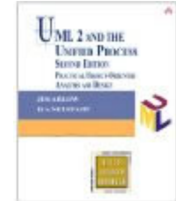
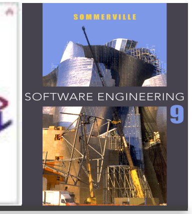




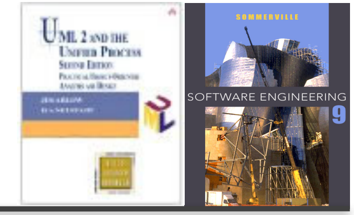
Lecture 10

OPERATION, MAINTENANCE AND EVOLUTION

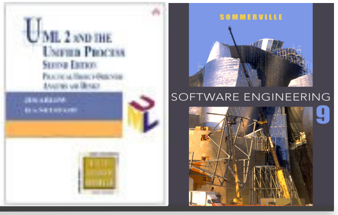
PB007 Software Engineering I
Faculty of Informatics, Masaryk University

Fall 2016

Topics covered



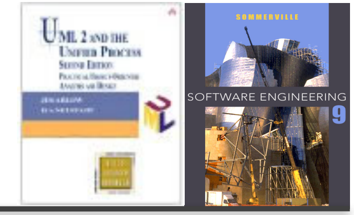
- Evolution processes
  - Change processes for software systems
- Lehman's laws
  - Understanding software evolution dynamics
- Software maintenance
  - Making changes to operational software systems
- Legacy system management
  - Making decisions about software change



Evolution Processes

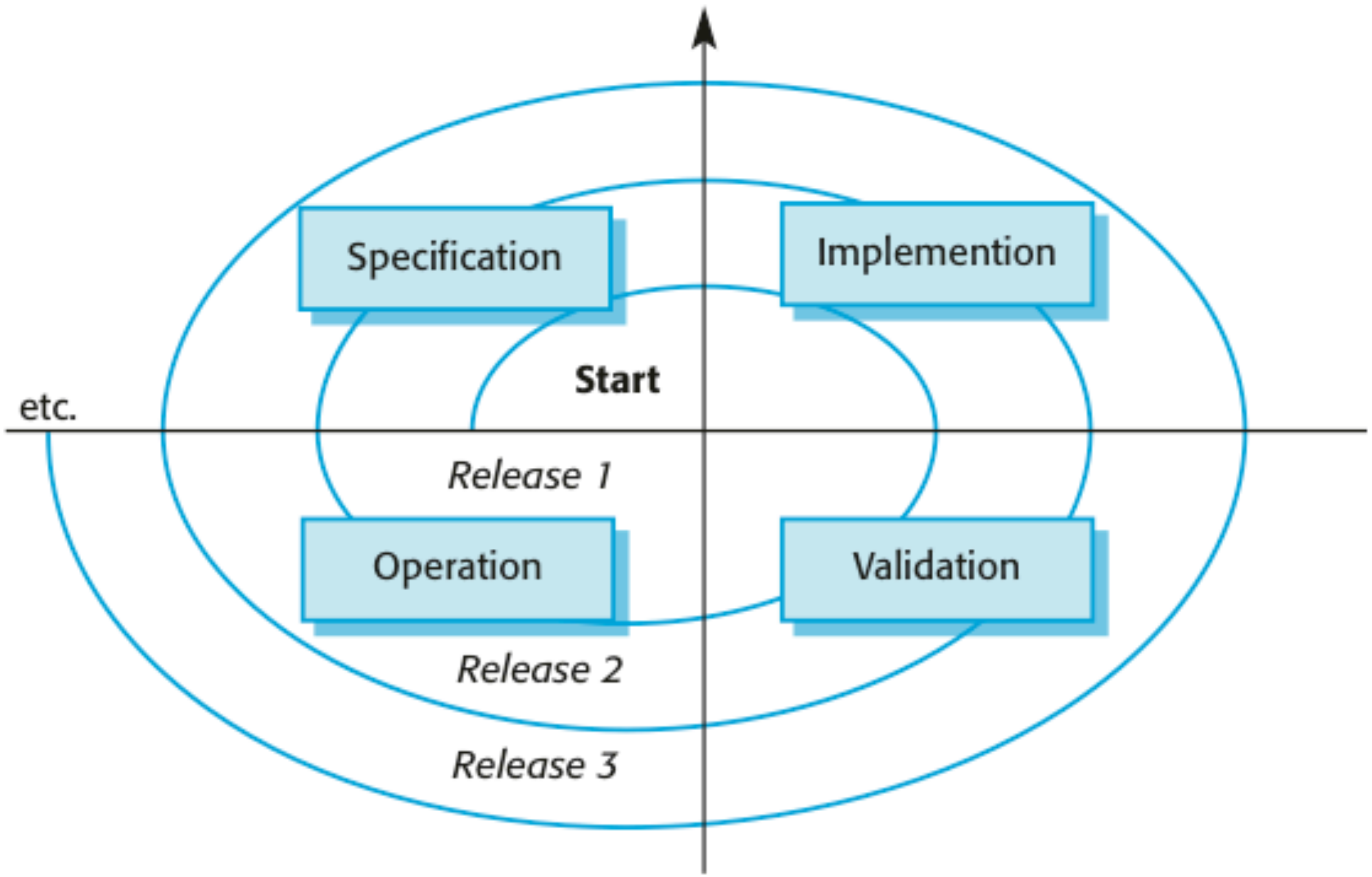
Lecture 10/Part 1

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.
- For custom systems, the **costs** of software maintenance usually **exceed the software development costs**.
- A key problem for all organizations is implementing and **managing change** to their existing software systems.

A spiral model of development & evolution

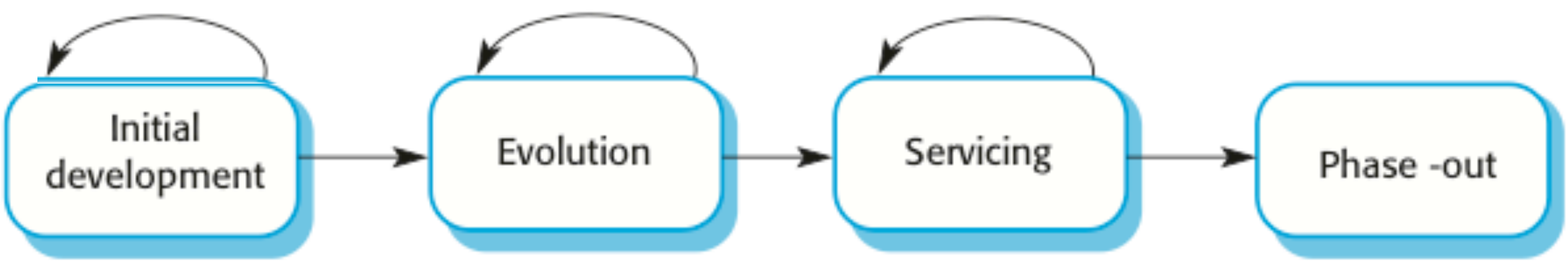


Evolution and servicing

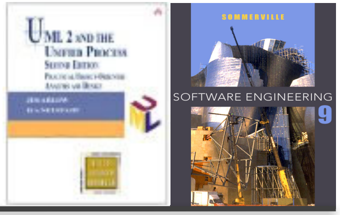
✧ Evolution
- New functionality added, faults repaired.

✧ Servicing
- Faults repaired, no new functionality added.

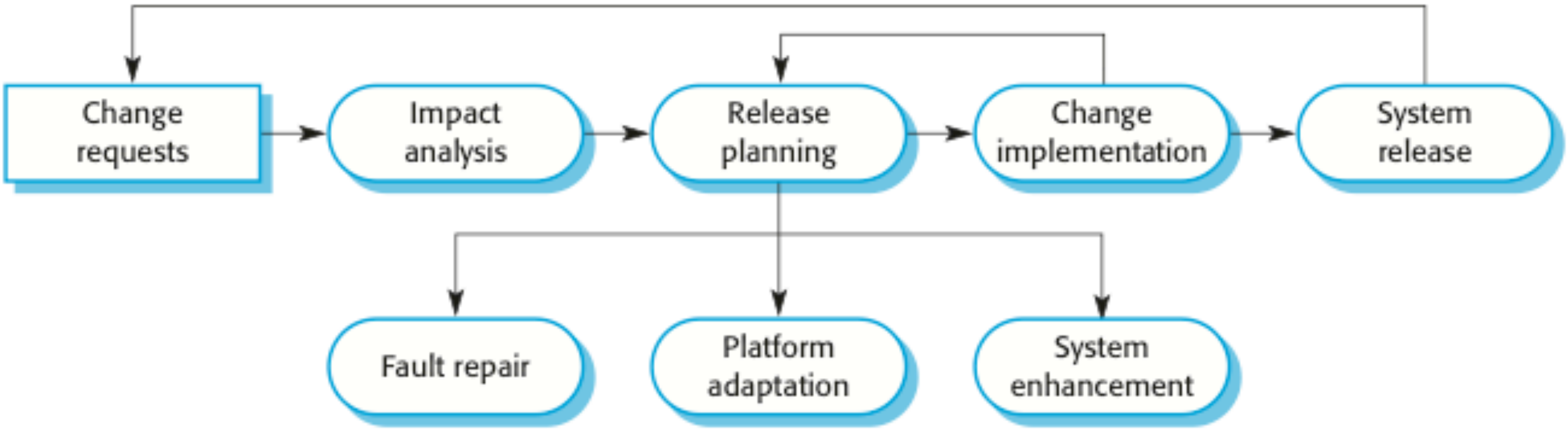
✧ Phase-out
- The software still in use but no further changes are made to it.



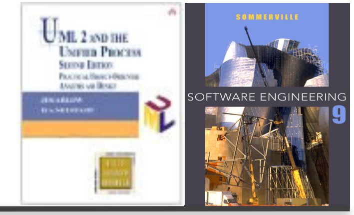
Evolution processes



- ✧ Proposals for **change** are the **driver** for SW 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.

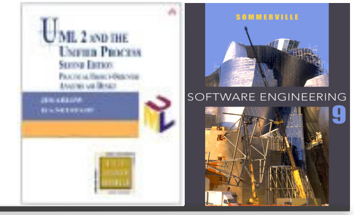


More on change implementation

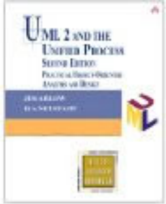


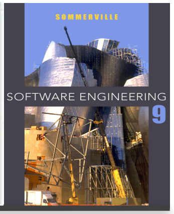
- **Iteration of the development process** where the revisions to the system are designed, implemented and tested.
  - May be done by a **different team**, not the original developers.
  - Involves **program understanding**, especially if no original developers are involved.
  - During the program understanding phase, one has to understand **how the proposed change might affect the program**.
- **Urgent changes** may have to be implemented without going through all stages of the evolution process.
- **Technical debt** may be created and must be managed.

Agile methods and evolution



- Agile methods are based on 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.
- **Handover problems**
  - What if the development team is agile and the evolution team is not?
  - What if the evolution team is agile and the development team is not?

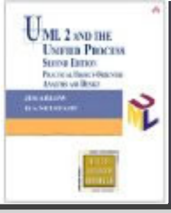


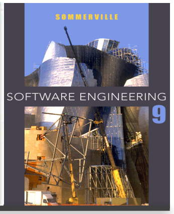


Lehman's laws

Lecture 10/Part 2

Lehman's laws

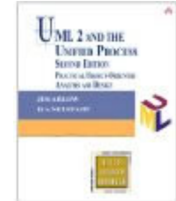
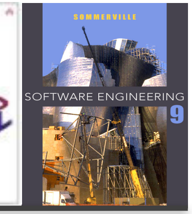




- **Program evolution dynamics** is the study of the processes of system change.
- After several major empirical studies **Lehman and Belady** proposed observational 'laws' which applied to systems under evolution.

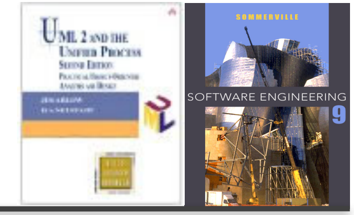
Law	Description
Continuing change	A program that is used in a real-world environment **must necessarily change**, or else become progressively **less useful** in that environment.
Increasing complexity	As an evolving program changes, its structure tends to become more complex. Extra resources must be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

Lehman’s laws

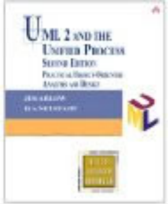



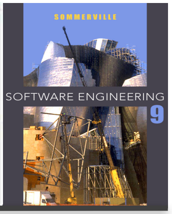
Law	Description
Organizational stability	Over a program’s lifetime, its **rate of development is approximately constant** and independent of the resources devoted to system development.
Conservation of familiarity	Over the lifetime of a system, the incremental change in each release shall be approximately constant, to allow system users to **maintain mastery of its usage**.
Continuing growth	The functionality offered by systems has to continually increase to **maintain user satisfaction**.
Declining quality	The quality of systems will decline unless they are modified to reflect **changes in their operational environment**.
Feedback system	Evolution processes incorporate multiagent, multiloop 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 organisations.
  - Confirmed in early 2000's by work by Lehman on the FEAST project.
- It is not clear how they should be modified for
  - Generic software products;
  - Systems that incorporate a significant number of COTS components;
  - Small organisations;
  - Small and medium sized systems.

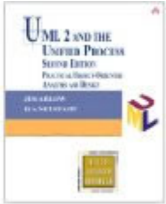


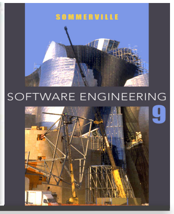


Software Maintenance

Lecture 10/Part 3

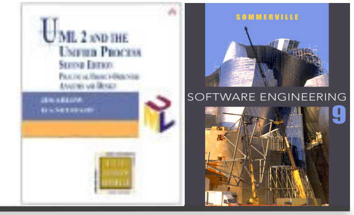
Software maintenance





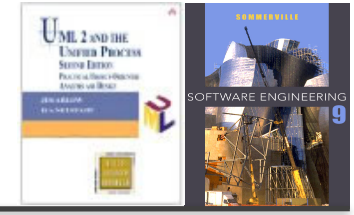
- Modifying a program after it has been put into use.
- The term is mostly used for changing **custom software**. Generic software products are said to evolve to create new versions.
- Maintenance **does not** normally **involve major changes** to the system's architecture.
- Changes are implemented by modifying existing components and adding new components to the system.

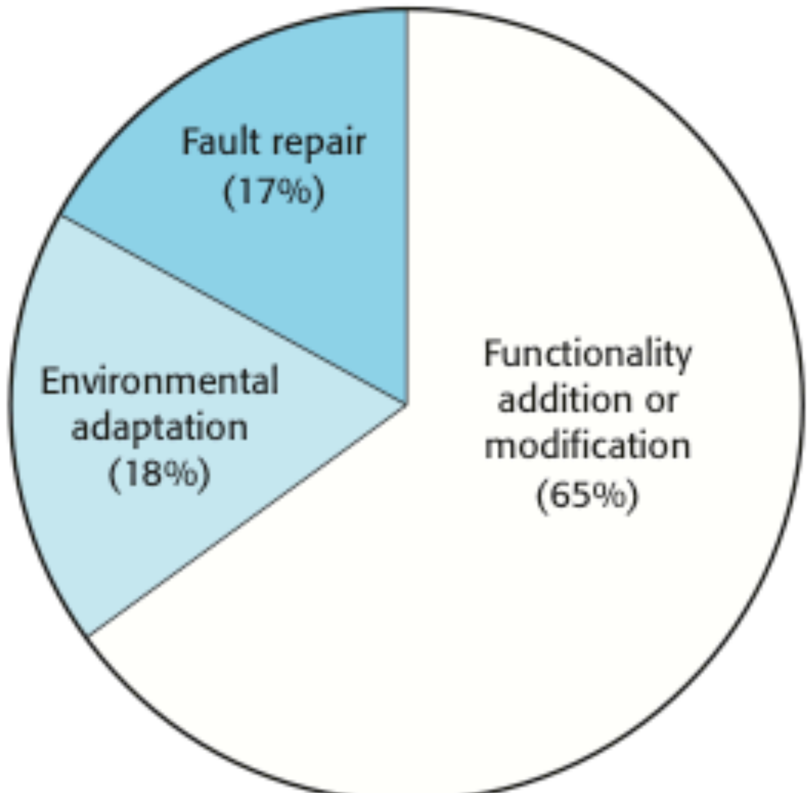
Types of maintenance



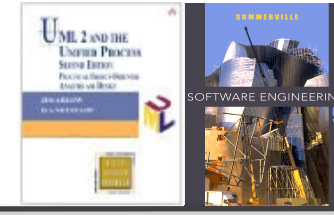
- **Corrective:** Maintenance to repair software faults
  - Changing a system to correct deficiencies in the way meets its requirements.
- **Adaptive:** Maintenance to adapt software to a different operating environment
  - Changing a system so that it operates in a different environment (computer, OS, etc.) from its initial implementation.
- **Evolutionary:** Maintenance to add to or modify the system’s functionality
  - Modifying the system to satisfy new requirements.

Maintenance effort distribution



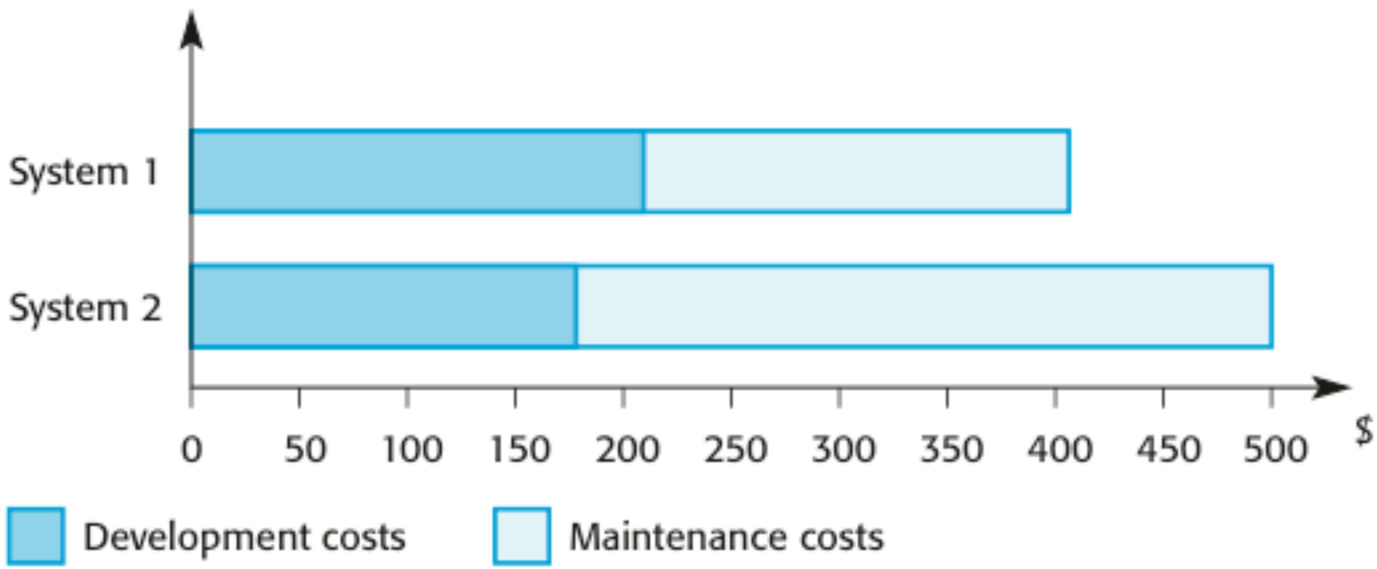


Maintenance costs

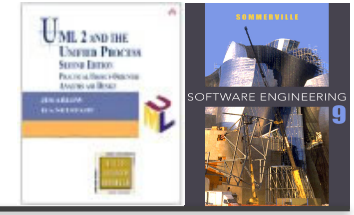


- Usually greater than development costs (1* to 20* depending on the application).
  - Experience with custom information systems shows that around 20% of development costs needs to be allocated to maintenance every year (within the first five years).
- Affected by both technical and non-technical factors.
- 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

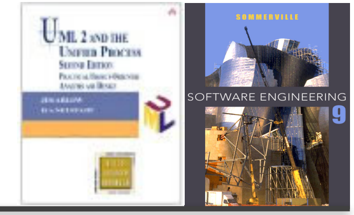


Maintenance cost factors



- 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.

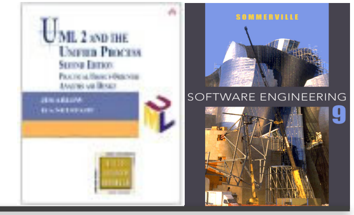
Maintenance prediction

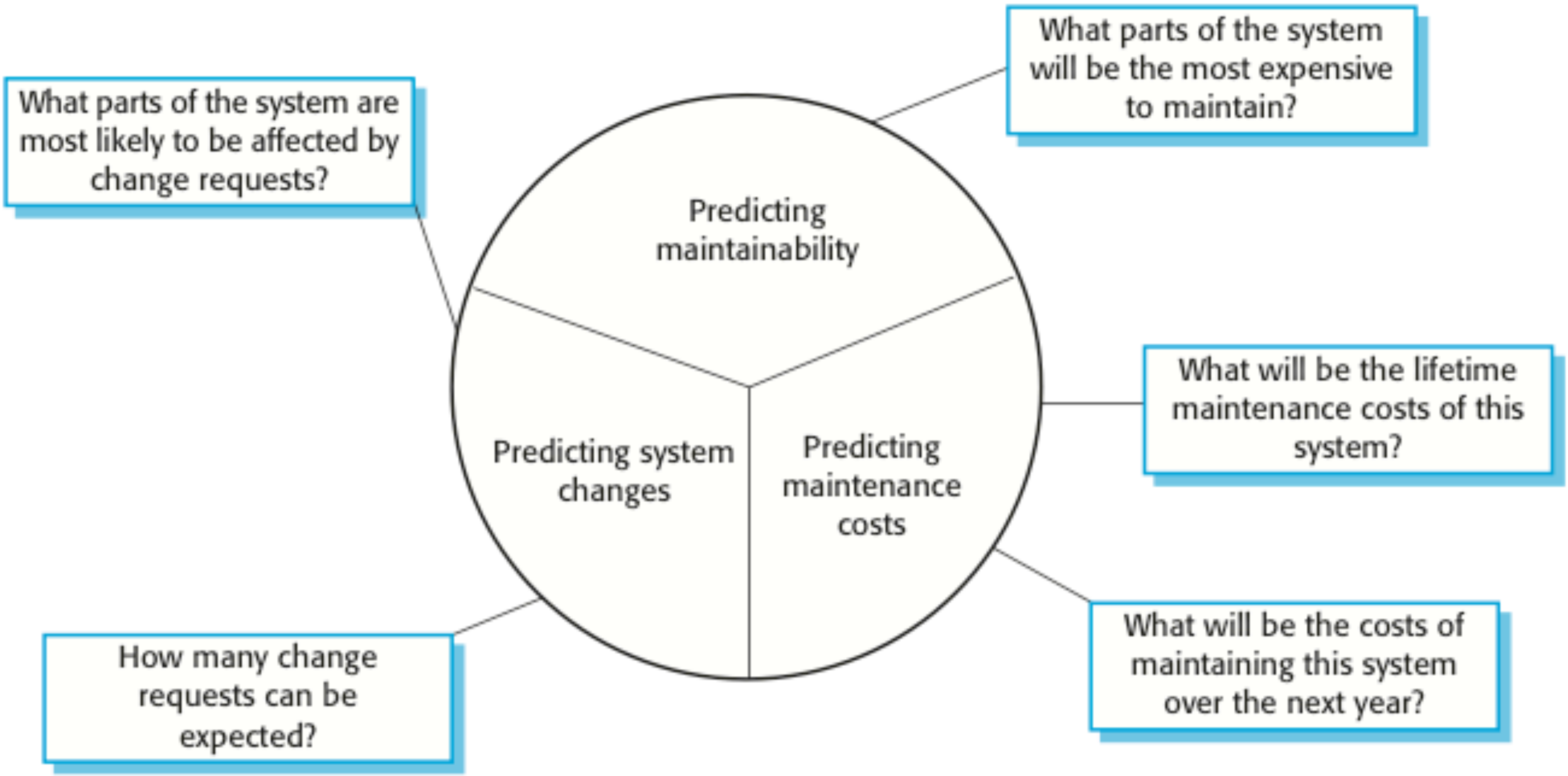


✧ Maintenance prediction is concerned with assessing which parts of the system may cause problems and have high maintenance costs

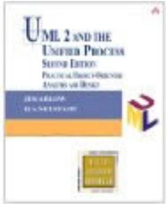
- Change acceptance **depends on the maintainability** of the components affected by the change;
- Implementing **changes degrade the system** and reduces its maintainability;
- Maintenance costs depend on the number of changes and costs of change depend on maintainability.

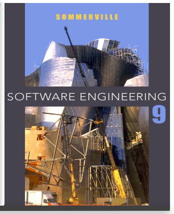
Maintenance prediction





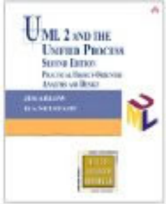
Change prediction

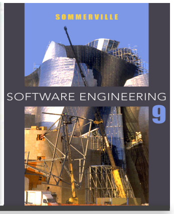




- Predicting the number of changes requires an **understanding of the relationships** between the system and its environment.
- Tightly coupled systems require changes **whenever the environment is changed**.
- Factors influencing this relationship are
  - Number and complexity of system **interfaces**;
  - Number of inherently **volatile system requirements**;
  - The **business processes** where the system is used.

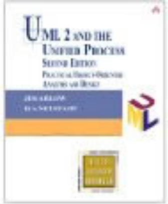
Complexity metrics

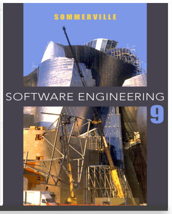




- ✧ Predictions of maintainability can be made by assessing the complexity of system components.
- ✧ Studies have shown that most maintenance effort is spent on a **relatively small number** of system components.
- ✧ Complexity depends on
  - Complexity of control structures;
  - Complexity of data structures;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

Process metrics





✧ Process 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.

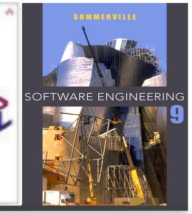
✧ If any or all of these **is increasing**, this may indicate a **decline in maintainability**.

System reengineering

- 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.
- Reengineering involves adding effort to **make them easier to maintain**. The system may be re-structured and re-documented.

- How does reengineering relate to refactoring?
- How does reengineering relate to technical debt?

Advantages of reengineering



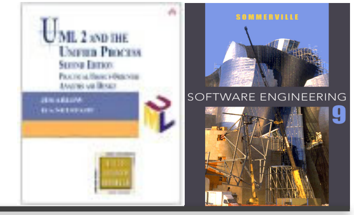
✧ Reduced risk

- There is a high risk in new software development. There may be development problems, staffing problems and specification problems.

✧ Reduced cost

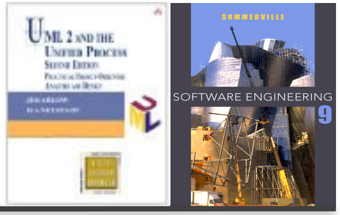
- The cost of reengineering is often significantly less than the costs of developing new software.

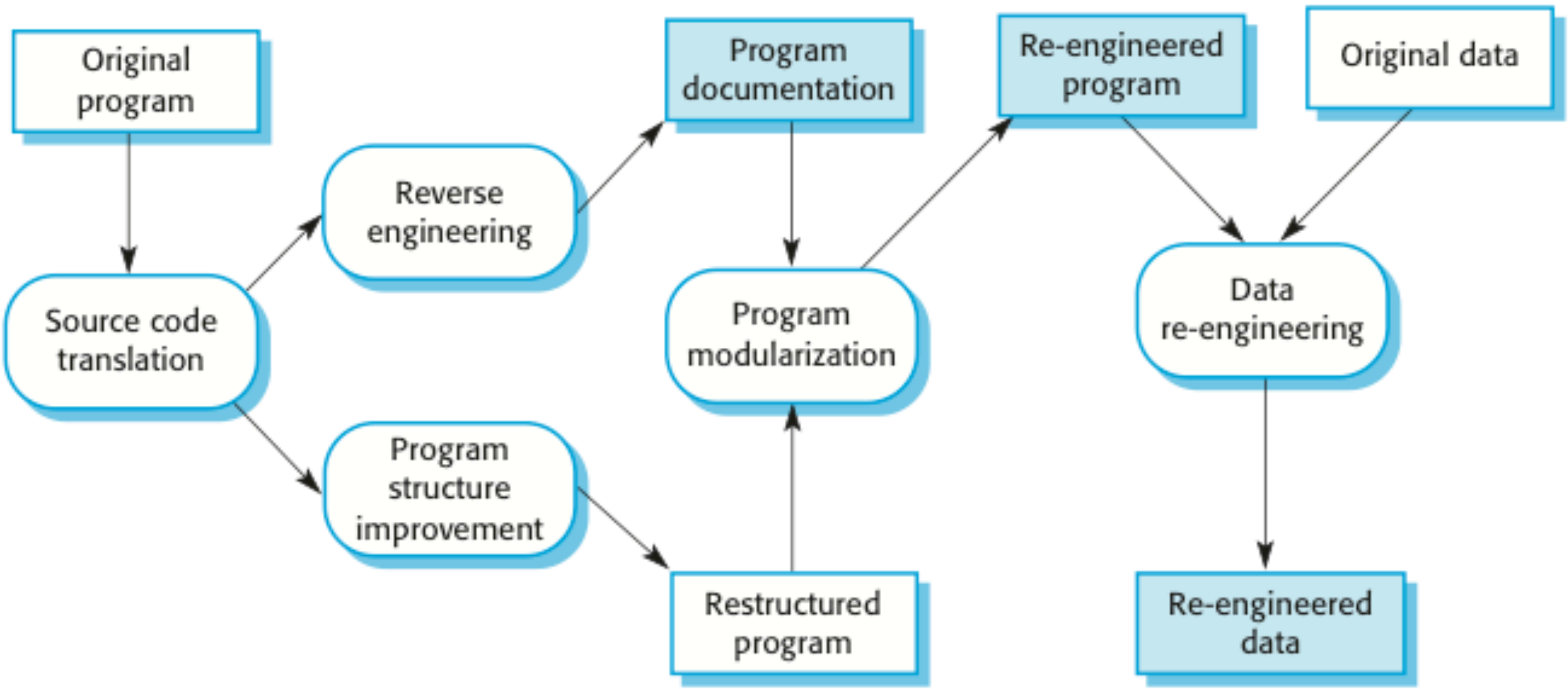
Reengineering activities



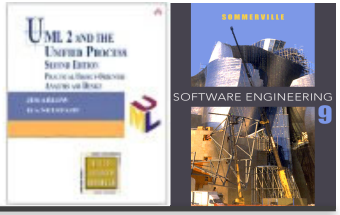
- ✧ Source code translation
  - Convert code to a new language.
- ✧ Reverse engineering
  - Analyse the program to understand it;
- ✧ Program structure improvement
  - Restructure automatically for understandability;
- ✧ Program modularisation
  - Reorganise the program structure;
- ✧ Data reengineering
  - Clean-up and restructure system data.

The reengineering process example

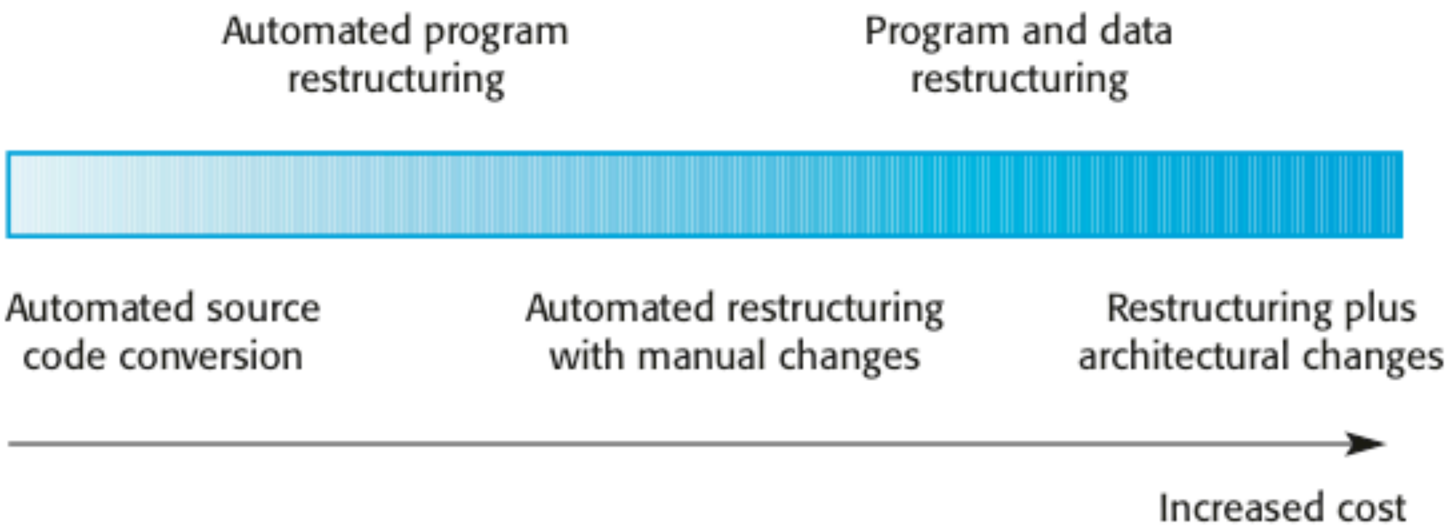




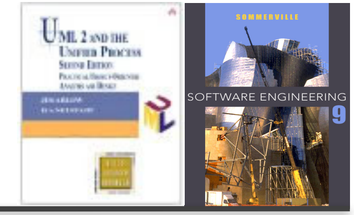
Reengineering cost factors



- 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.

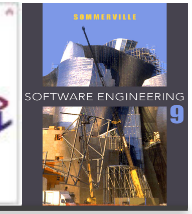


Preventative maintenance by refactoring



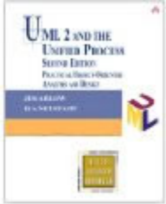
- Refactoring is the process of making improvements to a program to **slow down degradation** through change.
- You can think of refactoring as ‘**preventative maintenance**’ that reduces the problems of future change.
- Refactoring involves modifying a program **to improve its structure**, **reduce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

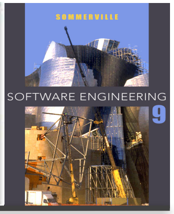
Refactoring and reengineering



- **Reengineering** takes place after a system has been maintained for some time and maintenance costs are increasing. You use automated tools to process and reengineer 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.

Examples of ‘Bad Smells’ in program code





✧ Duplicate code

- Can be implemented as a single method or function that is called as required.

✧ Long methods

- Can be redesigned as a number of shorter methods.

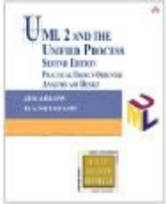
✧ Switch (case) statements

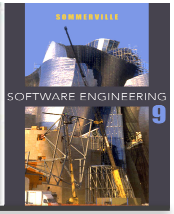
- Whenever the switch depends on the type of a value, consider using polymorphism to achieve the same thing.

✧ Data clumping

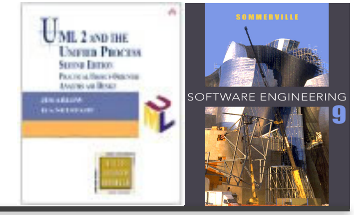
- Data clumps (repeating groups of data items) can often be encapsulated into an object.

Key points





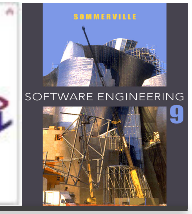
- There are 3 types of software maintenance, namely **bug fixing**, modifying software to **work in a new environment**, and implementing **new or changed requirements**.
- **Software reengineering** is concerned with re-structuring and re-documenting software to make it easier to understand and change.
- **Refactoring**, making program changes that preserve functionality, is a form of preventative maintenance.



Legacy System Management

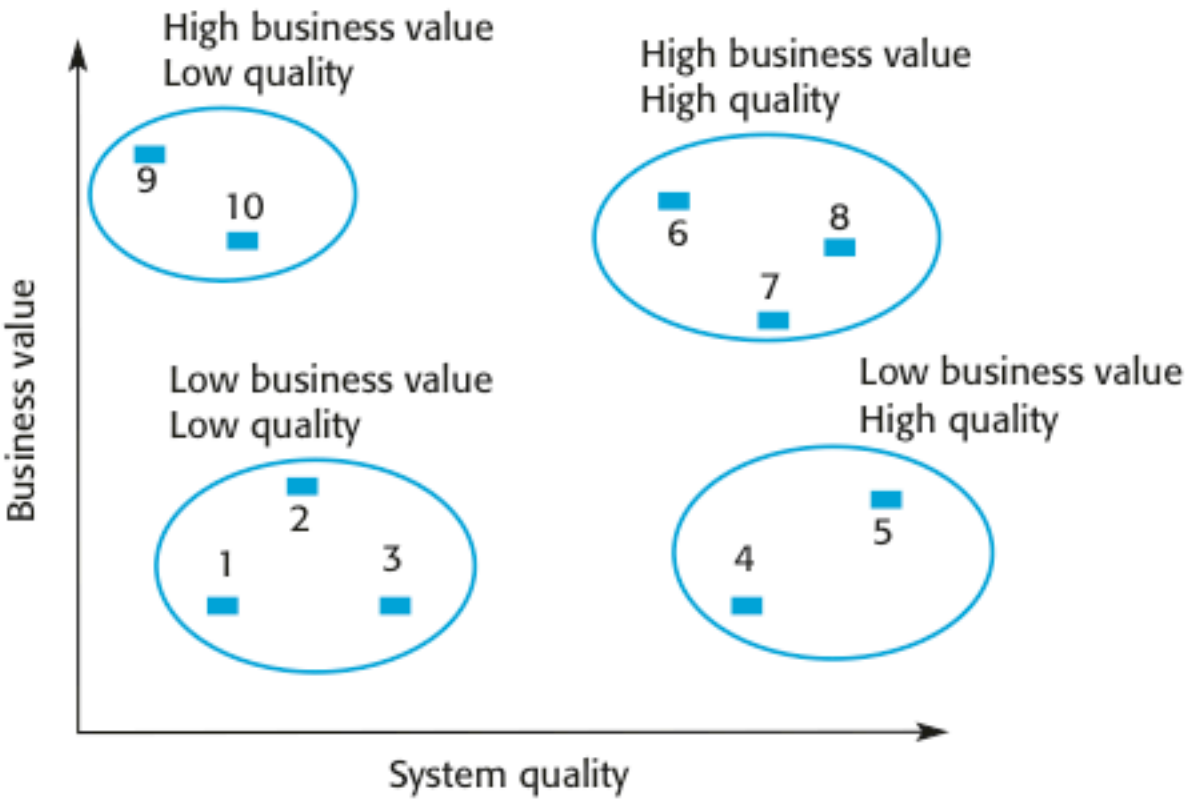
Lecture 10/Part 4

Legacy system management

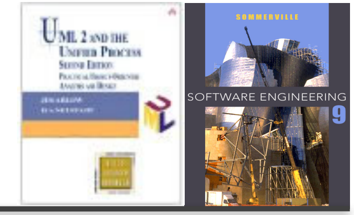


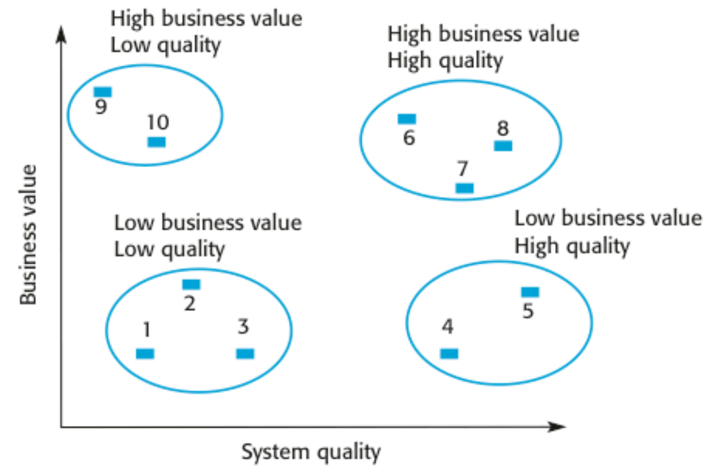
- Organisations that rely on legacy systems must choose a strategy for evolving these systems
  - **Scrap:** Scrap the system completely and modify business processes so that it is no longer required;
  - **Maintain:** Continue maintaining the system;
  - **Reengineer:** Transform the system by reengineering to improve its maintainability;
  - **Replace:** Replace the system with a new system.
- The strategy chosen should depend on the **system quality** and its **business value**.

An example of a legacy system assessment



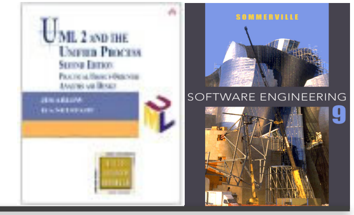
Legacy system categories





- ✧ 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 reengineered or replaced if a suitable system is available.
- ✧ High quality, low business value
  - Scrap or increase the business value – see on later slides.
- ✧ High quality, high business value
  - Continue in operation using normal system maintenance.

Business value assessment

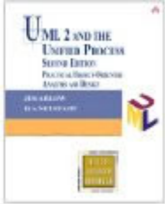


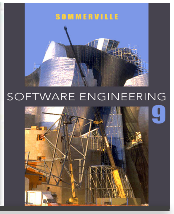
✧ Assessment should take different viewpoints into account

- System end-users;
- Business customers;
- Line managers;
- IT managers;
- Senior managers.

✧ Interview different stakeholders and collate results.

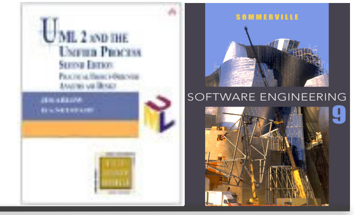
Issues in business value assessment





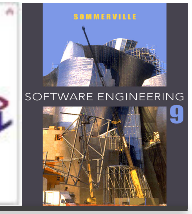
- The use of the system
  - If systems are only used occasionally or by a small number of people, they may have a low business value.
- The business processes that are supported
  - A system may have a low business value if it forces the use of inefficient business processes.
- System dependability
  - 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



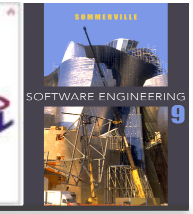
- Application assessment
  - What is the quality of the application software system? How expensive it is to maintain?
- Environment assessment
  - How effective is the system's environment? How expensive it is to maintain?
- You may collect quantitative data to make an assessment of the quality of the 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.

Factors used in application assessment



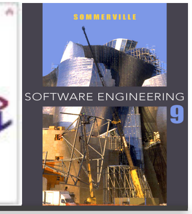
Factor	Questions
Understandability	How difficult is it to understand the source code of the current system? How complex are the control structures that are used? Do variables have meaningful names that reflect their function?
Documentation	What system documentation is available? Is the documentation 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



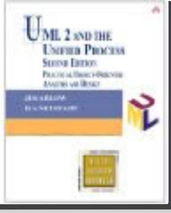
Factor	Questions
Programming language	Are modern compilers available for the programming language used to develop the system? Is the programming language still used for new system development?
Configuration management	Are all versions of all parts of the system managed by a configuration management system? 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?

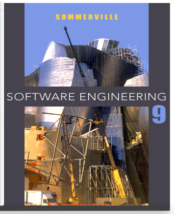
Factors used in environment assessment



Factor	Questions
Supplier stability	Is the supplier still in existence? Is the supplier financially stable and likely to continue in existence? Is the supplier replaceable?
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.
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 high, it may be worth considering system replacement.
Maintenance costs	What are the costs of hardware maintenance and support software licenses (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?

Key points





- 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.
- The business-value assessment should take different stakeholder viewpoints into account.