Software Engineering COMP1035

Lecture 14

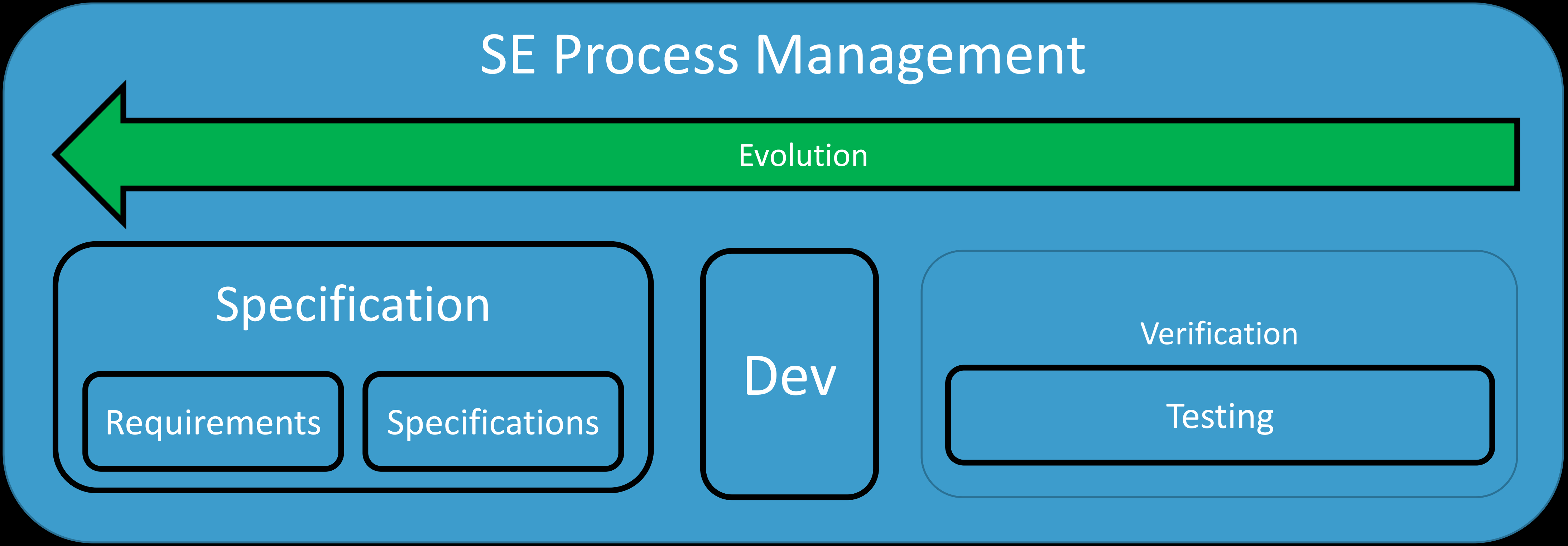
Evolution & Maintenance

Outline

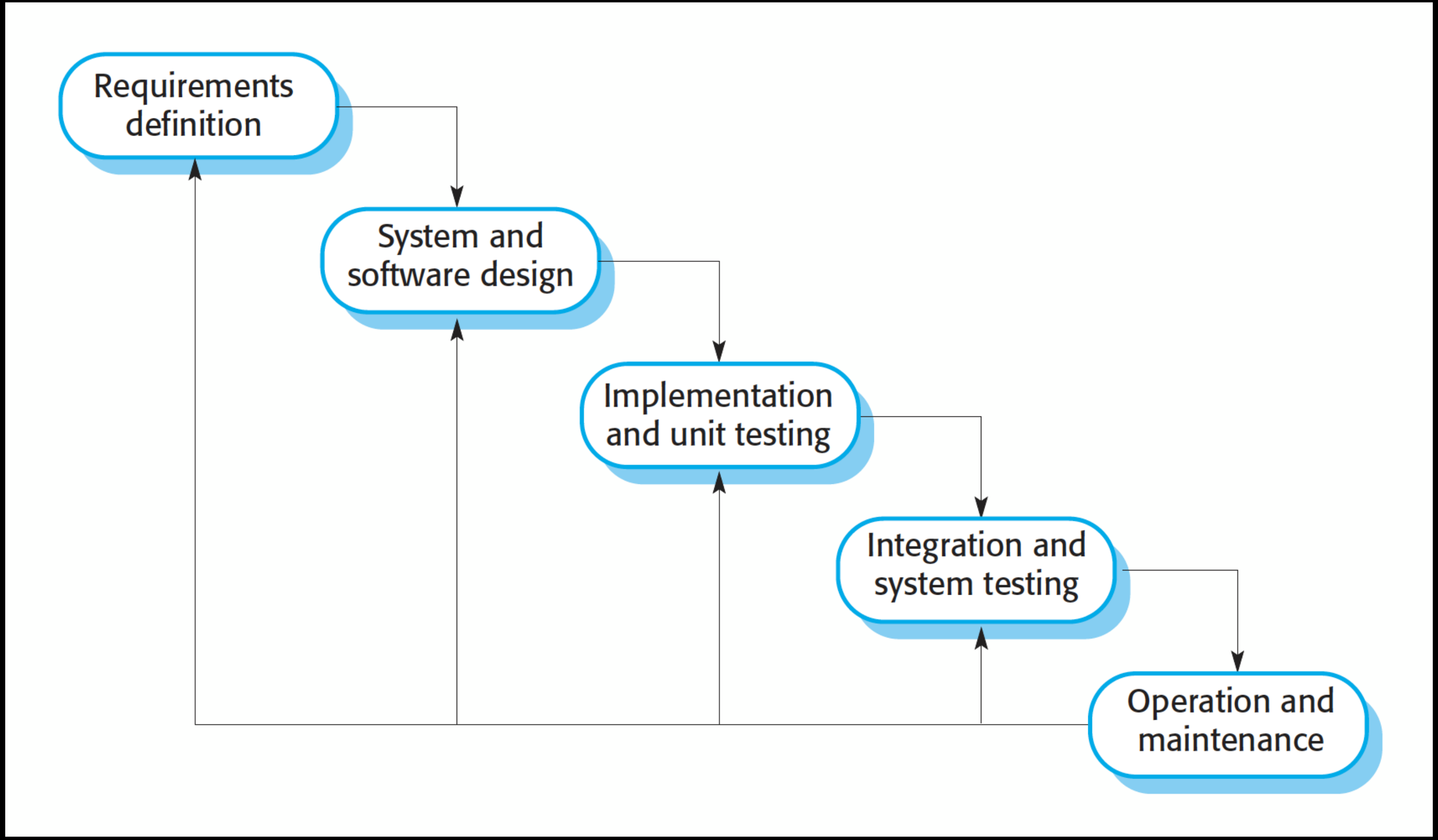
- Software Evolution Process
- Software Maintenance
- Legacy System Management

Where We Are In the Process

Keeping Track of SE Module



Waterfall model


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Maintenance

- Maintenance involves:

» **Correcting** errors which were not discovered in earlier stage of the life cycle

» **Improving** the implementation of system units

» **Enhancing** the system's services as new requirements are discovered

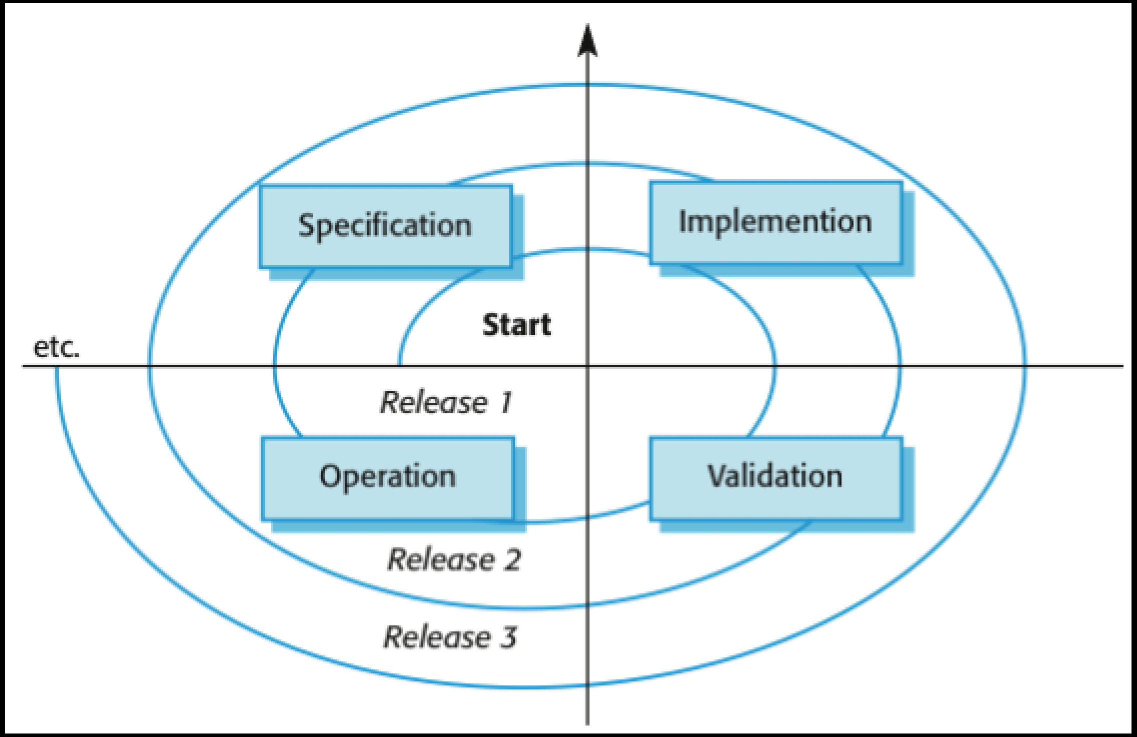
- ***"Normally, this is the longest lifecycle phase"***

Software Eng, 9th - p31

Change is Inevitable

Change is Inevitable

- **New requirements** emerge when the software is used.
- The business **environment changes**.
- **Errors** must be repaired.
- **New equipment** is added to the system.
- The **performance** or **reliability** of the system may have to **be improved**.



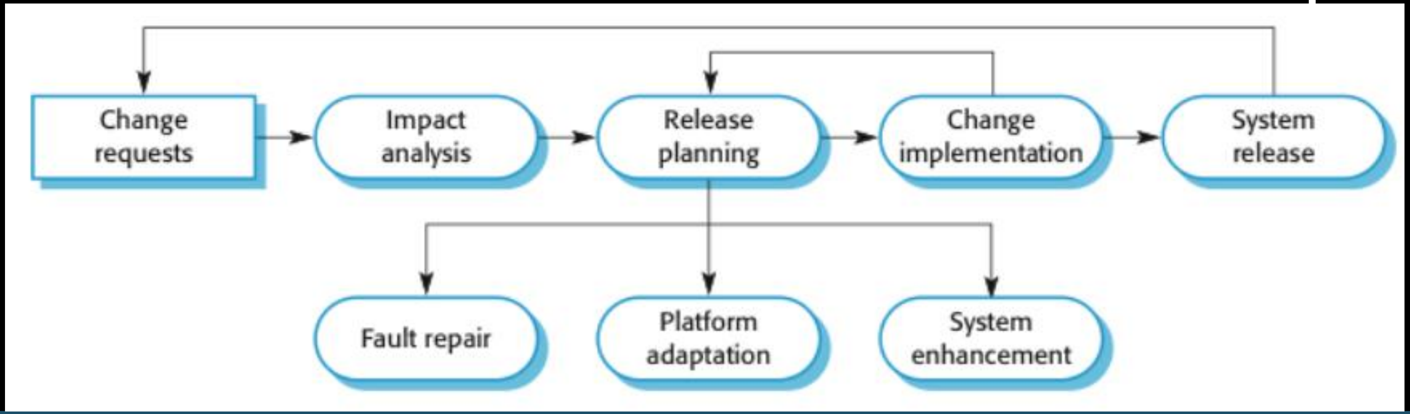
Change is Inevitable

- The **majority** (**85%-90%**) of the software budget in large companies is devoted to **changing** and **evolving** existing software rather than developing new software.
    - Building a whole new piece of software is a risk
        - Errors might be made in specifications for new software
        - New software might take longer to arrive
    - A whole new project might be more expensive
        - Move data, use unfamiliar software, train people

Evolution Process

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: driver for system evolution
- Change implementation: an iteration of the development process



Program Evolution Dynamics

- Lehman and Belady (laws which apply to all systems as they evolved.)

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.
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 multi agent, multi loop feedback systems and you have to treat them as feedback systems to achieve significant product improvement.

Software Maintenance

Software Maintenance

- Focus on **modifying a program after it has been put into use**.
  - Does not normally involve major changes to the system's architecture.
  - Rather, modify existing components and add new components to the system.
- Types of software maintenance:
  - Maintenance to **repair software faults**: changing a system to correct faults 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 Costs

- Maintenance costs are usually greater than development costs
  - 2x to 100x depending on the application
- Can be affected by both technical and non-technical 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

- Concerned with assessing **which** parts of the system **may cause problems** and have high maintenance costs.
- Predicting **the number of changes** requires and understanding of the **relationships** between a **system** and its **environment**.
  - Number and complexity of system interfaces;
  - Number of inherently volatile system requirements;
  - The business processes where the system is used.

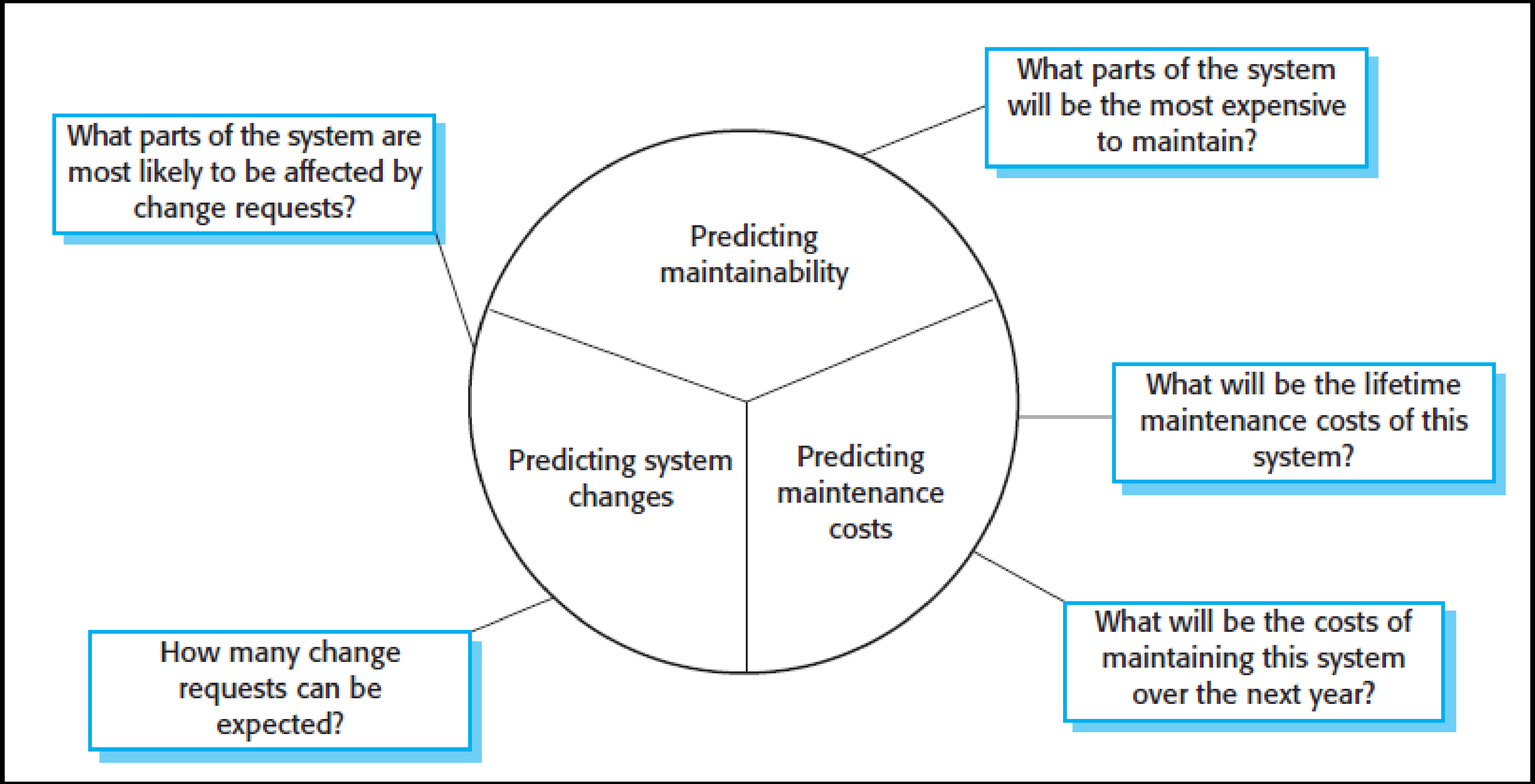
Maintainability Prediction

- Assess the **complexity** of system components.
    - 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.

Maintainability Prediction (Process Metrics)

- Number of requests for corrective maintenance
  - An increase in this may indicate that more errors are being introduced into the program than are being repaired during the maintenance process.
- Average time required for impact analysis
  - An increase in this may imply more and more components are affected
- Average time taken to implement a change request
- Number of outstanding (unfinished) change requests

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
Predicting system changes
Predicting maintenance costs
What will be the lifetime maintenance costs of this system?
How many change requests can be expected?
What will be the costs of maintaining this system over the next year?

Software Reengineering

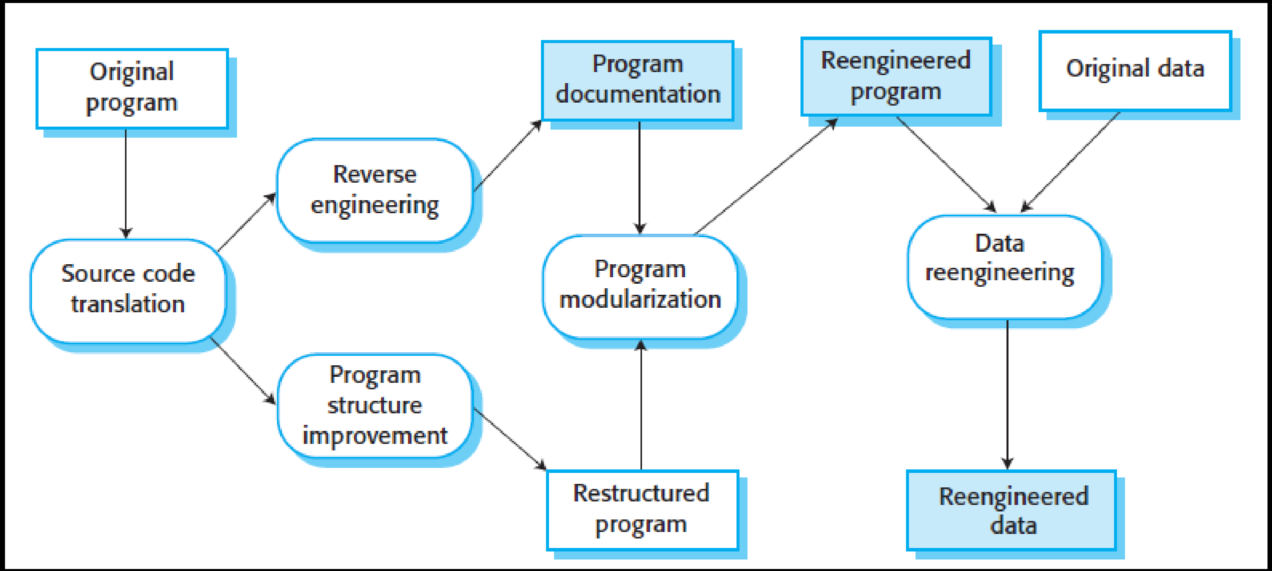
Software Reengineering

- Restructure or rewrite part or all of a legacy system **without changing its functionality**.
  - Redocument the system
  - Refactor the system architecture (avoid big changes)
  - Translate programs to a modern programming language
  - Modify and update the structure and values of the system's data
- Benefits:
  - Reduce risk
  - Reduce cost

University of Nottingham UK | CHINA | MALAYSIA

Software Reengineering Process

- **Source code translation**: convert code to a new language;
- **Reverse engineering**: analyze the program to understand it;
- **Program structure improvement**: restructure for understandability;
- **Program modularization**: reorganize the program structure;
- **Data reengineering**: clean-up and restructure system data.



Refactoring

University of Nottingham UK | CHINA | MALAYSIA

Refactoring

- Make improvements to a program to slow down degradation through change.
  - **Preventative maintenance** that reduces the problems of future change.
- Modify a program to **improve its structure**, reduce its complexity or make it easier to understand.
  - Not to add functionality but rather concentrate on program improvement.
- **Continuous process** of improvement throughout the development and evolution process.

Bad Smells of Code

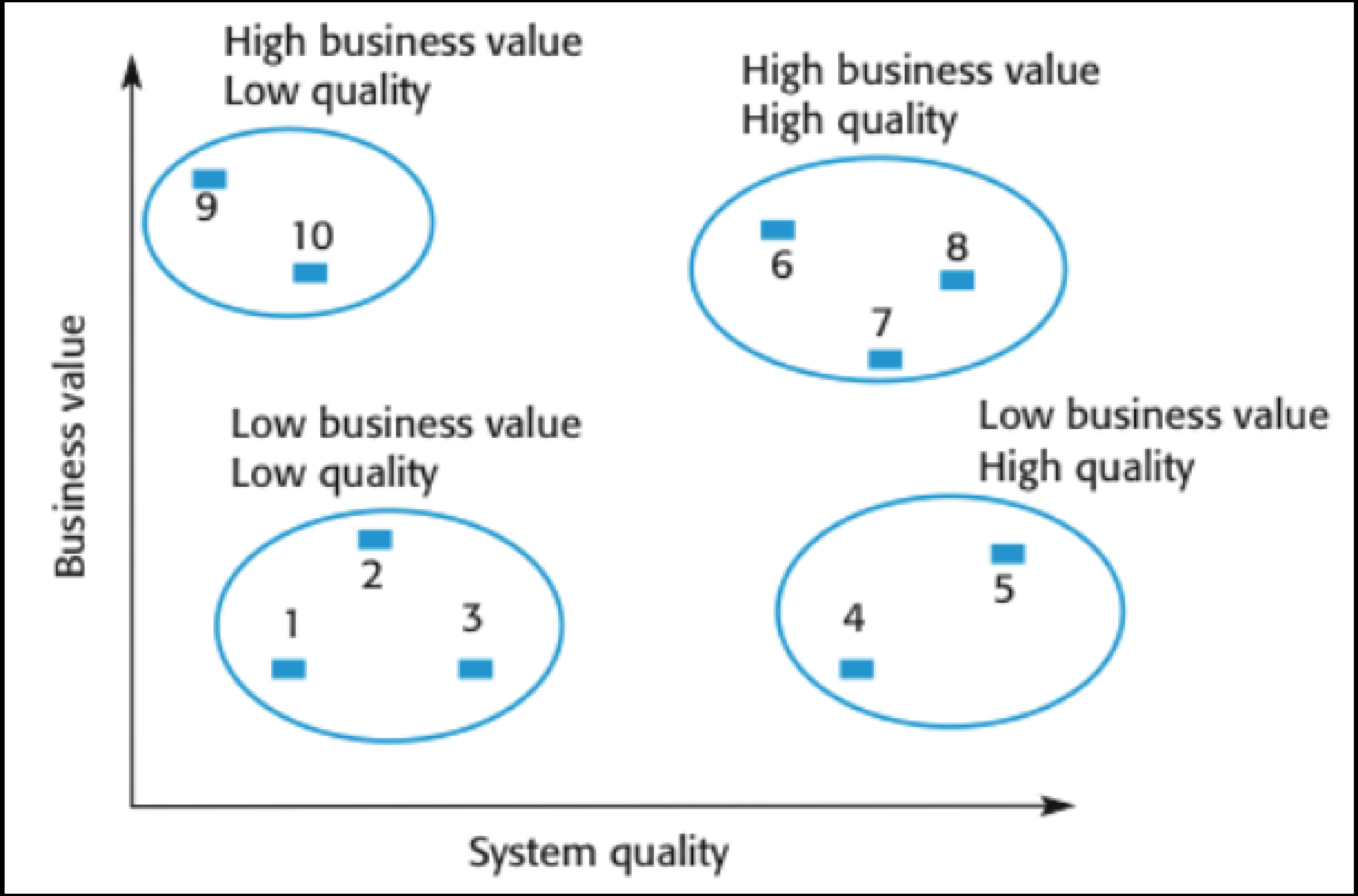
- **Duplicate code**:
  - Very similar code may be included at different places in a program; it can be removed and implemented as a single method or function that is called as required.
- **Long methods**:
  - Should be redesigned as a number of shorter methods.
- **Switch (case) statements**:
  - Often involve duplication, where the switch depends on the type of a value or the switch statements may be scattered around a program. Use polymorphism in OO (inheritance).

Bad Smells of Code

- **Data clumping**:
  - Same group of data items (fields in classes, parameters in methods) re-occur in several places in a program. Can be replaced with an object that encapsulates all of the data.
- **Speculative generality**:
  - Occurs when developers include generality (fields, methods, parameters, etc) in a program in case it is required in the future.

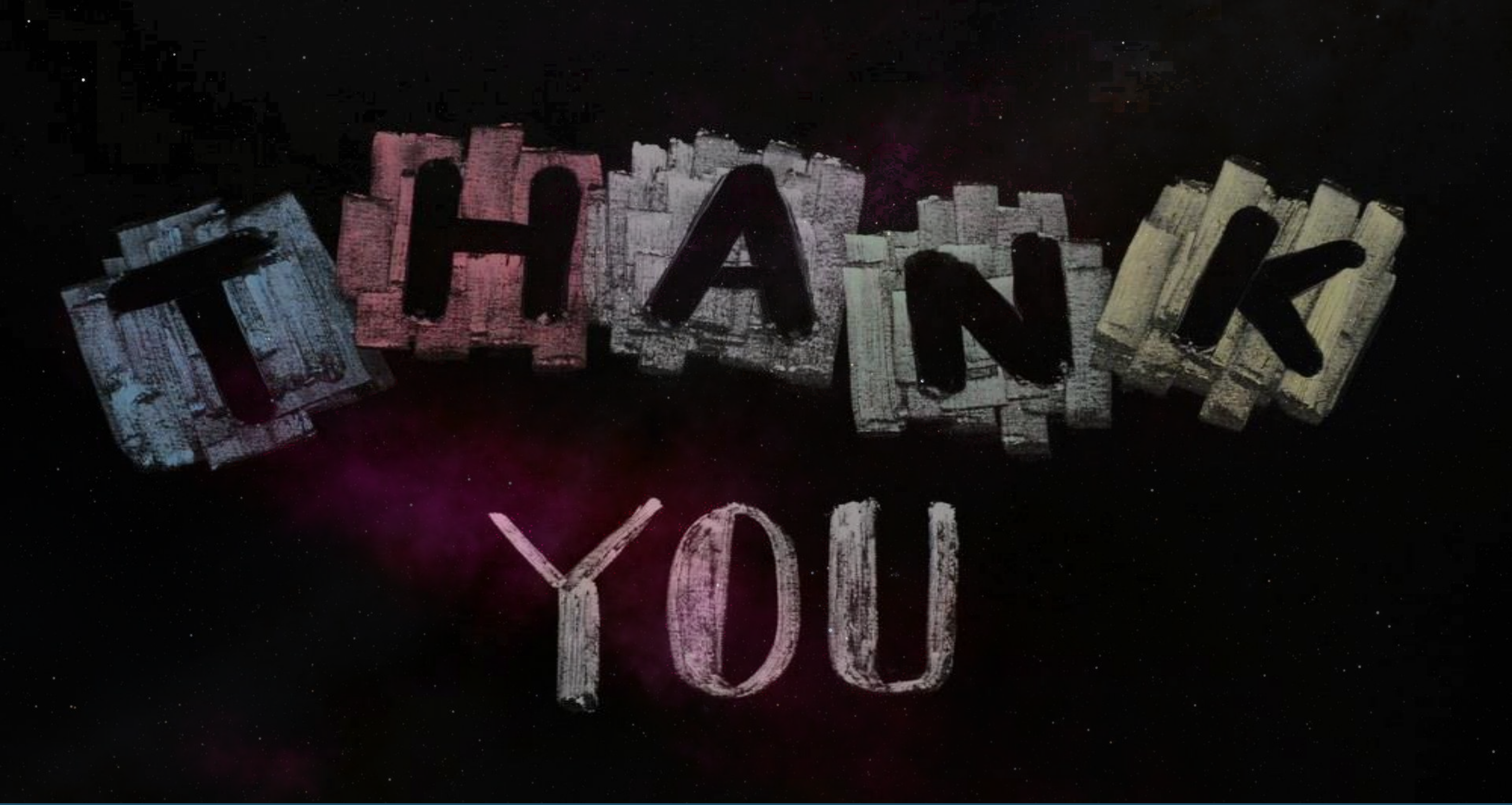
Legacy System Management

- **Low quality**, **low business value**: should be scrapped.
- **Low-quality, high-business value**: 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.



Summary

- Software Evolution Process
  - Program evolution dynamics
- Software Maintenance
  - Maintenance cost, maintenance prediction
  - Software reengineering
  - Refactoring, “bad smells of code”
- Legacy System Management


THANK
YOU