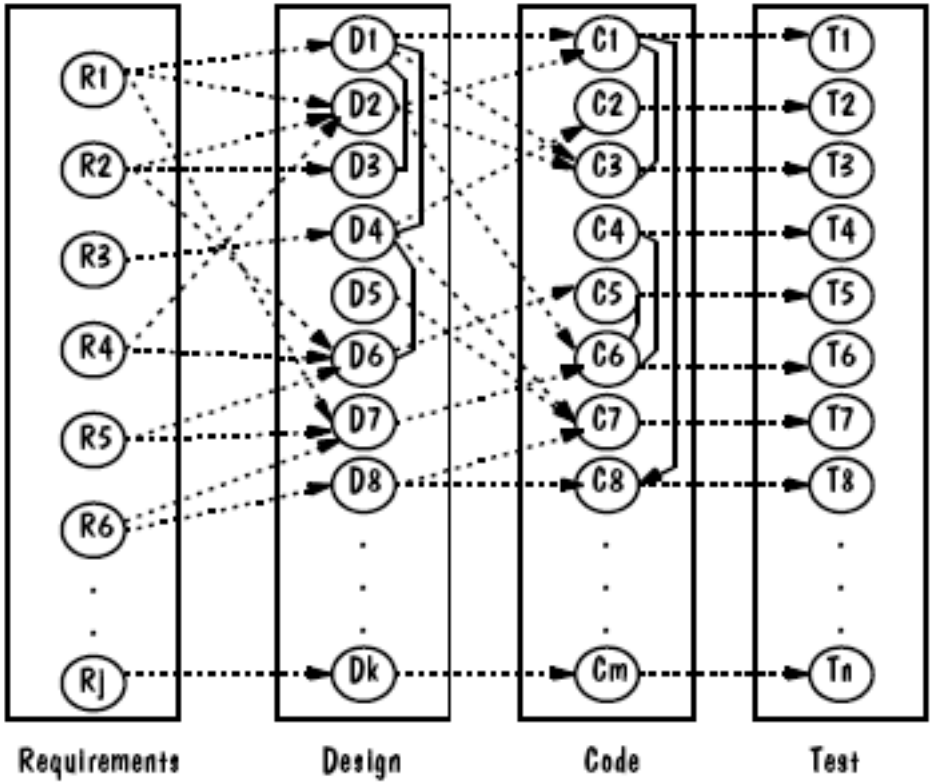
Managing software maintenance



Horizontal traceability



Underlying graph for maintenance



Automated maintenance tools

- Text editors
- File comparators
- Compilers and linkers
- Debugging tools
- Cross-reference generators
- Static code analyzers
- Configuration management repositories

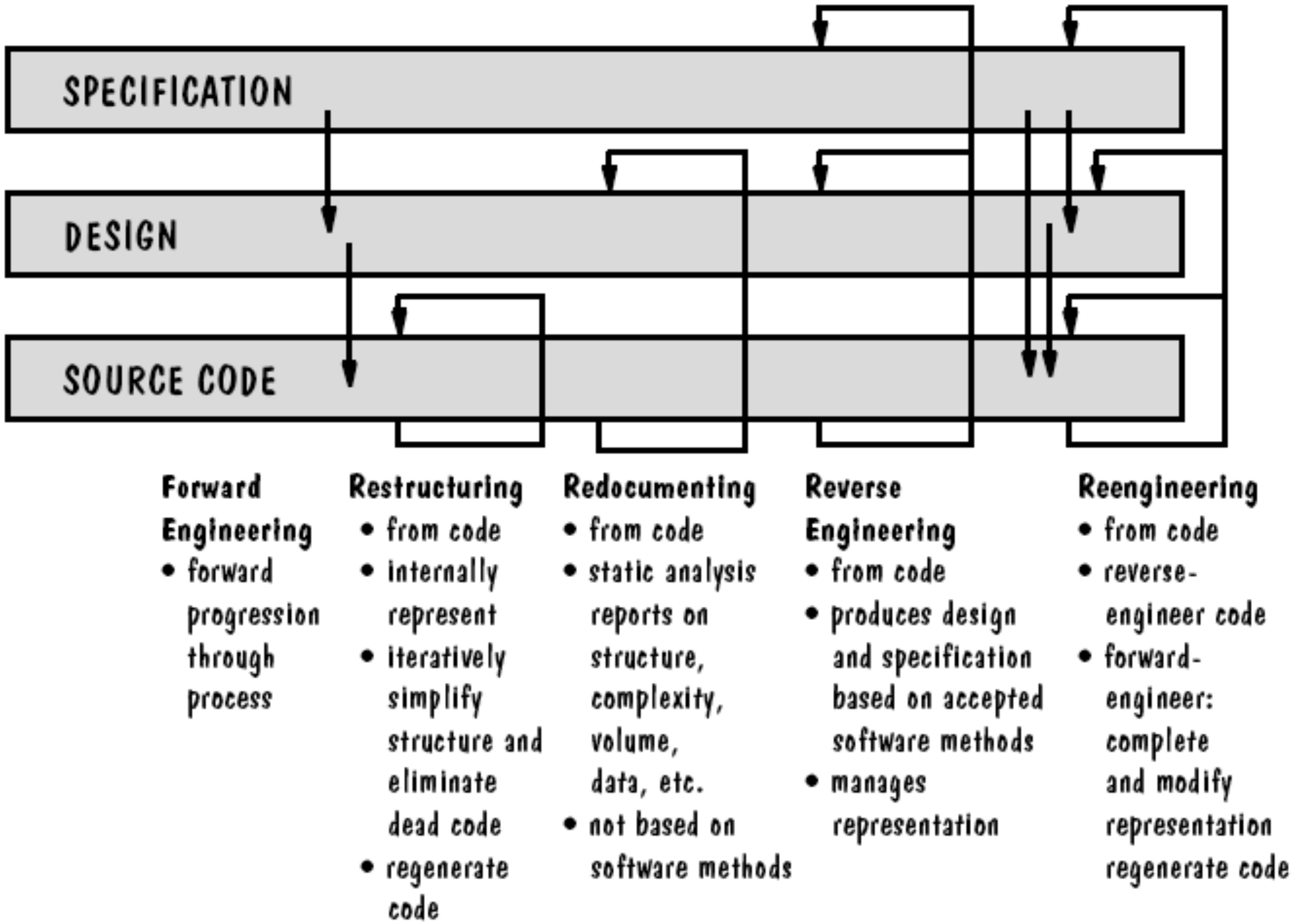
Reengineering

Universität Zürich
Institut für Informatik

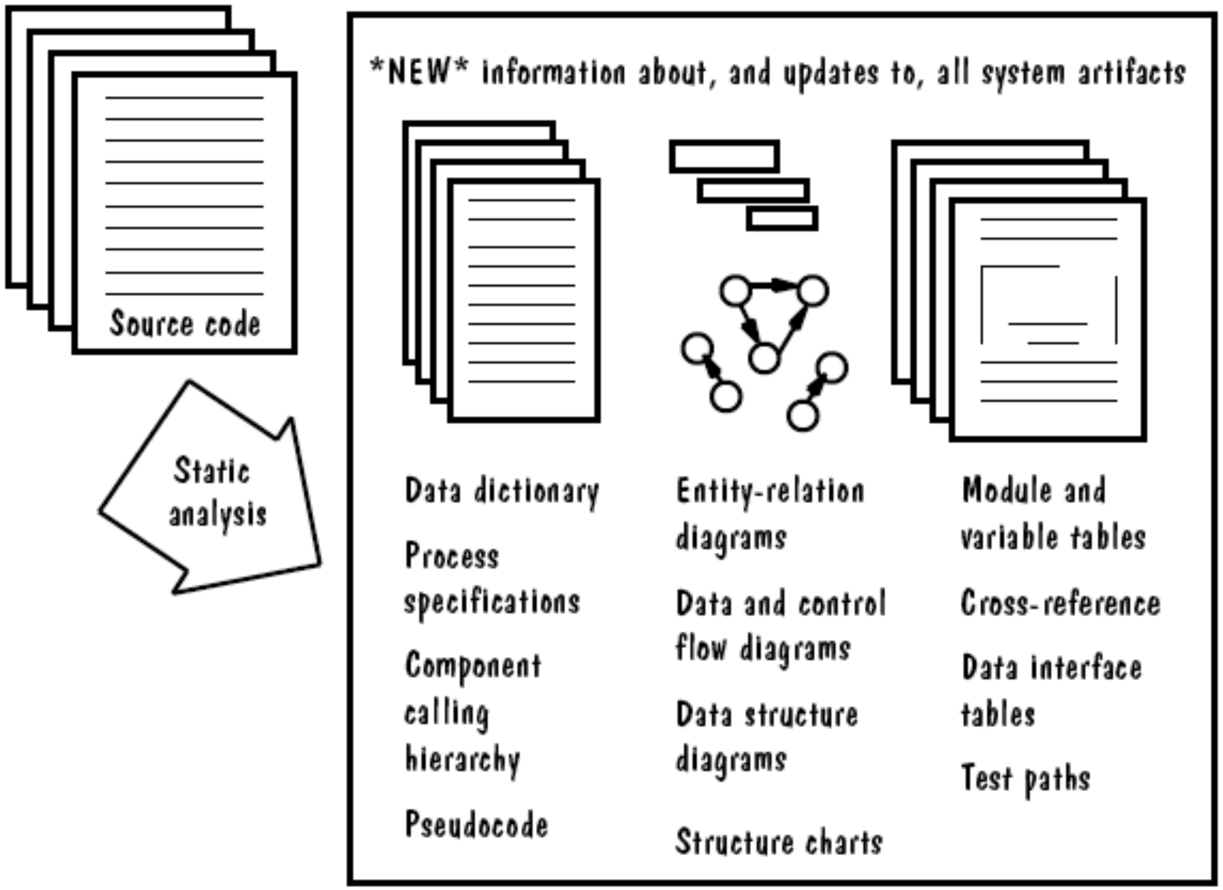
Software Rejuvenation

- *Redocumentation*: static analysis adds more information
- Restructuring: transform to improve code structure
- Reverse engineering: recreate design and specification information from the code
- Reengineering: reverse engineer and then make changes to specification and design to complete the logical model; then generate new system from revised specification and design

Taxonomy of software rejuvenation



Reverse Engineering



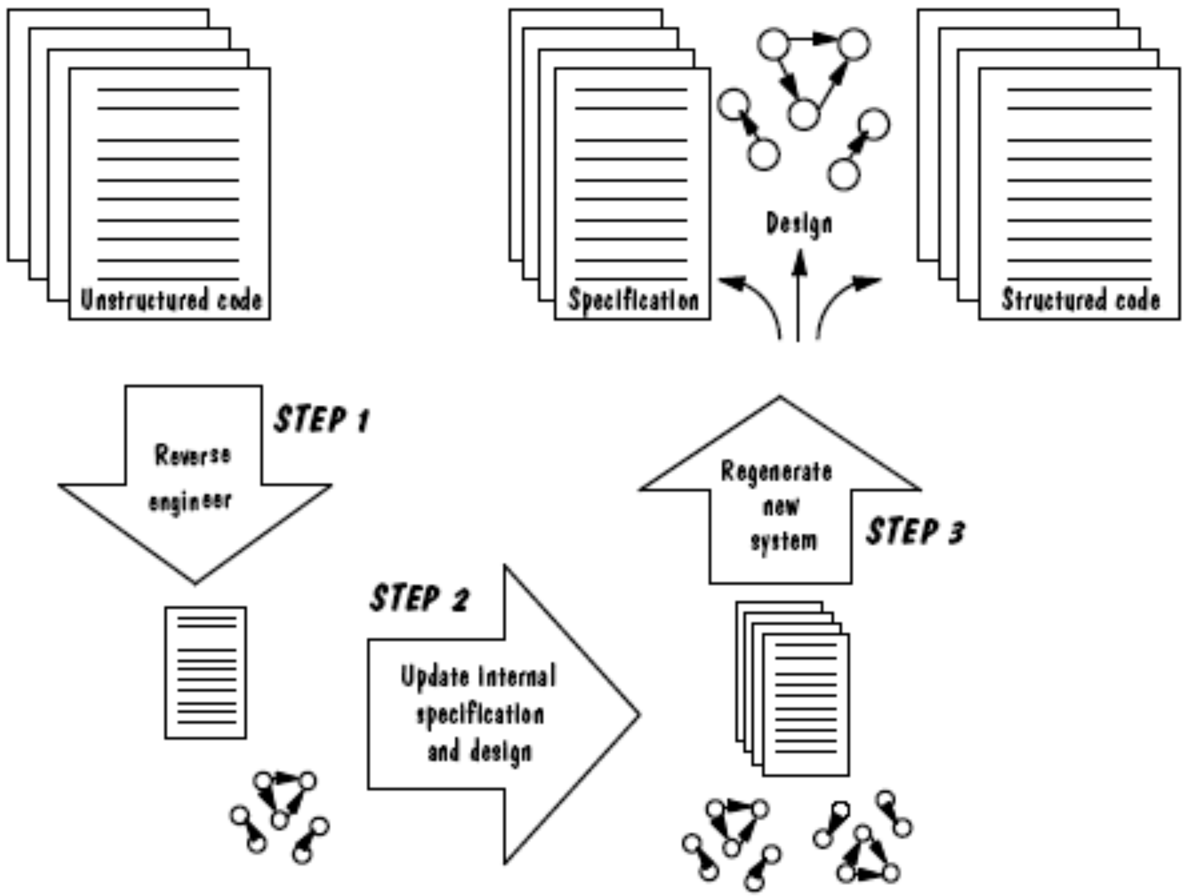
Redocumentation

- Output may include:
  - component calling relationships
  - data-interface tables
  - data-dictionary information
  - data flow tables or diagrams
  - control flow tables or diagrams
  - pseudocode
  - test paths
  - component and variable cross-references

Reengineering

- Restructuring or re-writing part or all of a legacy system plus changing its functionality according to new requirements
- 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. The system may be re-structured and re-documented.
- = Reverse Engineering + Delta + Forward Engineering

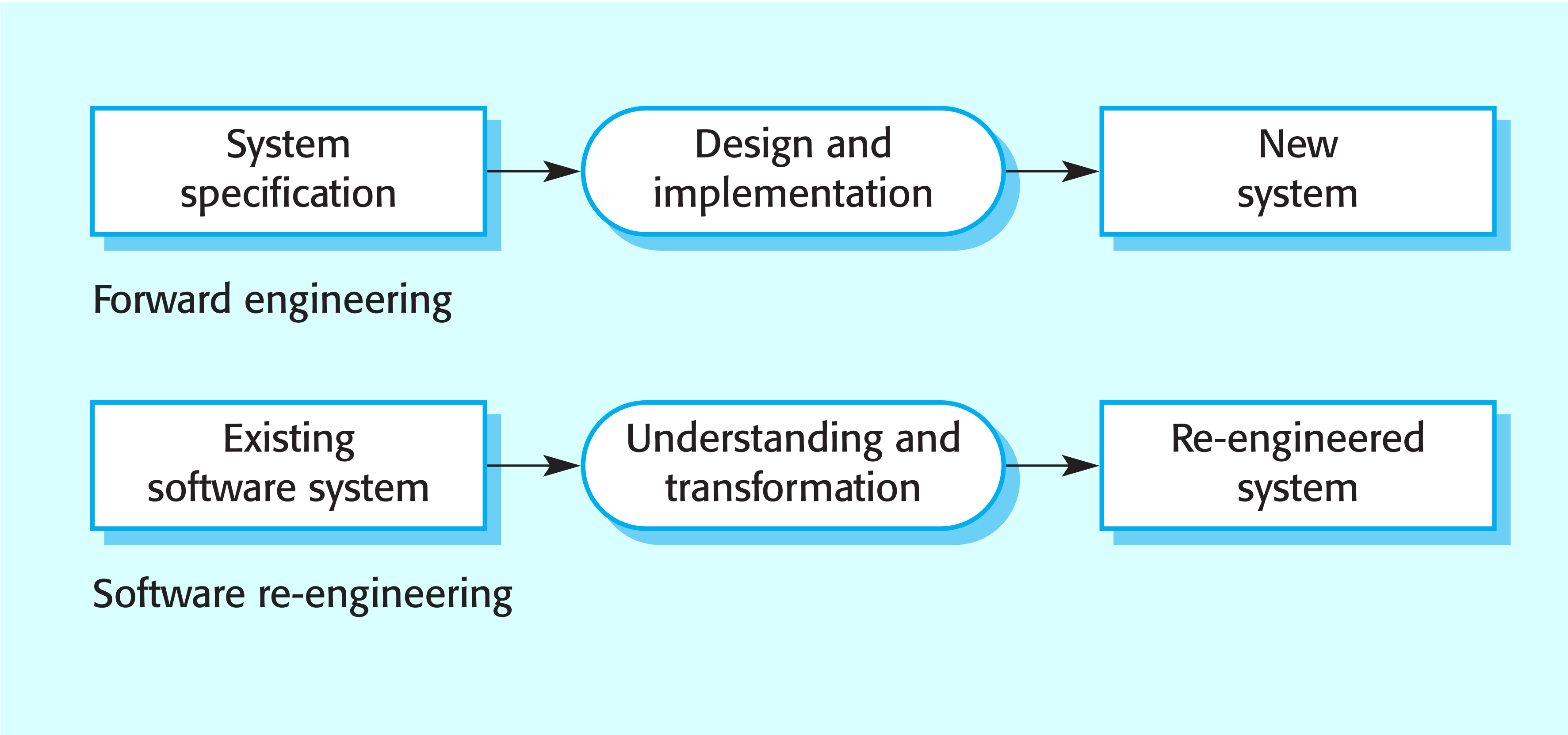
Reengineering



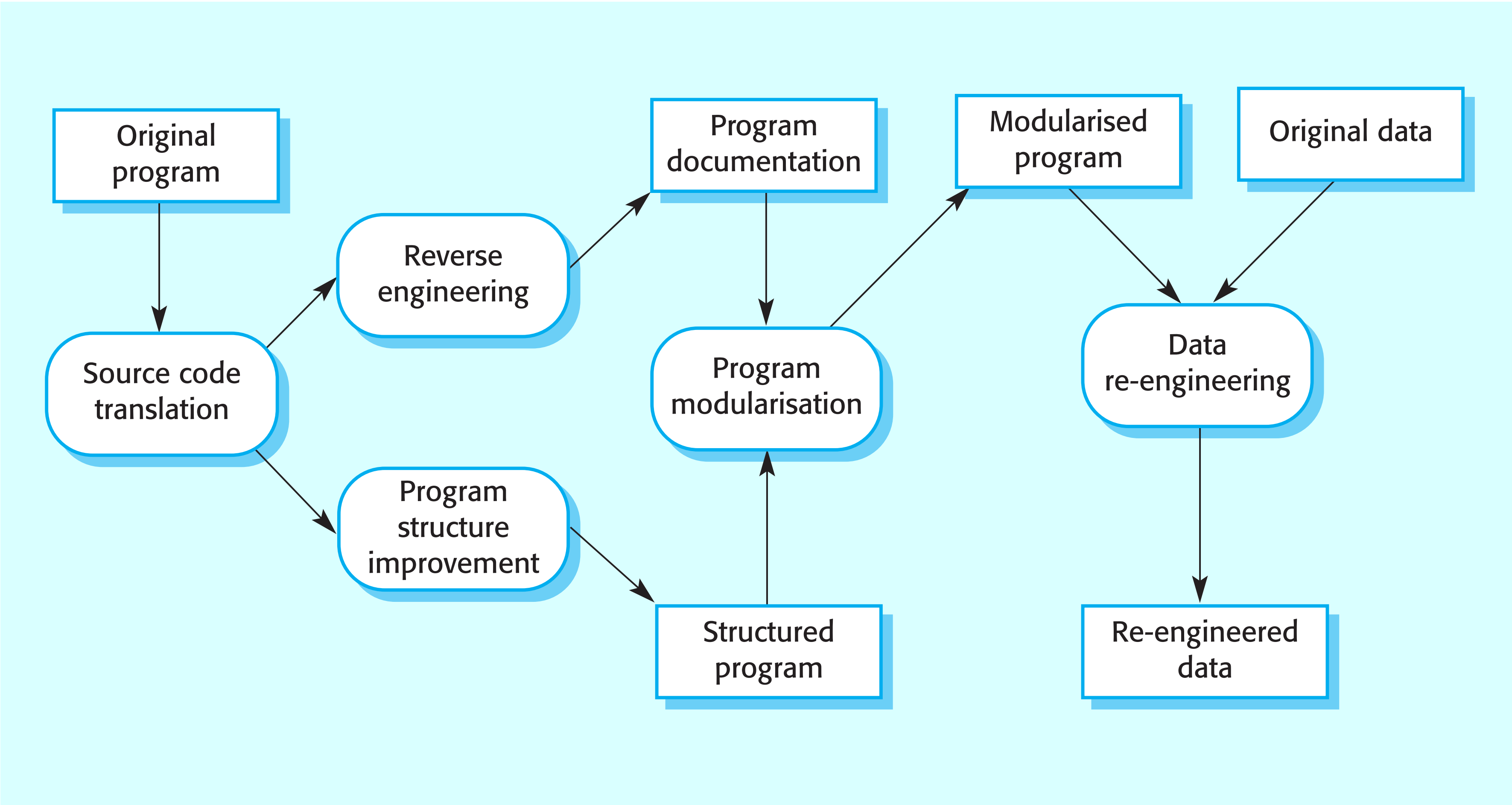
Advantages of Reengineering

- 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.
- e.g. Object-oriented Reengineering Patterns

Forward and Re-engineering



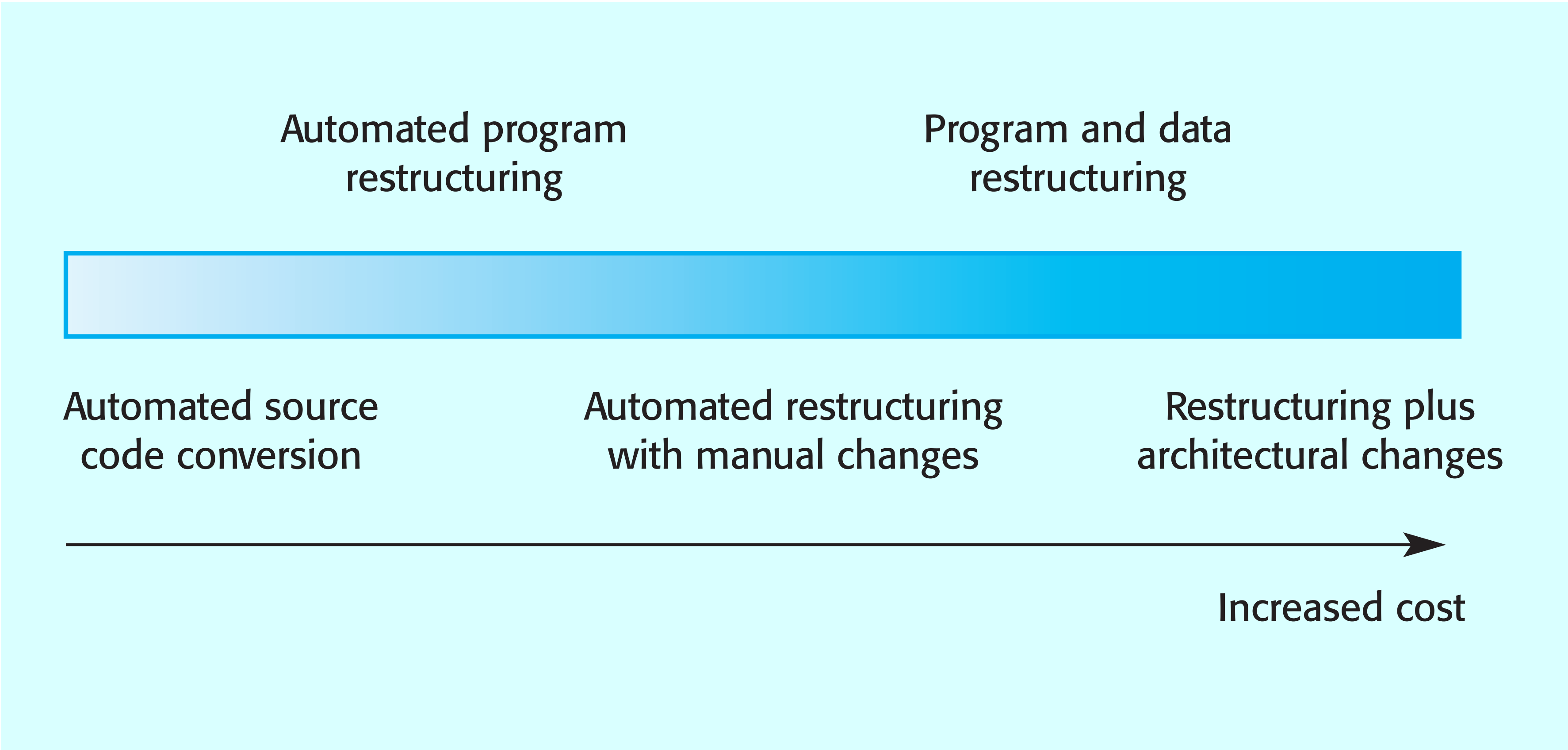
The Reengineering process



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



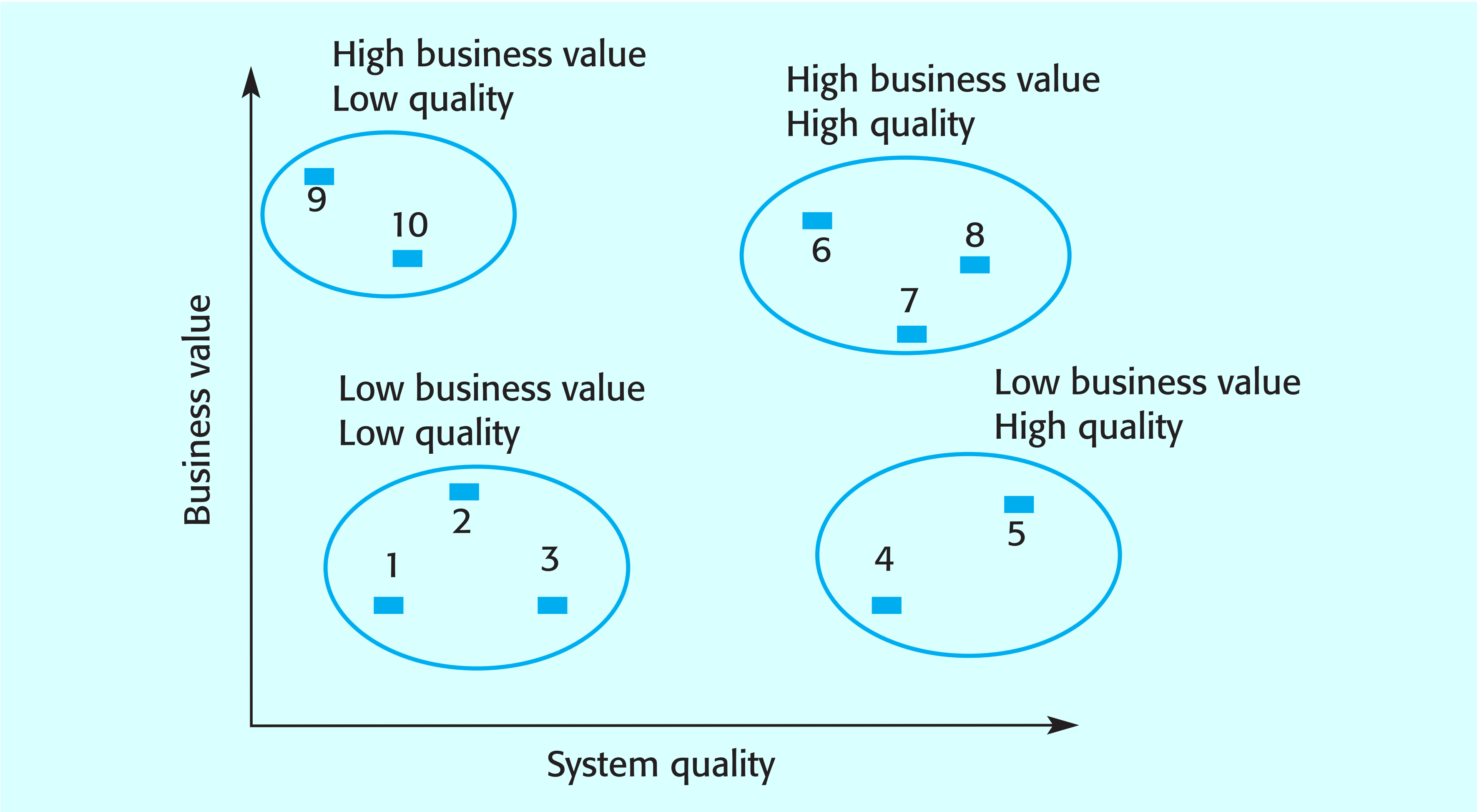
Reengineering cost factors

- 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.

Legacy system evolution

- 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.

System quality and business value



Legacy Systems

Universität Zürich
Institut für Informatik

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.

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?
- 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.

Environment assessment 1

Factor	Questions
Supplier stability	Is the supplier is 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 more modern systems.
Performance	Is the performance of the system adequate? Do performance problems have a significant effect on system users?

Environment assessment 2

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 etc. be used with current versions of the operating system? Is hardware emulation required?

Application assessment 1

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 up-to-date?
Data	Is there an explicit data model for the system? To what extent is data duplicated in different 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?

Application assessment 2

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 only a limited number of people who understand 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 number of different user interfaces used by the system;
  - The volume of data used by the system.

Summary

Summary - Key points (1)

- Software development and evolution should be a single iterative process.
- Lehman's Laws describe a number of insights into system evolution.
- Three types of maintenance are bug fixing, modifying software for a new environment and implementing new requirements.
- For custom systems, maintenance costs usually exceed development costs.

Summary - Key points (2)

- The process of evolution is driven by requests for changes from system stakeholders.
- Software re-engineering is concerned with re-structuring and re-documenting software to make it easier to change.
- The business value of a legacy system and its quality should determine the evolution strategy that is used.

References

- S.L. Pfleeger, J.M. Atlee. Software Engineering: Theory and Practice, 3rd edition, Pearson Education, 2006.
- I. Sommerville. Software Engineering, 7th edition, Pearson Education, 2004.
- S. Demeyer, S. Ducasse, O. Nierstrasz. Object-Oriented Reengineering Patterns, Morgan-Kaufmann 2003. http://www.iam.unibe.ch/~scg/OORP/
- International Conference on Software Maintenance
- International Conference on Program Comprehension, IEEE
- International Workshop on Principles of Software Evolution, IEEE