Software Re-Engineering

Lecture: 02

Dr. Imran Ali
School of Computing- Software Engineering
FAST-National University of Computer & Emerging Sciences, Karachi

Sequence [Todays Agenda]

Content of Lecture

- Software Evolution Process
- Software Maintenance Process

General Terminologies

- Evolution processes
  - Change processes for software systems
- Software maintenance
  - Making changes to operational software systems

Software Evolution

- Software Evolution is a term that refers to the process of developing software initially, and then timely updating it for various reasons, i.e., to add new features or to remove obsolete functionalities.
- The software evolution process includes fundamental activities of change analysis, release planning, system implementation, and releasing a system to customers

Laws of Software Evolution

- The fundamental work in the field of software evolution was done by Lehman and his collaborators.
- Based on empirical studies Lehman and his collaborators formulated some observations and they introduced them as laws of evolution.
- These laws evolved from three in 1974 to eight by 1997.
- Those laws are the results of studies of the evolution of large-scale proprietary or closed source software (CSS) systems.

Laws of Software Evolution

- The eight laws are briefly explained as follows:
- Law of Continuing Change
- Law of Increasing Complexity
- Law of Large Program Evolution
- Law of Organizational Stability
- Law of Conservation of Familiarity
- Law of Continuing Growth
- Law of Declining Quality
- Law of Feedback System

Laws of Software Evolution

❑ Law of Continuing Change:

This law states that any software system that represents some real-world reality undergoes continuous change or become progressively less useful in that environment. (This law highlights the inevitability of change in software systems. As the environment in which software operates evolves due to new user requirements, technological advancements, or regulatory changes software must adapt to remain relevant and useful.

❑ Law of Increasing Complexity:

As an evolving program changes, its structure becomes more complex unless effective efforts are made to avoid this phenomenon. (With each modification and addition, software tend to become more complex. This increasing complexity can lead to higher maintenance costs and greater potential for defects. To manage this complexity, developers must invest in continuous refactoring, code reviews, and the use of modular architectures.)

Laws of Software Evolution

❑ Law of Large Program Evolution

This law implies a level of predictability in software evolution. By analyzing past releases, project managers can estimate future resource requirements and timelines. Agile methodologies and iterative development processes embody this principle by promoting regular, incremental updates and continuous feedback loops.

❑ Law of Conservation of Organization Stability:

Over the lifetime of a program, the rate of development of that program is approximately constant and independent of the resource devoted to system development. (This law suggests that increasing resources (adding more developers) does not accelerate development. This principle highlights the importance of efficient team collaboration and well-defined processes over mere manpower. Effective communication, agile frameworks, and lean development practices are essential to maintaining productivity.)

Laws of Software Evolution

- Law of Conservation of Familiarity:

This law states that during the active lifetime of the program, changes made in the successive release are almost constant. (Software systems must balance the need for innovation with the need to remain familiar to their users. Drastic changes can lead to user frustration. Therefore, incremental improvements and consistent user interfaces are crucial for maintaining user satisfaction and ensuring a smooth transition between versions.)

- Law of Continuing Growth

The functional content of a program must continually increase to maintain user satisfaction over its lifetime. (User expectations evolve, often requiring additional features and enhancements to meet new demands. To remain competitive and satisfy users, software must continually grow in functionality. This law drives the need for ongoing research and development, user feedback mechanisms, and robust product roadmaps.)

Laws of Software Evolution

- Law of Declining Quality:

The quality of a program will appear to be declining unless it is rigorously maintained and adapted to operational environment changes. (Perceived software quality tends to degrade over time unless proactive measures are taken. Regular maintenance, updates, and performance tuning are essential to counteract this decline. Automated testing, continuous integration, and deployment pipelines are modern strategies to ensure sustained software quality.)

- Law of Feedback System:

Evolving a large program is like running a multi-loop, multi-person, multi-level feedback system. (Effective software evolution relies on feedback from various sources, including users, developers, and automated monitoring tools. This law emphasizes the complexity of managing software projects and the necessity of feedback loops. Agile practices, user feedback sessions, and monitoring systems help gather and integrate feedback to guide development.)

Software Change Management

- Change Management in software development refers to the transition from an existing state of the software product to another improved state of the product.
- It controls, supports, and manages changes to artifacts, such as code changes, process changes, or documentation changes.
- Where CCP (Change Control Process) mainly identifies, documents, and authorizes changes to a software application.

Software Change Management

- Change Management in software development refers to the transition from an existing state of the software product to another improved state of the product.
- It controls, supports, and manages changes to artifacts, such as code changes, process changes, or documentation changes.
- Where CCP (Change Control Process) mainly identifies, documents, and authorizes changes to a software application.

Software Change Management

- Each software development process follows Software Development Life Cycle (SDLC) where each phase is accordingly followed to finally deliver a good quality software product.
- Change Management does not come under any phases of SDLC still it has great importance in the entire software development process.
- Various types of change management tools are used for various purposes like to adopt, control, represent, and effect the change required.
- For example, Change management tools for Flow Charting, Project Planning, Data collection.

Process of Software Change Management

- When any software application/product goes for any changes, it undergoes a series of sequential processes as follows:
- Creating a request for change
- Reviewing and assessing a request for change
- Planning the change
- Testing the change
- Creating a change proposal
- Implementing changes
- Reviewing change performance
- Closing the process

Importance of Software Change Management

- To improve performance
- To increase engagement
- For enhancing innovation
- For including new technologies
- For implementing new requirements
- To reduce cost (Quality control, Long term maintenance etc.)

Source of Software Change Management

- There may be multiple reasons involved during the development process for which certain changes are required to be implemented in the product. These sources are as follows :
- Business reorganization
- New Market conditions
- New equipment
- Fixing any bugs/errors
- New customer needs
- Performance or reliability improvement
- Budgetary or scheduling constraints

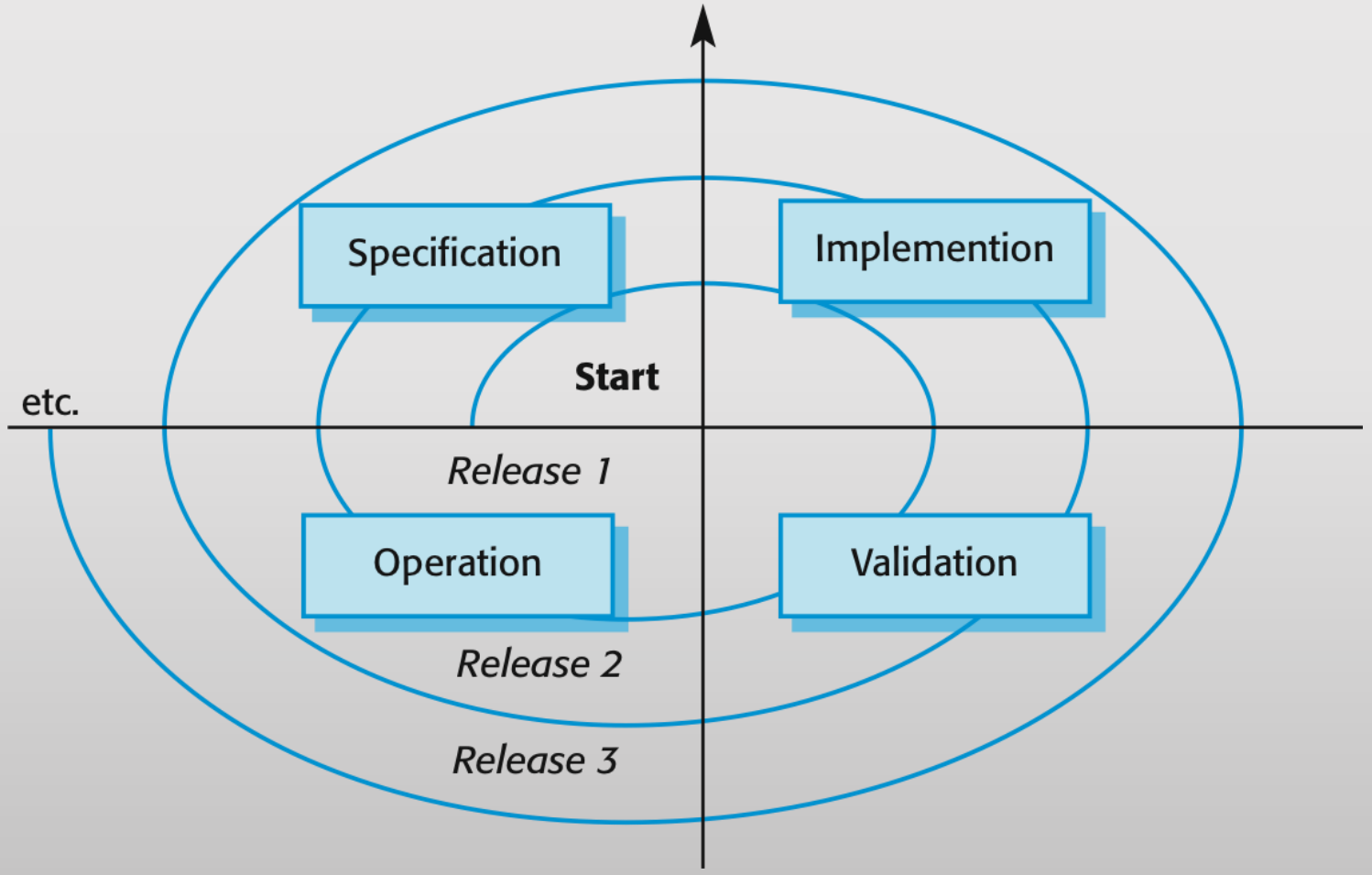
Importance of Evolution

- 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

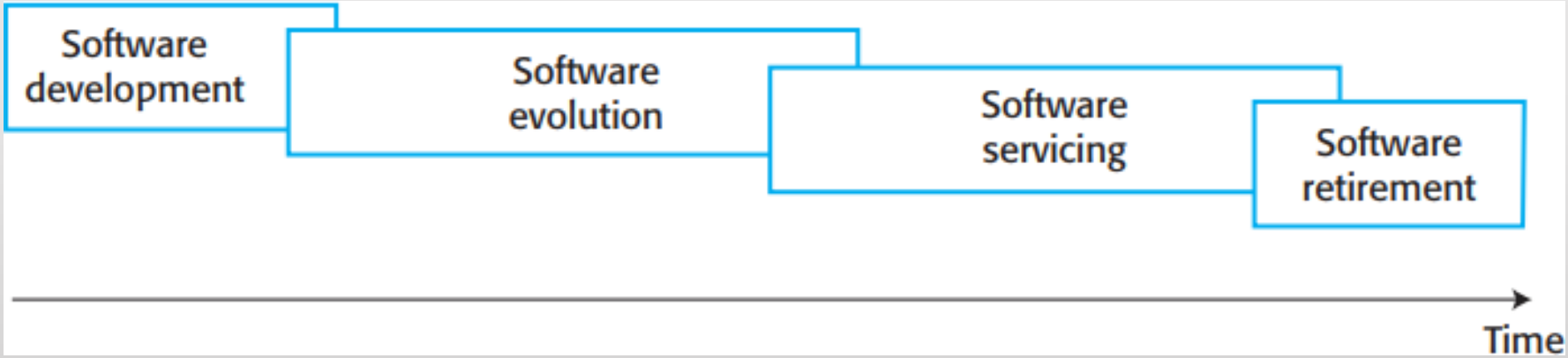
Spiral Model of Development and Evolution

- The Spiral Model is a Software Development Life Cycle (SDLC) model that provides a systematic and iterative approach to software development.
- The spiral model uses the approach of the Prototyping Model by building a prototype at the start of each phase as a risk-handling technique.
- Also, the spiral model can be considered as supporting the Evolutionary model
- The iterations along the spiral can be considered as evolutionary levels through which the complete system is built.

Spiral Model of Development and Evolution



Evolution and Servicing



Evolution and servicing

- Evolution

  The stage in a software system's lifecycle 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

Software Retirement

- It involves removing or replacing software that is no longer needed, supported, or compatible with the current system.
- Retiring software can improve security, performance, and usability, as well as reduce costs and risks
- Best Practices for Software Retirements are:
- Identify which software components need to be retired
- Communicate retirement plan to the relevant parties (users, clients, stakeholders, developers, testers, and administrators)
- Performing necessary tasks and tests to remove or replace software (version control, backup, documentation, testing, and monitoring)
- Measuring and analyzing the effects and outcomes of retirement, such as performance, security, usability, or satisfaction

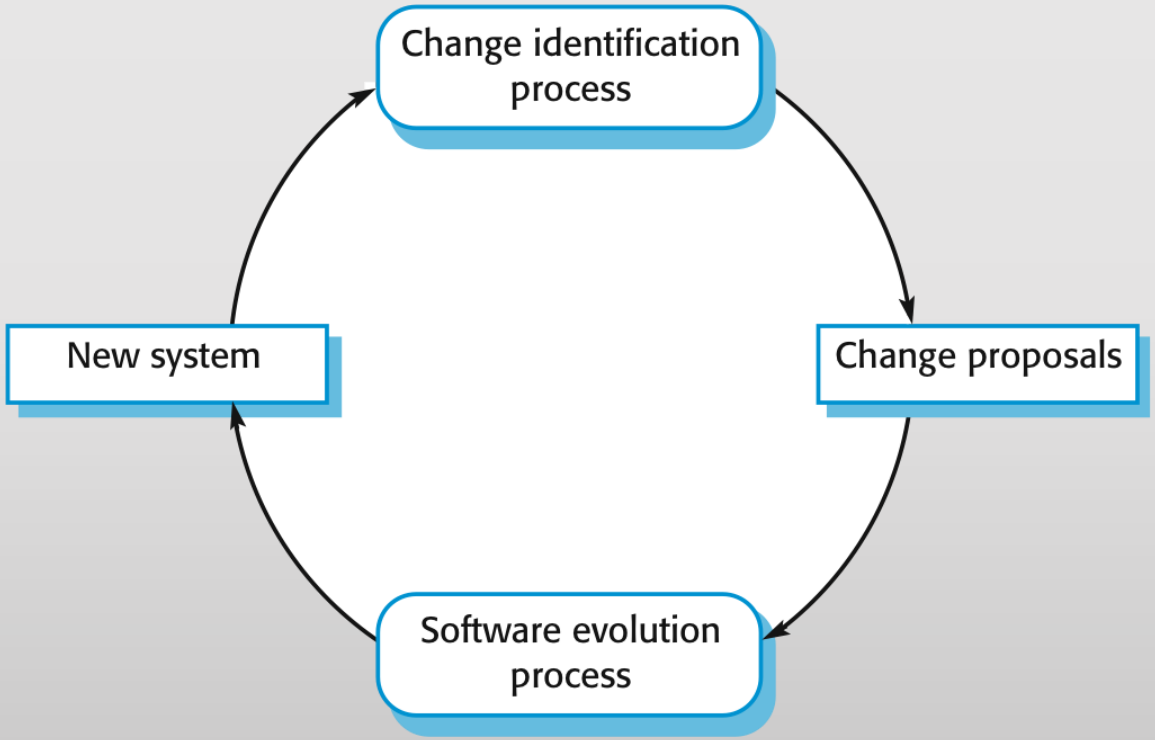
Evolution Processes

- 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:

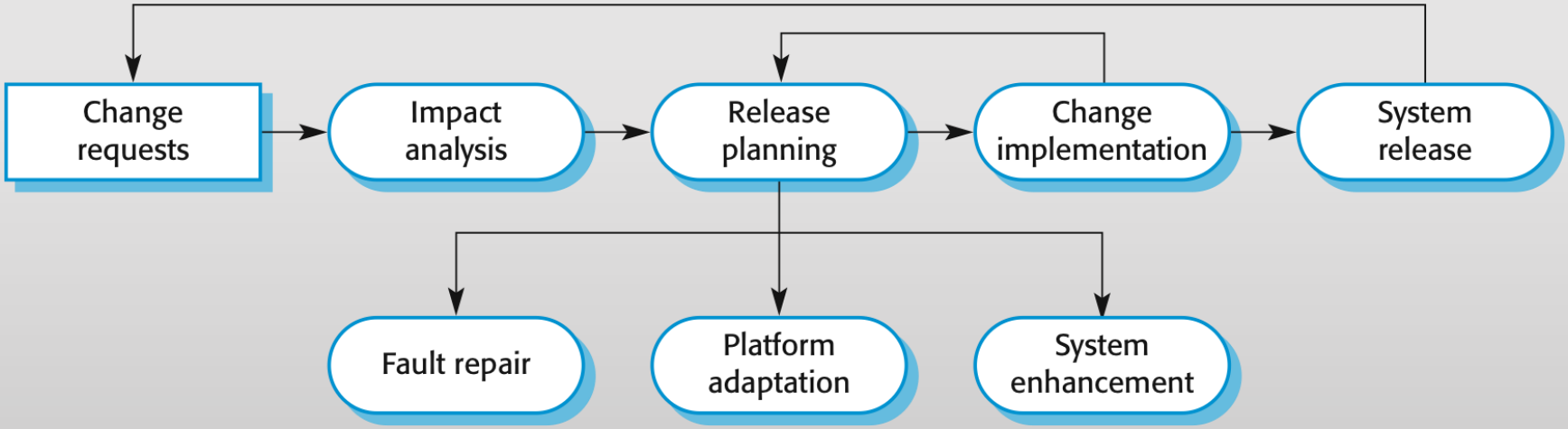
Should be linked with components that are affected by the change, thus allowing the cost and impact of the change to be estimated

- Change identification and evolution continues throughout the system lifetime

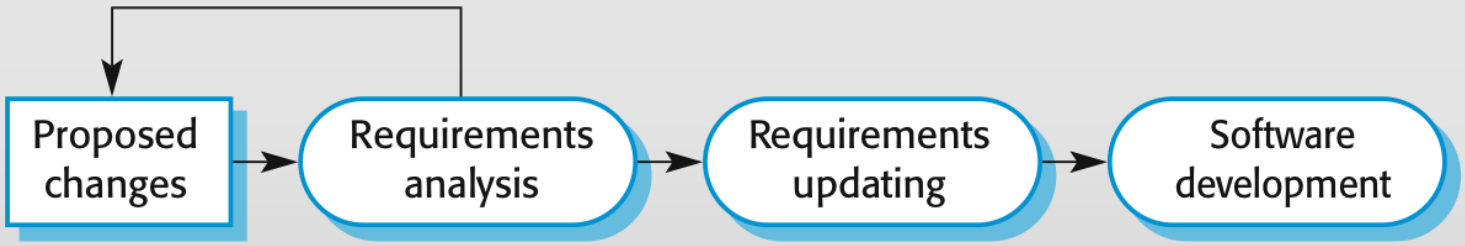
Change Identification and Evolution Processes



The Software Evolution Process



Change Implementation



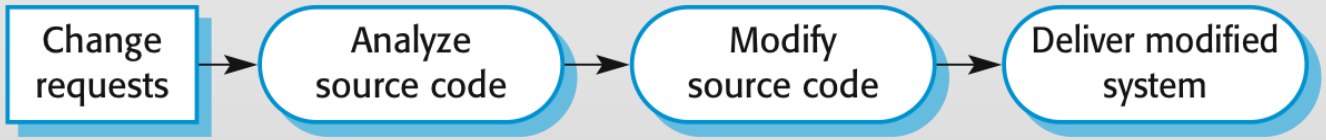
Change Implementation

- 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



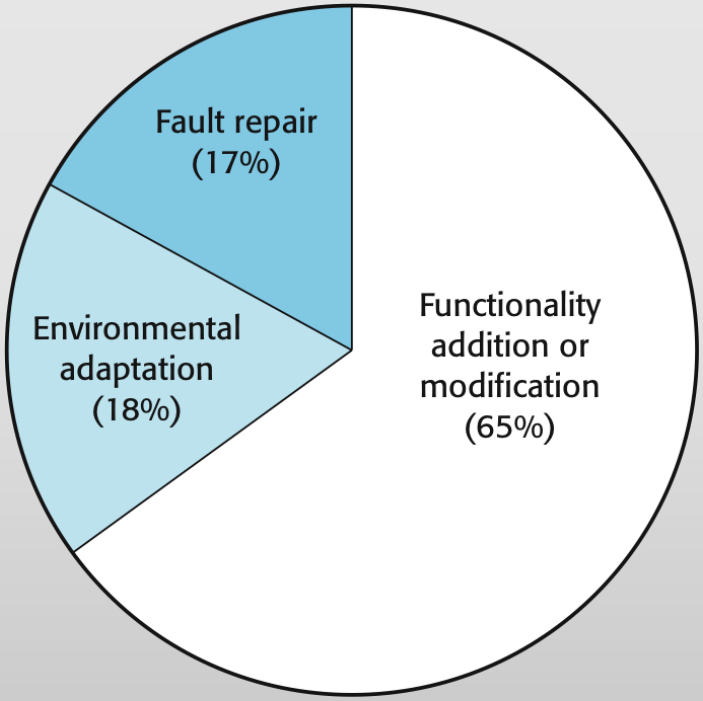
Software Maintenance

- 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. (A type of software that is developed for general use and can be used by multiple customers or industries. Typically designed to perform common functions that are useful to many users, such as word processing, spreadsheet management, or web browsing. Generic software is often readily available for purchase or download.)
- 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

- 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 Distribution

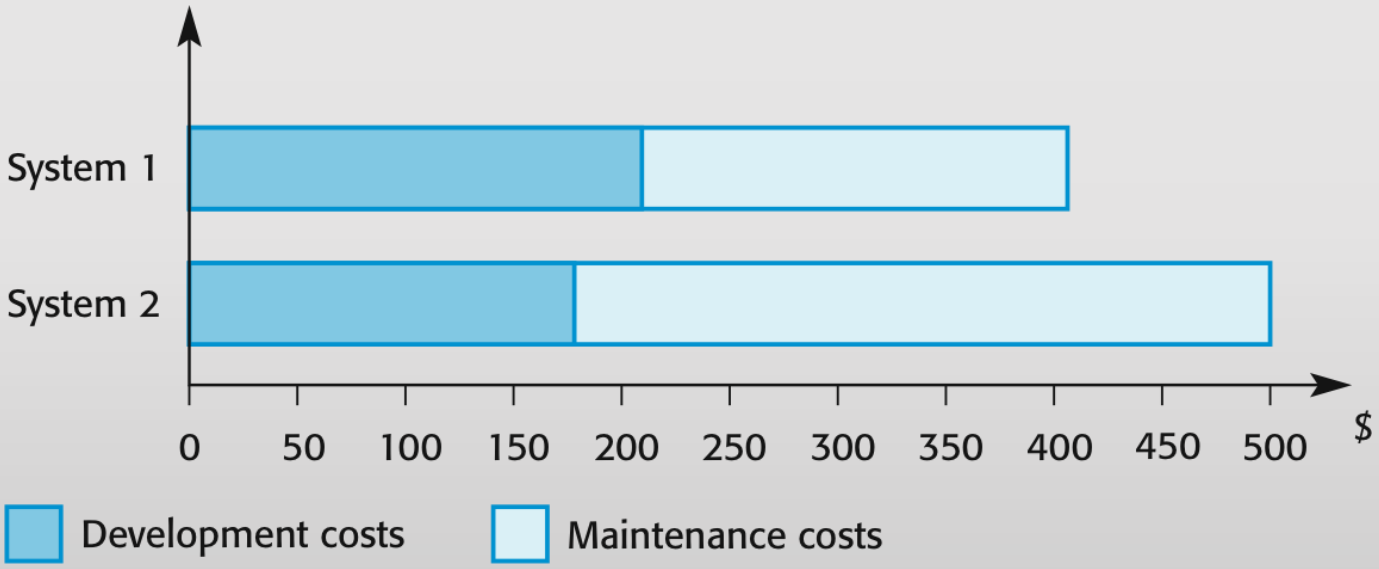


Maintenance Costs

- Usually greater than development costs
(depends upon the application)
- Affected by both technical and non-technical factors
- Increases as software is maintained. Maintenance makes the software structure complex, so further maintenance becomes more difficult.
- Ageing software can have high support costs (e.g., old languages, compilers etc.).

(Ageing software: tendency for software to fail or cause a system failure after running continuously for a certain time)

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

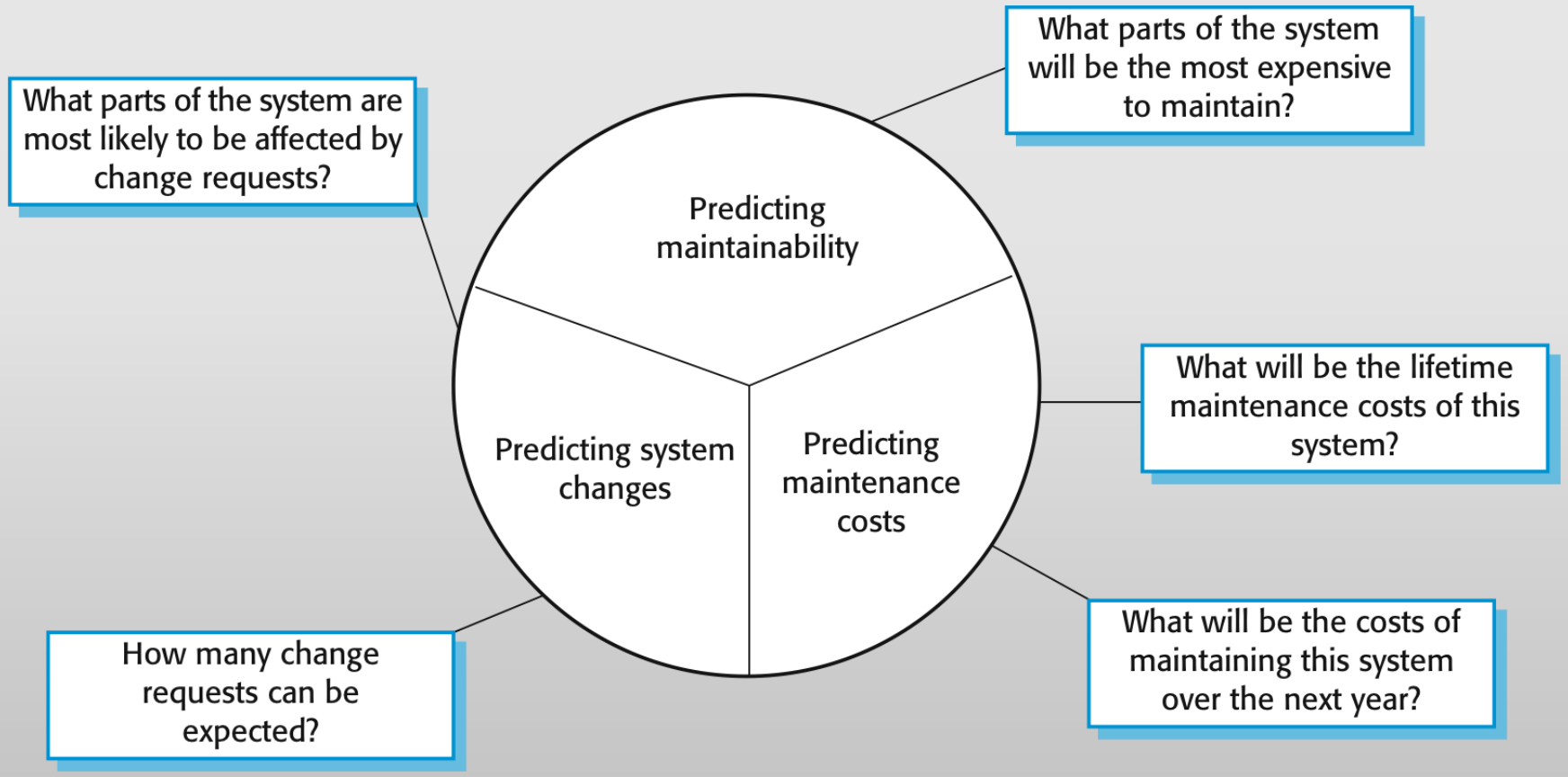
- Program age and structure

  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



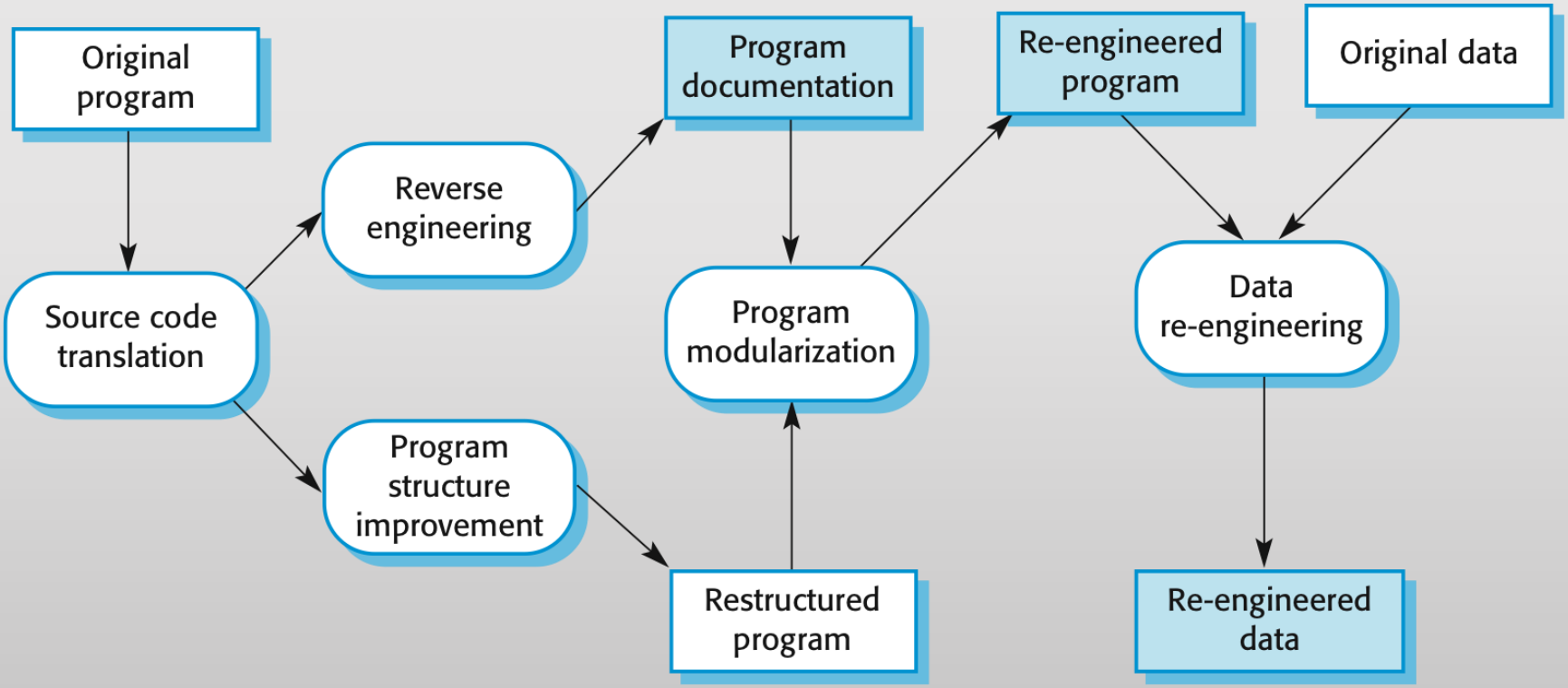
System Re-Engineering

- 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.

The Re-Engineering 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

Re-Engineering 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

Preventive 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 ‘preventive 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 Re-Engineering

- 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.

“Bad smells” in Program Code

- Duplicate code

  The same or very similar code may be included at different places in a program. This can be removed and implemented as a single method or function that is called as required.

- Long methods

  If a method is too long, it should be redesigned as a number of shorter methods

- Switch (case) statements

  These often involve duplication, where switch depends on the type of value. The switch statements may be scattered around a program. In object-oriented languages, you can often use polymorphism to achieve the same thing.

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

Key Points

- 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.
- 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


Thank You!