Software Re-Engineering

Lecture: 03

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

Sequence [Todays Agenda]

Content of Lecture

- Why Reengineer?
  - Lehman's Laws
  - Object-Oriented Legacy
- Typical Problems
  - Common symptoms
  - Architectural problems & refactoring opportunities
- Reverse and Reengineering
  - Definitions
  - Techniques

Legacy System

- ❑Legacy:
  - ❑A sum of money, or a specified article, given to another by will; anything handed down by an ancestor or predecessor.
- ❑Legacy in Computing:
  - ❑A legacy system is an old method, technology, , computer system, or application program of relating to or being a previous or outdated system, still in use.

Problems with Legacy Systems

- ❑ Typical problems with legacy systems:
  - ❑ Original developers not available
  - ❑ Outdated development methods used
  - ❑ Extensive patches and modifications have been made
  - ❑ Missing or outdated documentation

So, the further evolution and development may be prohibitively expensive.

Software Maintenance Cost

- Traditionally it is estimated that between 60% to 80% of development effort is spent on maintenance during the lifetime of a software solution and only 20% to 40% on new development.
- In the old waterfall approach, the maintenance phase started after initial delivery.
- In a modern agile environment, software delivery is a more constant process, happening in smaller increments.

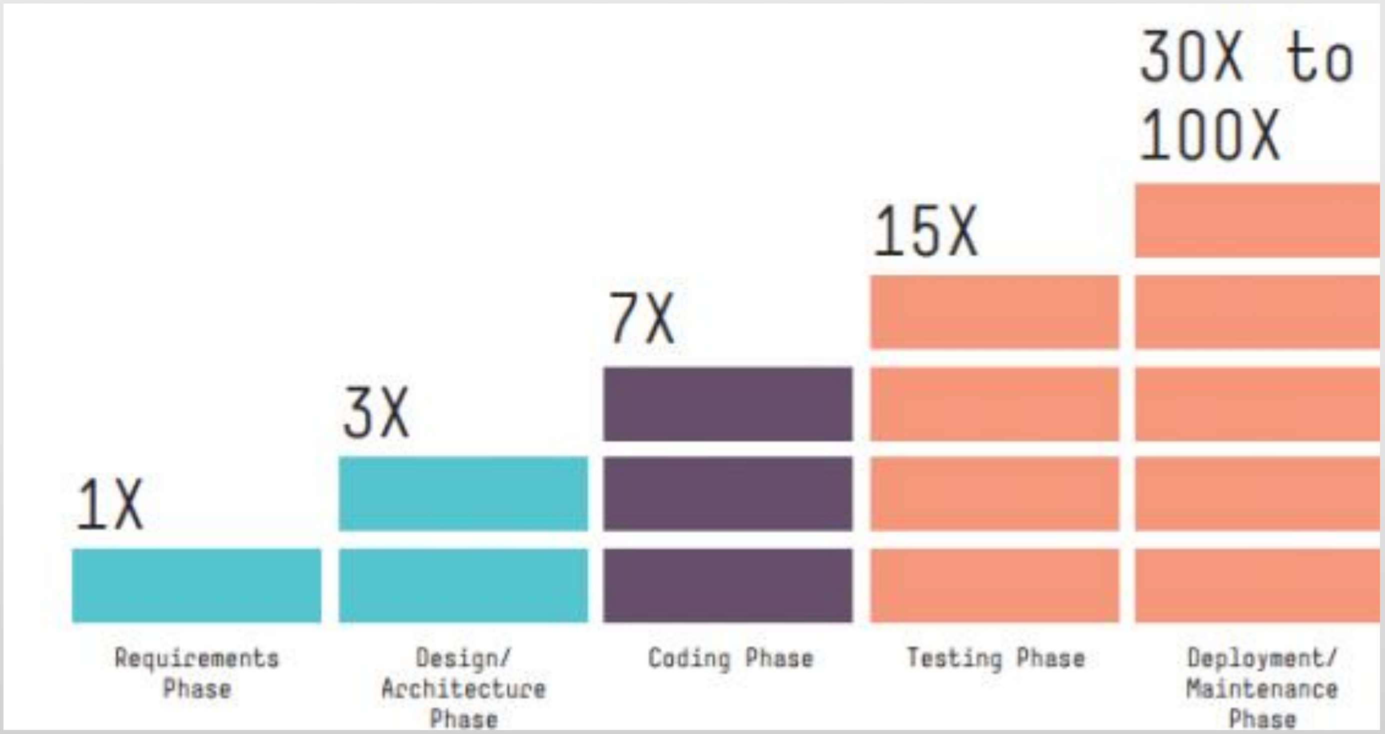
Maintenance Cost Calculation

- ❑ Software maintenance costs primarily depend on software's maturity level.
- ❑ The higher the frequency and volume of updates, the higher the costs will be.
- ❑ Other factors influencing maintenance costs include:
  - ❑ Software infrastructure and purpose.
  - ❑ Architecture complexity.
  - ❑ The number of users and integrations.
  - ❑ The range of maintenance activities.

Relative Cost of Fixing

- Costs for resolving defects become exponentially high if they are detected late in the development process.
- After a software system has been released, the costs for fixing a defect are up to 100 times higher compared to a defect detected in the early requirement phase.

Relative Cost of Fixing



- ❑ Solution ?

  Better requirements engineering
  Better software methods & tools
  (database schemas, CASE-tools, objects, components, etc.)

Continuous Development

- ❑Corrective maintenance (fixing reported errors)
  - ❑Around 20–25% of the total maintenance effort or more if the software is relatively new or still undergoing development.
- ❑Adaptive maintenance (new platforms or OS)
  - ❑About 15–20% of the maintenance effort or more if changes in the software environment or regulatory requirements are frequent.
- ❑Preventive maintenance
  - ❑Around 10–15% of the total maintenance effort.
- ❑Perfective maintenance (new functionality)
  - ❑Makes up 25–30% of the maintenance effort or more if there is a strong focus on enhancing the software's features and usability.

The bulk of the maintenance cost is due to **new functionality** even with better requirements engineering, it is hard to predict new functions.

Implementing Object-Oriented Programming in Legacy Systems

- ❑Assess the current system:
  - ❑Need to understand its current structure, functionality, and quality.
- ❑Define the OOP Goals and Scope:
  - ❑OOP features and principles should be prioritized that are most relevant such as encapsulation, inheritance, polymorphism, abstraction, and design patterns. Scope and boundaries should also be defined as which modules, classes, or functions you want to refactor or redesign with OOP.
- ❑Apply OOP Principles Incrementally:
  - ❑Common techniques for applying OOP incrementally include extracting methods to reduce code duplication and improve readability, introducing classes to encapsulate data and behavior, and replacing conditionals with polymorphism to avoid long and complex conditional statements.
- ❑Refactor and Redesign with OOP Patterns:
  - ❑To solve the issue of design flaws or anti-patterns, the code should be refactored and redesigned with OOP patterns that improve the design quality.
  - ❑Some of the common patterns to consider are Adapter, Facade, and Strategy.

How to Deal with Legacy Systems

- New or changing requirements will gradually degrade original design unless extra development effort is spent to adapt the structure.
- Best practices to deal with Legacy Systems:
- Understand the System: (How it works, what are its dependencies, and what are its strengths and weaknesses.)
- Refactor the Code: (Helps to deal with legacy systems by removing complexity, duplication, and technical debt, by applying consistent coding standards, naming conventions, and design patterns.)
- Isolate the System: (Helps to protect from unauthorized access, malicious attacks, or unintended side effects, by enforcing security policies, validation rules, and error handling.)
- Integrate the System: (with new or existing system, in order to provide new functionality, improve performance, or enhance user experience.)

Common Symptoms

- ❑Lack of Knowledge
- ❑Process Symptoms
- ❑Code Symptoms

- ❑Lack of Knowledge:
  - ❑Obsolete or No documentation
  - ❑Departure of the original (developers or users)
  - ❑Disappearance of inside knowledge about the system
  - ❑Limited understanding of entire system

- ❑Process Symptoms:
  - ❑Too long to turn things over to production
  - ❑Need for constant bug fixes
  - ❑Maintenance dependencies
  - ❑Difficulties separating products
  - ❑Simple changes take too long

- ❑ Code Symptoms:
    - ❑ Duplicated code
    - ❑ Code smells

Common Problems

- ❑Architectural Problems
- ❑Refactoring Opportunities

Architectural Problems

- Insufficient documentation (non-existent or out-of-date)
- Improper layering (lack of modularity, strong coupling)
- Duplicated code (copy, paste & edit code)
- Duplicated functionality (similar functionality by separate teams)

Refactoring Opportunities

- ❑ Misuse of Inheritance (code reuse vs polymorphism)
- ❑ Missing Inheritance (duplication, case-statements)
- ❑ Misplaced operations (operations outside classes, violation of encapsulation)

Some Terminologies

- Forward Engineering:

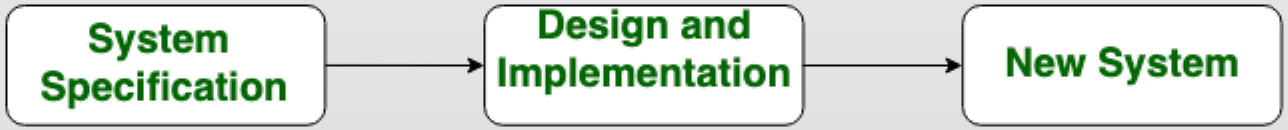
  Is the traditional process of moving from high- level abstractions and logical, implementation, independent designs to the physical implementation of a system."
- Forward engineering is known as Renovation and Reclamation.
- It requires high proficiency skills.
- It takes more time to construct or develop an application.
- Forward Engineering applies all the software engineering process that contains SDLC to recreate associated existing applications.
- It is near to full fill new needs of the users into re-engineering.

Some Terminologies

- ❑ Forward Engineering:

  Is the traditional process of moving from high- level abstractions and logical, implementation, independent designs to the physical implementation of a system.

- ❑ Forward engineering is also known as the Renovation and Reclamation.
- ❑ It requires high proficiency skills.
- ❑ It takes more time to construct or develop an application.
- ❑ Forward Engineering applies all the software engineering process that contains SDLC to recreate associated existing applications.
- ❑ It is near to full fill new needs of the users into re-engineering.



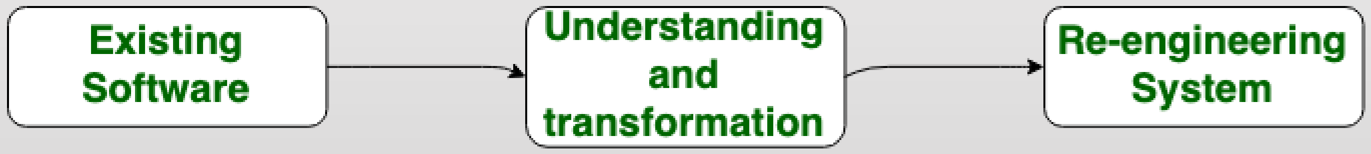
Forward Engineering

Some Terminologies

- ❑Reverse Engineering:

It is the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction.

- ❑Reverse Engineering is also known as backward engineering.
- ❑It is the process of forward engineering in reverse.
- ❑In this process, the information is collected from the given or existing application.
- ❑It takes less time than forward engineering to develop an application.
- ❑In reverse engineering, the application is broken to extract knowledge or its architecture.



Reverse Engineering

Some Terminologies

- Reengineering:

  It is the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form.

Goals of Reverse Engineering

Cope with complexity

- ❑ Need techniques to understand large, complex systems

Generate alternative views

- ❑ Automatically generate different ways to view systems
- ❑ Recover lost information
- ❑ Extract what changes have been made and why

Detect side effects

- ❑ Help understand ramifications of changes

Synthesize higher abstractions

- ❑ Identify latent abstractions in software

Facilitate reuse

- ❑ Detect candidate reusable artifacts and components

Reverse Engineering Techniques

- Redocumentation:

Redocumentation is the simplest and oldest form of reverse engineering, and many consider it to be a weak form of restructuring. The “re-” prefix implies that the intent is to recover documentation about the subject system that existed or should have existed.

Reverse Engineering Techniques

- ❑Design Recovery:

Design recovery recreates design abstractions from a combination of code, existing design documentation (if available), personal experience, and general knowledge about problem and application domain.

Goals of Reengineering

Unbundling

- Unbundle the software system into subsystems that can be tested, delivered and marketed separately.

Performance

- “first do it, then do it right, then do it fast” — experience shows this is the right sequence!
- Improving performance is sometimes a goal and sometimes considered as a potential problem once the system is reengineered.

Port to other Platform

- the architecture must distinguish the platform dependent modules

Goals of Reengineering

Design extraction:

- Always a necessary step in understanding the system; sometimes even an explicit reengineering goal.
- to improve maintainability, portability, etc.

Exploitation of New Technology

- i.e., new language features, standards, libraries, etc.

Reengineering Techniques

- Restructuring
  - automatic conversion from unstructured to structured code
  - source code translation
- Data reengineering
  - integrating and centralizing multiple databases
  - unifying multiple, inconsistent representations
  - upgrading data models
- Refactoring
  - renaming/moving methods/classes etc.

Summary

Software “maintenance” is really continuous development

Object-oriented software also suffers from legacy symptoms

Reengineering goals differ; symptoms don’t

Common, lightweight techniques can be applied to keep software healthy

Thank You!