Reengineering Object-Oriented Applications

Goals

We will try to convince you:

- Yes, Virginia, there are object-oriented legacy systems too!
- Reverse engineering and reengineering are essential activities in the lifecycle of any successful software system. (And especially OO ones!)
- There is a large set of lightweight tools and techniques to help you with reverse engineering.
- Despite these tools and techniques, people must do job and they represent the most valuable resource.

Lehman’s Laws

A classic study by Lehman and Belady [Lehm85a] identified several “laws” of system change.

Continuing change

- A program that is used in a real-world environment must change, or become progressively less useful in that environment.

Increasing complexity

- As a program evolves, it becomes more complex, and extra resources are needed to preserve and simplify its structure.

- Those laws are still applicable...

What is a Legacy System?

“legacy”

A sum of money, or a specified article, given to another by will, anything handed down by an ancestor or predecessor. — Oxford English Dictionary

A legacy system = a piece of software that:

- you have inherited, and
- is valuable to you.

Typical problems with legacy systems:

- original developers not available
- outdated development methods used
- extensive patches and modifications have been made
- missing or outdated documentation

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

What about Objects?

Object-oriented legacy systems

- = successful OO systems whose architecture and design no longer respond to changing requirements

Compared to traditional legacy systems

- The symptoms and the source of the problems are the same
- The technical details and solutions may differ

OO techniques promise better

- flexibility,
- reusability,
- maintainability
- ...

Modern Methods & Tools?


- Modern methods(*) lead to more reliable software
- Modern methods lead to less frequent software repair and...
- Modern methods lead to more total maintenance time

Contradiction? No!
- modern methods make it easier to change — this capacity is used to enhance functionality!

(*) process-oriented structured methods, information engineering, data-oriented methods, prototyping, CASE tools - not OO!
Some Terminology

“Forward Engineering is the traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system.”

“Reverse Engineering 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.”

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

— Chikofsky and Cross [in Arnold, 1993]

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

— Chikofsky and Cross [in Arnold, 1993]
Goals of Reengineering

- **Unbundling**
  - split a monolithic system into parts that can be separately marketed
- **Performance**
  - “First do it, then do it right, then do it fast” — experience shows this is the right sequence!
- **Port to other Platform**
  - the architecture must distinguish the platform dependent modules
- **Design extraction**
  - to improve maintainability, portability, etc.
- **Exploitation of New Technology**
  - i.e., new language features, standards, libraries, etc.

Reverse engineering Patterns

**Reverse engineering patterns** encode expertise and trade-offs in extracting design from source code, running systems and people.

- Even if design documents exist, they are typically out of sync with reality.

**Example:** Interview During Demo

Reengineering Patterns

**Reengineering patterns** encode expertise and trade-offs in transforming legacy code to resolve problems that have emerged.

- These problems are typically not apparent in original design but are due to architectural drift as requirements evolve

**Example:** Move Behaviour Close to Data

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

The Reengineering Life-Cycle

- **Reengineering techniques**
  - automatic conversion from unstructured to structured code
  - source code translation

---

A Map of Reengineering Patterns

- **Tests:** Your Life Insurance
- **Detailed Model Capture**
- **Migration Strategies**
- **Detecting Duplicated Code**
- **Redistribute Responsibilities**
- **Transform Conditional to Polymorphism**

---

Reengineering Life-Cycle

- **Requirements**
- **Design**
- **Model Capture**
- **Program Transformation**

---

DILBERT