Software Evolution & Maintenance

HND Thilini

(Slides courtesy: Prof. Ian Sommerville)

Course Outline

Learning Outcomes

- Understand Software evolution process
- Describe the types of software maintenance
- Describe the software maintenance process
- Describe activities of configuration management

Outline

- Evolution processes
  - Change processes for software systems
- Program evolution dynamics
  - Understanding software evolution
- Software maintenance
  - Making changes to operational software systems
- Software Configuration management
  - Developing and applying of standards and procedures for managing an evolving software product

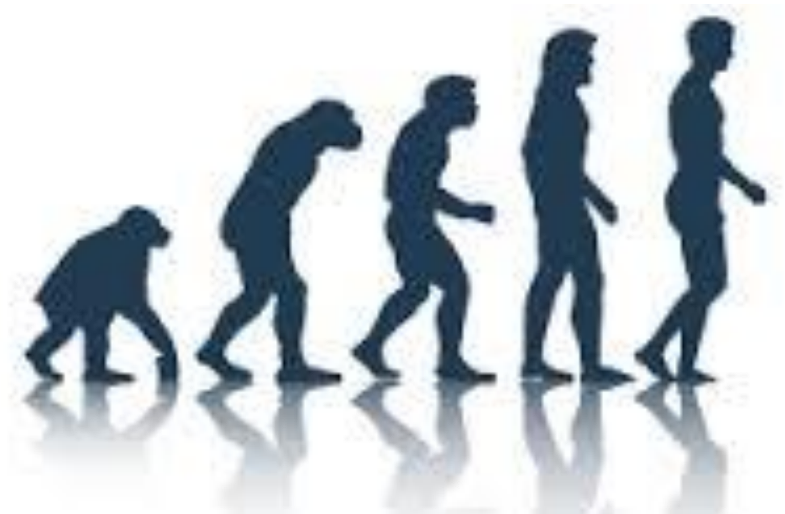
Software change

- 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.

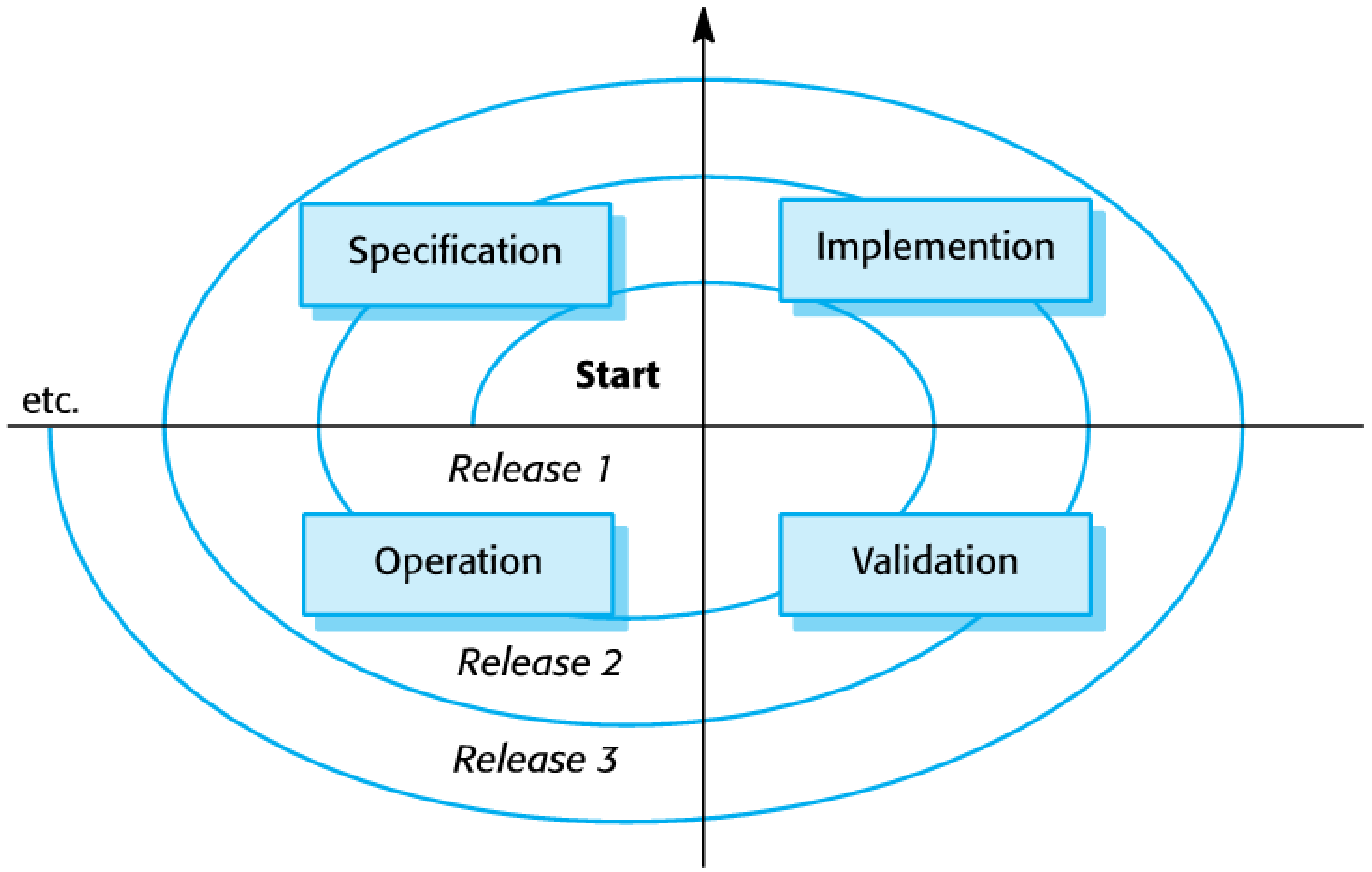
- A key problem for all organizations is implementing and managing change to their existing software systems.

Importance of evolution

- 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 changing and evolving existing software rather than developing new software.



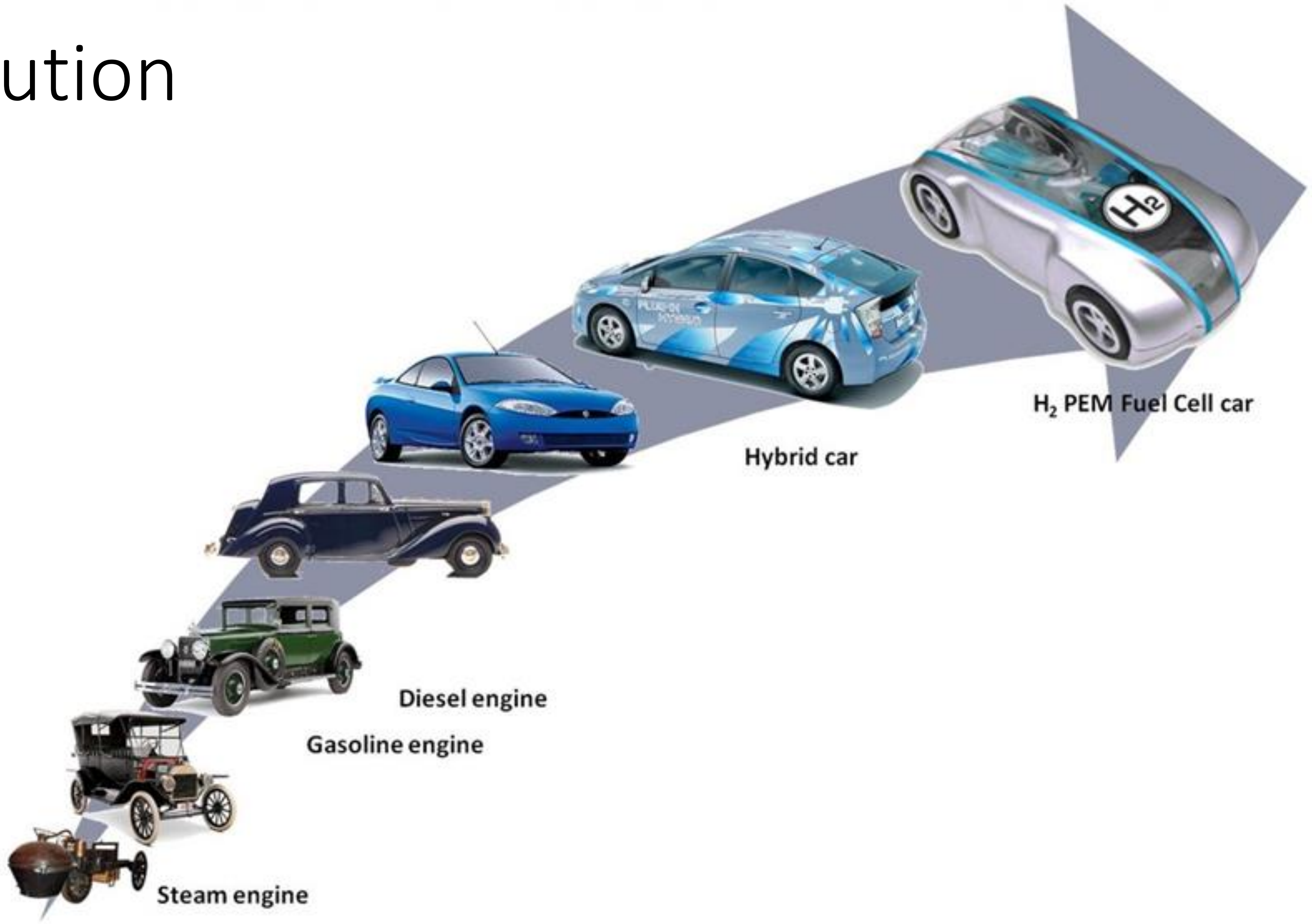
A spiral model of development and evolution



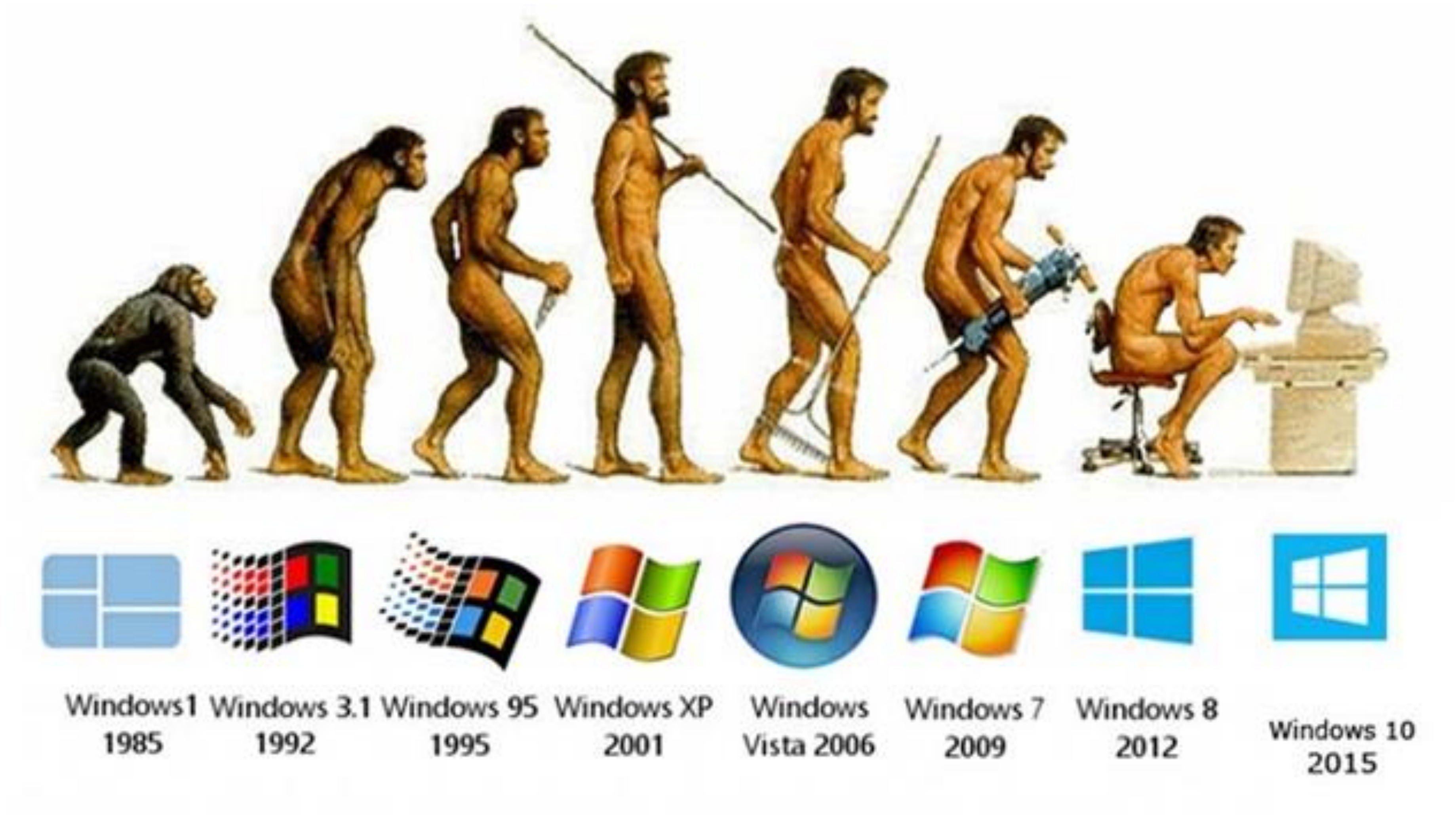
Evolution and Servicing

Servicing

Evolution


Steam engine
Gasoline engine
Diesel engine
Hybrid car
H_2 PEM Fuel Cell car

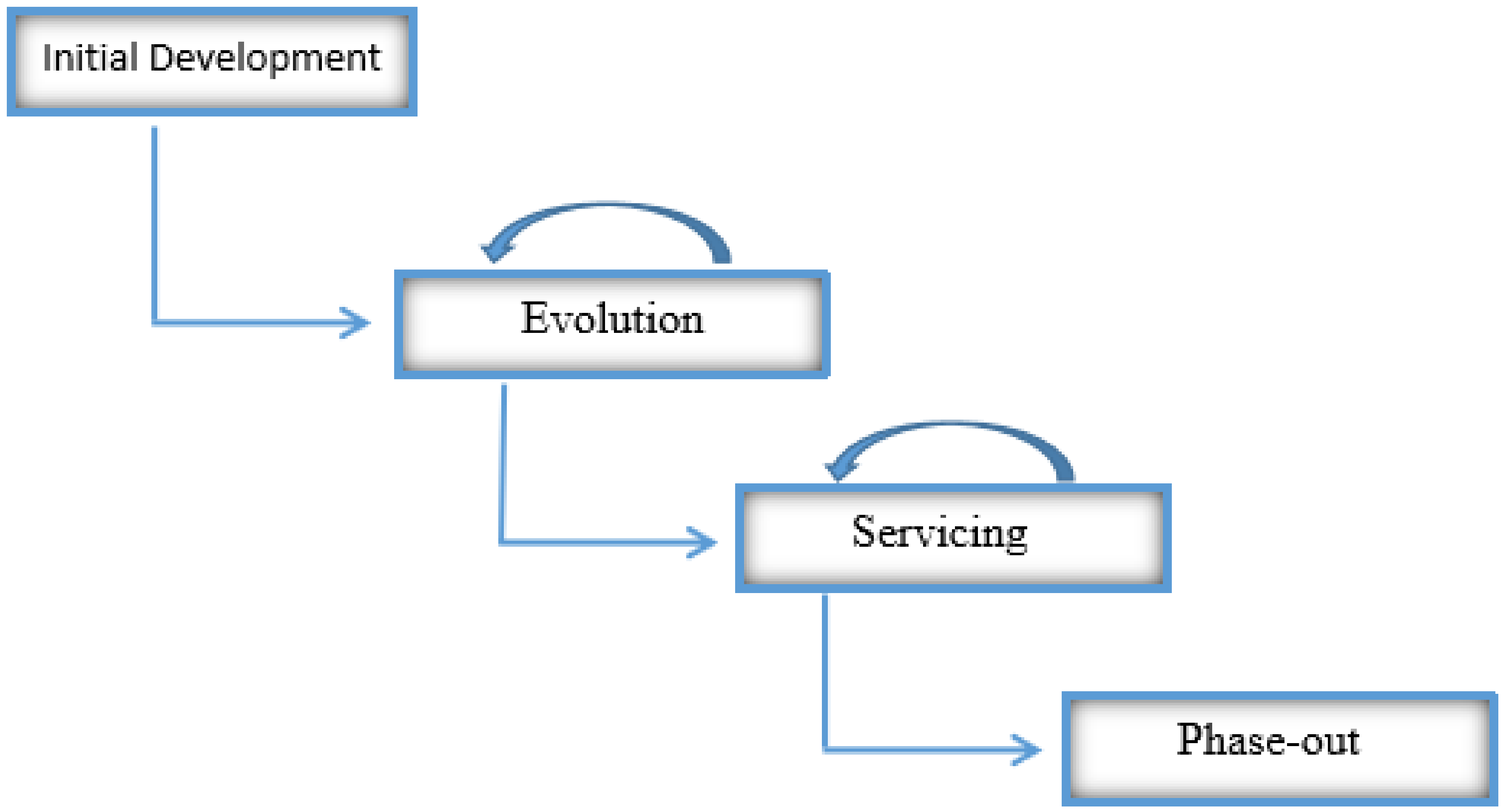
Evolution


Windows1 1985
Windows 3.1 1992
Windows 95 1995
Windows XP 2001
Windows Vista 2006
Windows 7 2009
Windows 8 2012
Windows 10 2015

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 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.

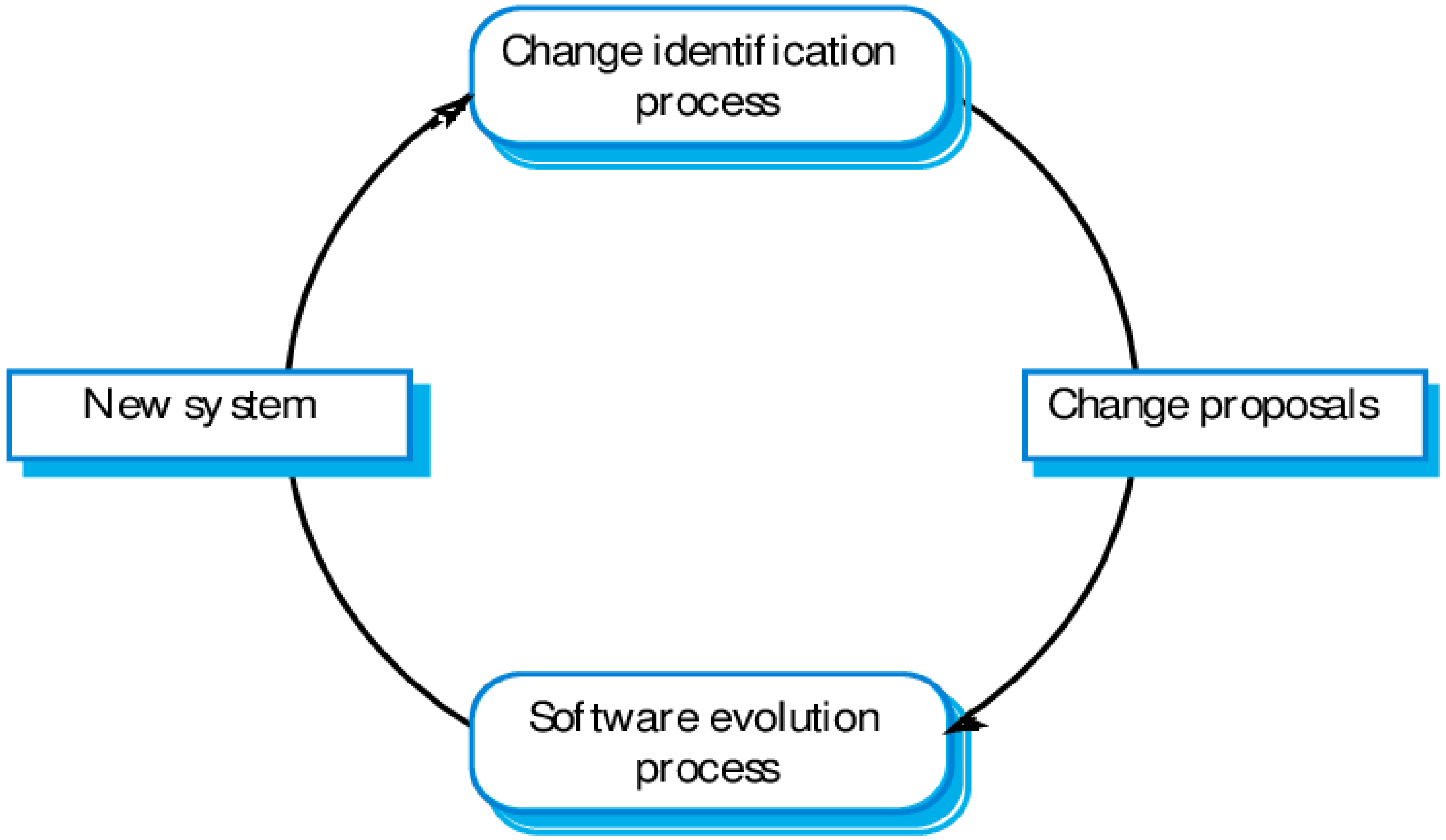
Evolution and servicing



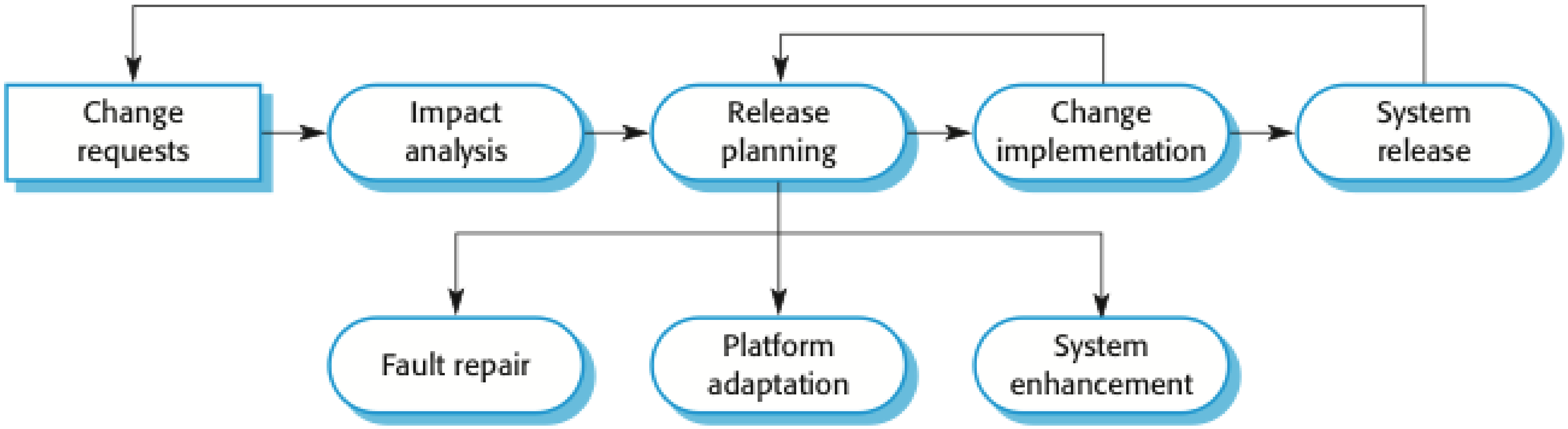
Evolution processes

- 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



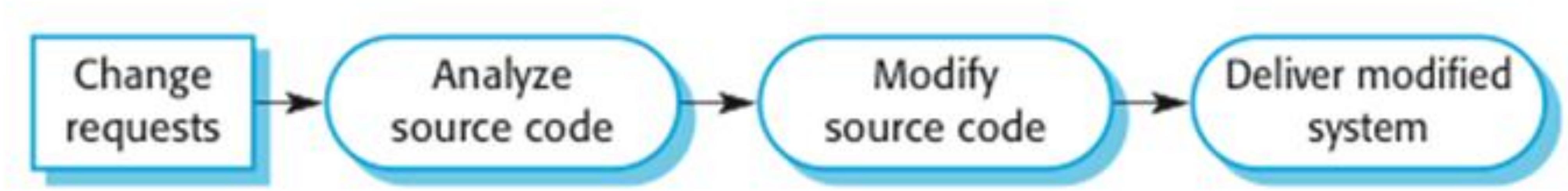
Change implementation

- 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.

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;
  - If there are business changes that require a very rapid response (e.g. the release of a competing product).

The emergency repair process



Outline

- Evolution processes
  - Change processes for software systems
- Program evolution dynamics
  - Understanding software evolution
- Software maintenance
  - Making changes to operational software systems
- Software Configuration management
  - Developing and applying of standards and procedures for managing an evolving software product

Program evolution dynamics

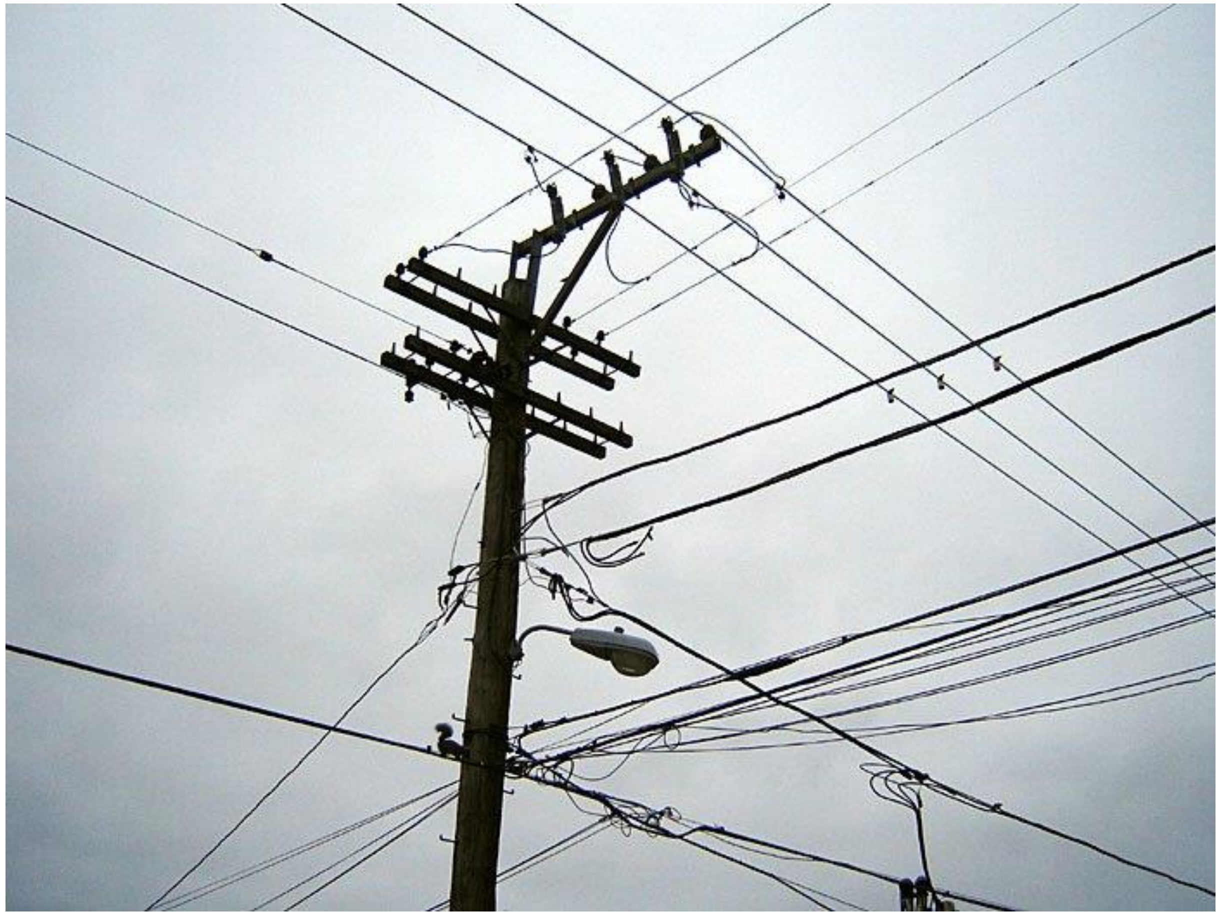
- *Program evolution dynamics* is the study of the processes of system change.
- After several major empirical studies, *Lehman* and *Belady* proposed that there were a number of ‘laws’ which applied to all systems as they evolved.
- There are sensible observations rather than laws. They are applicable to large systems developed by large organisations.
  - It is not clear if these are applicable to other types of software system.

Lehman’s laws

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.
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.
Organizational stability	Over a program’s lifetime, its rate of development is approximately constant and independent of the resources devoted to system development.

Lehman’s laws

Law	Description
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 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.



Hercul
U.P. CYCL

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
  - Shrink-wrapped software products;
  - Systems that incorporate a significant number of COTS components;
  - Small organisations;
  - Medium sized systems.

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.
- Lehman’s laws, such as the notion that change is continuous, describe a number of insights derived from long-term studies of system evolution.

Outline

- Evolution processes
  - Change processes for software systems
- Program evolution dynamics
  - Understanding software evolution
- Software maintenance
  - Making changes to operational software systems
- Software Configuration management
  - Developing and applying of standards and procedures for managing an evolving software product

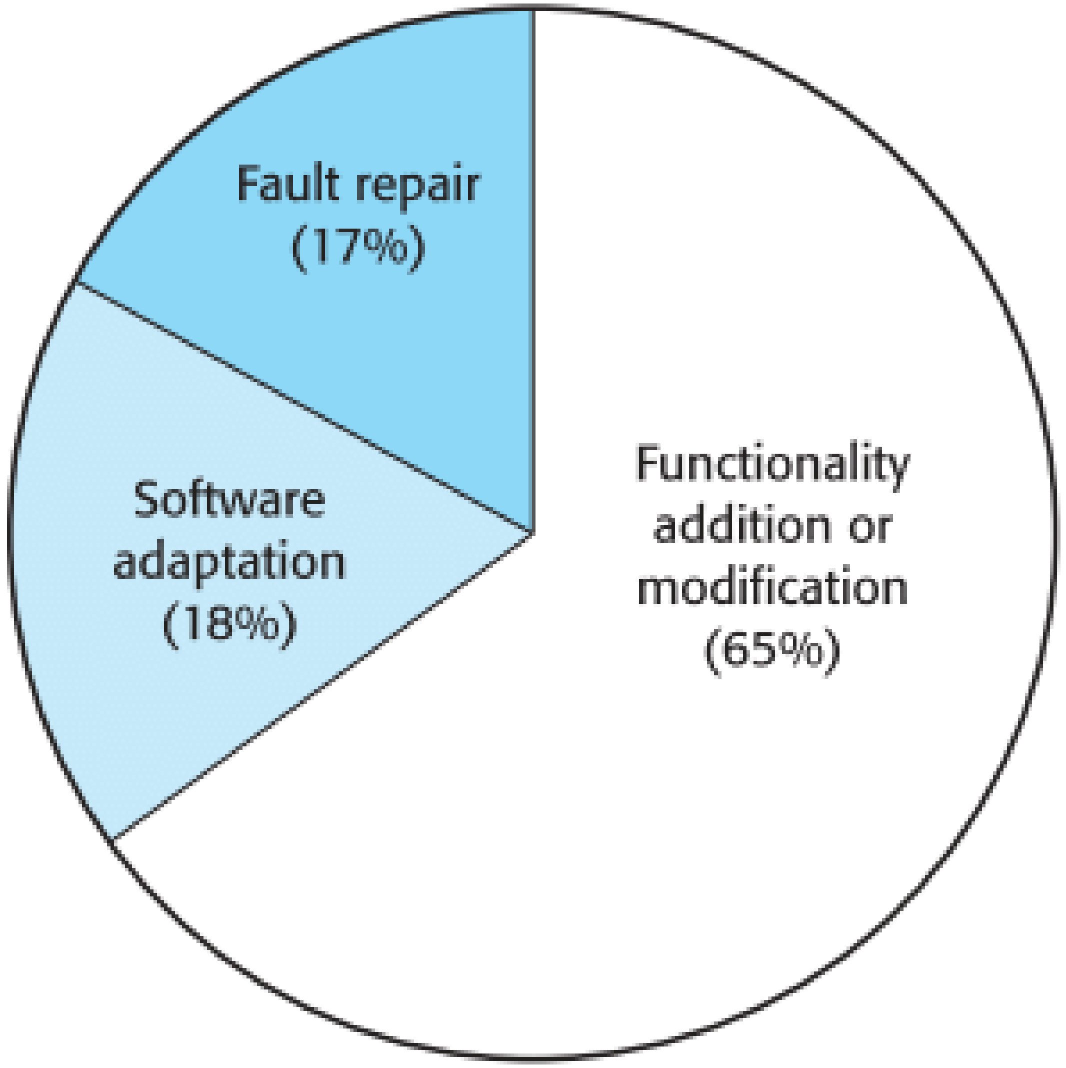
Software maintenance

- 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** (*Fault Repairs*)
  - Maintenance to repair software faults
  - Changing a system to correct deficiencies in the way meets its requirements.
- **Adaptive 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.
- **Perfective maintenance (***Functionality Addition***)**
  - Maintenance to add to or modify the system's functionality
  - Modifying the system to satisfy new requirements.

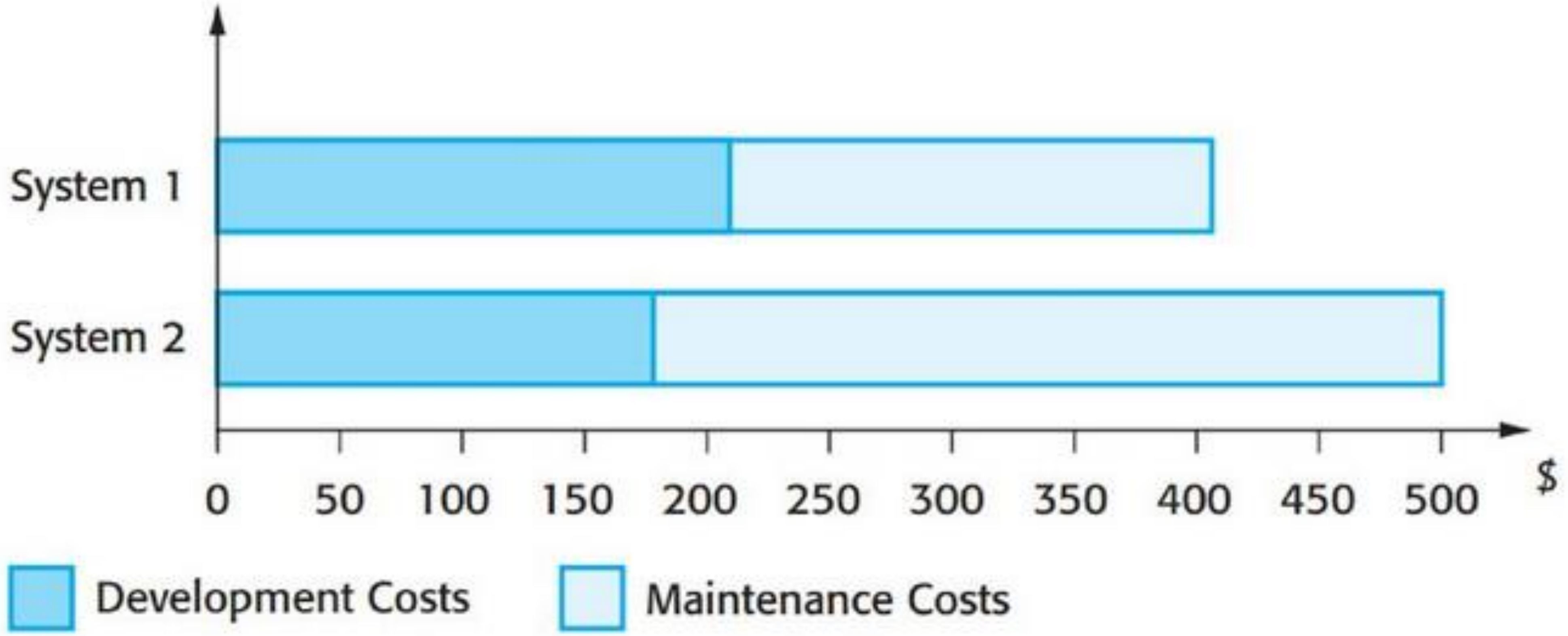
Maintenance effort distribution



Maintenance costs

- Usually greater than development costs (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



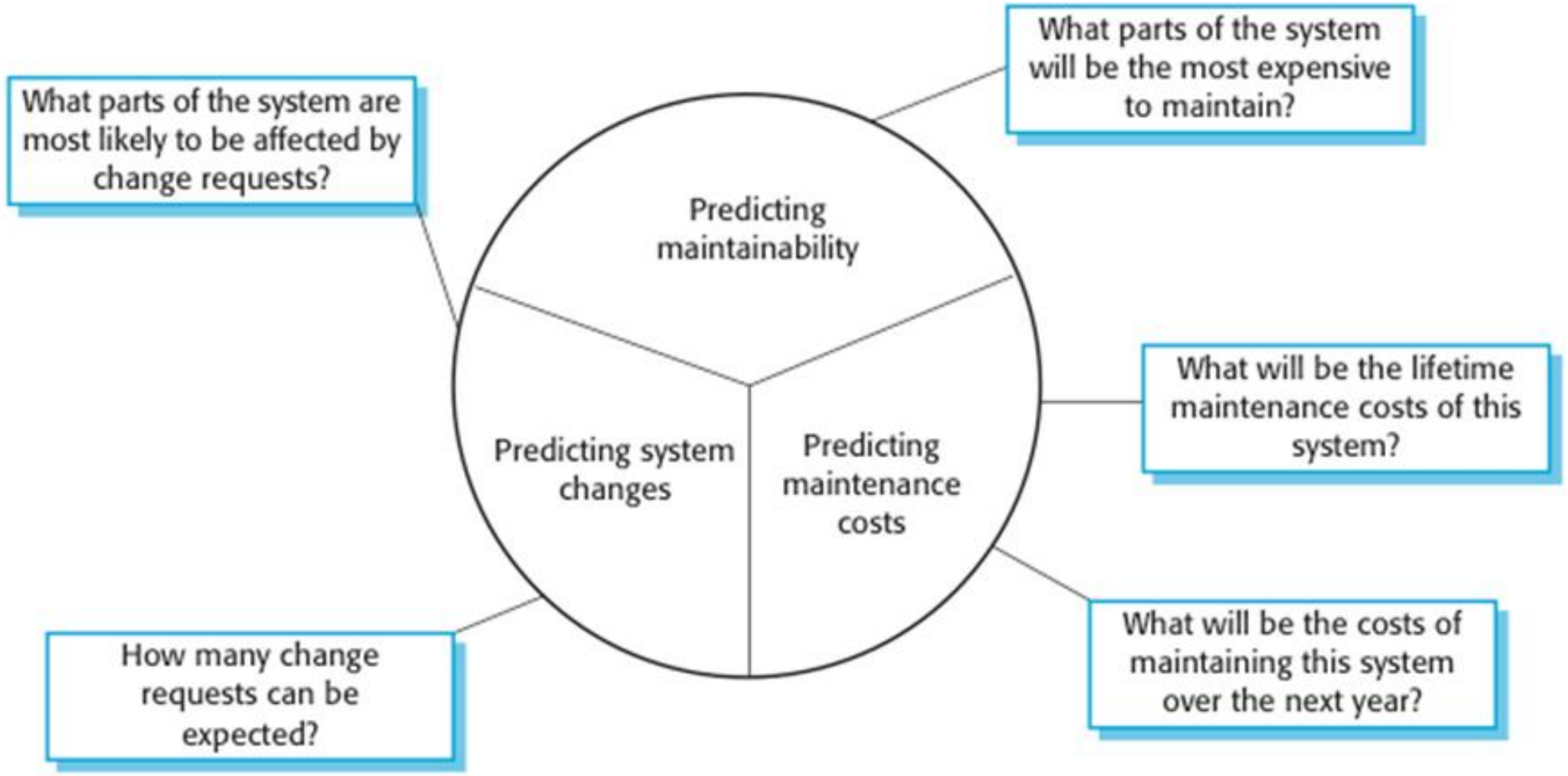
Maintenance cost factors

- 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.

Maintenance prediction


What parts of the system are most likely to be affected by change requests?
What parts of the system will be the most expensive to maintain?
Predicting maintainability
What will be the lifetime maintenance costs of this system?
Predicting system changes
Predicting maintenance costs
How many change requests can be expected?
What will be the costs of maintaining this system over the next year?

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 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 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.

System re-engineering

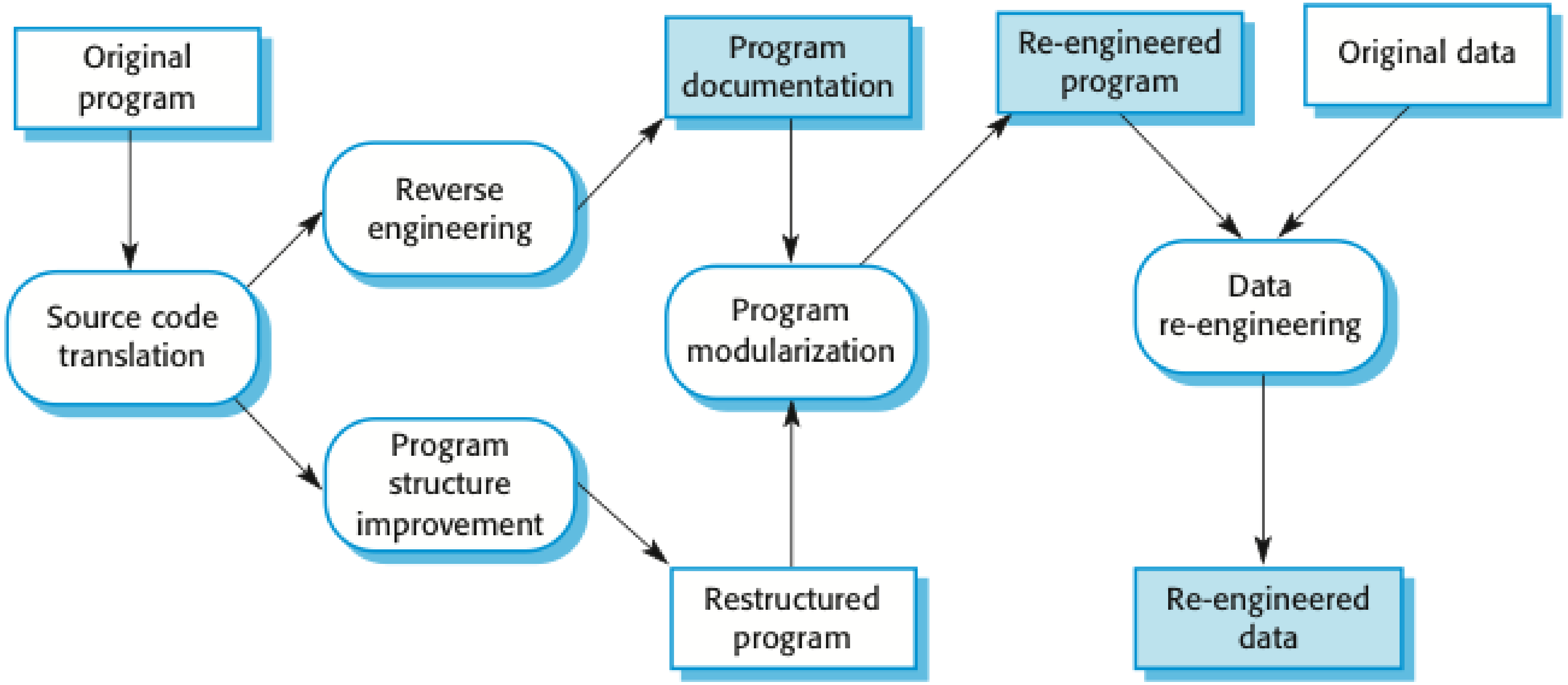
- What?
  - Re-structuring or re-writing part or all of an existing system without changing its functionality
- When?
  - When some but not all sub-systems of a larger system require frequent maintenance
  - When hardware or software support becomes obsolete
- How?
  - The system may be re-structured and re-documented to make it easier to maintain
- Why?
  - Reduced risk
    - New software development carries high risk.
  - Reduced cost
    - The cost of re-engineering is often significantly less than the costs of developing new software

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.

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 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.
- 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

- 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.

Outline

- Evolution processes
  - Change processes for software systems
- Program evolution dynamics
  - Understanding software evolution
- Software maintenance
  - Making changes to operational software systems
- Software Configuration management
  - Developing and applying of standards and procedures for managing an evolving software product

Configuration management

- Configuration management is the development and application of standards and procedures for managing an evolving software product.
- You need to manage evolving systems because, as they evolve, many different versions of the software are created.
- These versions incorporate corrections of faults and adaptations for different hardware and operating systems.

Configuration management

- There may be several versions under development and in use at the same time.
- You need to keep track of these changes that have been implemented.
- All products of the software process may have to be managed:
    - Specifications
    - Designs
    - Programs
    - Test data
    - User manuals

Configuration Management Plan

- Describes the standards and procedures which should be used for configuration management
- The definitions of what entities are to be managed and a formal scheme for identifying these entities.
- A statement of who takes responsibility for configuration management procedures and for submitting entities to the configuration management team.
- The configuration management policies that are used for change control and version management.

Configuration management Plan

- A description of the records of the configuration management process which should be maintained.
- A description of the tools to be used for configuration management and the process to be applied when using these tools.
- A definition of the configuration database which is used to record configuration information.

The Configuration Database

- All CM information should be maintained in a configuration database
- Usually caters for queries like :
    1. Who has a particular system version ?
    2. What platform is required for a particular version ?
    3. What versions are affected by a change to component X ?
    4. How many reported faults in version X ?
- The CM database should preferably be linked to the software being managed

CASE tools for configuration management

- Configuration management processes are standardized and involve applying pre defined procedures
- Large amounts of data must be managed
- CASE tool support for configuration management is therefore essential
- Mature CASE tools to support configuration management are available ranging from stand alone tools to integrated configuration management workbenches

Versions/variants/releases

- **Version**
  - An instance of a system which is functionally distinct in some way from other system instances
- **Variant**
  - An instance of a system which is functionally identical but non-functionally distinct from other instances of a system
- **Release**
  - An instance of a system which is distributed to users outside of the development team

Key points

- There are 3 types of software maintenance, namely Corrective, Adaptive and Perfective maintenance.
- 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.
- Configuration management is essential to maintain the versions of the system.