Software Reverse Engineering

Moumita Asad

IIT, DU

Software Reverse Engineering

- a broad term that encompasses an array of methods and tools to derive information and knowledge from existing software artifacts and leverage it into software engineering processes

When Reverse Engineering is Required?

- The original programmers have left the organization
- The language of implementation has become obsolete, and the system needs to be migrated to a newer one
- There is insufficient documentation of the system
- The business relies on software, which many cannot understand
- The company acquired the system as part of a larger acquisition and lacks access to all the source code
- The system requires adaptations and/or enhancements
- The software does not operate as expected

Purpose and Objectives of Reverse Engineering

- The goal of reverse engineering is to facilitate change by allowing a software system to be understood in terms of what it does, how it works and its architectural representation
- The objectives in pursuit of this goal are to

  - ✓ recover lost information
  - ✓ facilitate migration between platforms,
  - ✓ extract reusable components,
  - ✓ cope with complexity
  - ✓ assist migration to a CASE environment
  - ✓ improve and/or provide new documentation
  - ✓ reduce maintenance effort
  - ✓ detect side effects
  - ✓ develop similar or competitive products

Recover Lost Information

- With time, a system undergoes a series of changes. Because of such things as management pressure and time constraints, the corresponding documentation for the requirements specification and design may not be kept up to date and may not even exist. This makes the code the only source of information about the system.
- Reverse engineering tools allow this information (requirements specification and design) to be recovered.
- Example: recovering specification would be in a specification language such as Z, and representing design as data flow diagrams, control flow diagrams, and entity-relationship diagrams.

Facilitate Migration between Platforms

- In order to take advantage of a new software platform or hardware, a combination of reverse and forward engineering can be used.
- The specification and design are abstracted using reverse engineering tools. Forward engineering is then applied to the specification according to the standards of the new platform.

Improve or Provide Documentation

- One of the major problems with legacy systems is insufficient, out-of-date or non-existent documentation.
- During redocumentation, tools can be used to augment inadequate documentation or to provide new.

Provide Alternative Views

- Redocumentation tools can be used to provide alternative documentation such as data flow diagrams, control flow diagrams and entity-relationship diagrams in addition to the existing documentation.
- This is a means whereby other views of the system can be obtained. For example, data flow diagrams portray the system from the point of view of data flow within the system and outside. Control flow diagrams, on the other hand, show the system from the perspective of the flow of control between the different components.

Extract Reusable Components

- The use of existing program components can lead to an increase in productivity and improvement in product quality, the concept of reuse has increasingly become popular amongst software engineers.
- Success in reusing components depends in part on their availability.
- Reverse engineering tools and methods offer the opportunity to access and extract program components.

Cope with Complexity

- One of the major problems with legacy systems is that as they evolve, their complexity increases.
- In the event of a modification, this complexity must be dealt with by abstracting system information relevant to the change and ignoring that which is irrelevant.
- Reverse engineering tools together with CASE tools provide the maintainer with some form of automated support for both function and data abstractions

Detect Side Effects

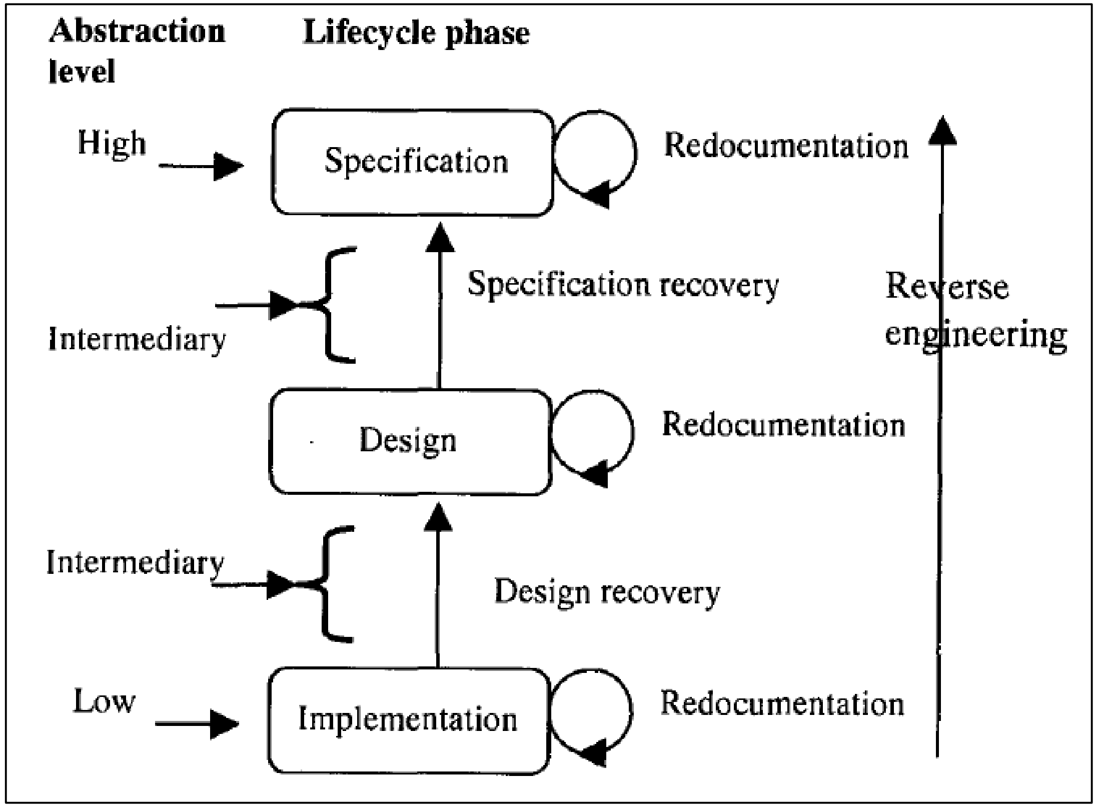
- In cases where the maintainer lacks a global view of the system, undesired side effects are caused and anomalies go unnoticed.
- Reverse engineering tools can make the general architecture of the system visible, thereby making it easier to predict the effect of change and detect logic and data flow problems.

Reduce Maintenance Effort

- This has been one of the main driving forces behind the increasing interest in reverse engineering.
- A large percentage of the total time required to make a change goes into understanding programs. The two main reasons for this are lack of appropriate documentation and insufficient domain knowledge.
- Reverse engineering has the potential to alleviate these problems and thus reduce maintenance effort because it provides a means of obtaining the missing information.

Levels of Reverse Engineering

- The product of a reverse engineering process does not necessarily have to be at a higher level of abstraction.
- If it is at the same level as the original system, the operation is commonly known as redocumentation.
- If on the other hand, the resulting product is at a higher level of abstraction, the operation is known as design recovery or specification recovery.



Redocumentation

- The recreation of a semantically equivalent representation within the same relative abstraction level.
- The goals of this process are threefold:
  1. to create alternative views of the system so as to enhance understanding, e.g., the generation of a hierarchical data flow or control flow diagram
  2. to improve current documentation
  3. to generate documentation for a newly modified program, aiming at facilitating future maintenance work on the system (preventive maintenance).

Design Recovery

- Entails identifying and extracting meaningful higher level abstractions beyond those obtained directly from examination of the source code.
- This may be achieved from a combination of code, existing design documentation, personal experience, and knowledge of the problem and application domains. The recovered design – which is not necessarily the original design - can then be used for redeveloping the system.
    - The resulting design forms a baseline for future system modifications.
    - The design could also be used to develop similar but non-identical applications. For example, after recovering the design of a spelling checker application, it can used in the design of a spell checking module in a new word processing package.

Specification Recovery

- In some situations reverse engineering that only leads to the recovery of the design of the system may not be of much use to an organization or a software engineer. For example, where there is a paradigm shift and the design of the new paradigm has little or nothing in common with the design of the original paradigm, e.g., moving from structured programming to object-oriented programming.
- In this case, an appropriate approach is to obtain the original specification of the system through specification recovery.
- During this process, the specification can be derived directly from the source code or from existing design representations through backward transformations.
- Information obtained from other sources such as system and design documentation, previous experience, and problem and application domain knowledge can greatly facilitate the recovery process.

Reverse Engineering Tools

- IDA Pro, Hex Rays
- CFF Explorer
- API Monitor
- WinHex
- Hiew
- Fiddler
- Scylla
- Relocation Section Editor
- PEiD
- dnSpy

Techniques Used for Reverse Engineering

- Lexical analysis.
- Syntactic analysis.
- Control flow analysis.
- Data flow analysis.
- Program slicing.

Lexical Analysis

- breaking syntaxes into a series of tokens, by removing any whitespace or comments in the source code.
- keywords, constants, identifiers, strings, numbers, operators and punctuations symbols can be considered as tokens.
- advantage: lexical analyzers do not require complete code.

```
int value = 100;
```

- int (keyword)
- value (identifier)
- = (operator)
- 100 (constant)
- ; (symbol)

Syntax Analysis

- Two types of representations are used to hold the results of syntactic analysis: **parse tree** and **abstract syntax tree**.
- A **parse tree** contains details unrelated to actual program meaning, such as the punctuation, whose role is to direct the parsing process.
- Grouping parentheses are implicit in the tree structure, which can be pruned from the parse tree.
- Removal of those extraneous details produces a structure called an **Abstract Syntax Tree** (AST).
- An AST contains just those details that relate to the actual meaning of a program.

Control Flow Analysis

- The two kinds of control flow analysis are:

  1. **Intraprocedural:** It shows the order in which statements are executed within a subprogram.
  2. **Interprocedural:** It shows the calling relationship among program units.

Control Flow Analysis

Intraprocedural analysis:

- The idea of basic blocks is central to constructing a CFG.
- A basic block is a maximal sequence of program statements such that execution enters at the top of the block and leaves only at the bottom via a conditional or an unconditional branch statement.
- A basic block is represented with one node in the CFG, and an arc indicates possible flow of control from one node to another.
- A CFG can directly be constructed from an AST by walking the tree to determine basic blocks and then connecting the blocks with control flow arcs.

Control Flow Analysis

Interprocedural analysis:

- Interprocedural analysis is performed by constructing a call graph.
- Calling relationships between subroutines in a program are represented as a call graph which is basically a directed graph.
- Specifically, a procedure in the source code is represented by a node in the graph, and the edge from node f to g indicates that procedure f calls procedure g.
- Call graphs can be static or dynamic.
  - A dynamic call graph is an execution trace of the program. Thus, a dynamic call graph is exact, but it only describes one run of the program.
  - On the other hand, a static call graph represents every possible run of the program.

Data Flow Analysis

- CFA can detect the possibility of loops, whereas DFA can determine data flow anomalies.
- One example of data flow anomaly is that an undefined variable is referenced.
- Another example of data flow anomaly is that a variable is successively defined without being referenced in between.
- Data flow analysis enables the identification of code that can never execute, variables that might not be defined before they are used, and statements that might have to be altered when a bug is fixed.

Data Flow Analysis

- Control flow analysis cannot answer the question: Which program statements are likely to be impacted by the execution of a given assignment statement?
- To answer this kind of questions, an understanding of definitions (def) of variables and references (uses) of variables is required.
- If a variable appears on the left hand side of an assignment statement, then the variable is said to be defined.
- If a variable appears on the right hand side of an assignment statement, then it is said to be referenced in that statement.

Program Slicing

- Originally introduced by Mark Weiser, program slicing has served as the basis of numerous tools.
- A program slice is a portion of a program with an execution behavior identical to the initial program with respect to a given criterion, but may have a reduced size.
- In Weiser’s definition, a slicing criterion of a program P is S < p; v > where p is a program point and v is a subset of variables in P.
- A **backward slice** with respect to a variable v and a given point p comprises all instructions and predicates which affect the value of v at point p.
- **Backward slices** answer the question “***What program components might effect a selected computation?***”
- The dual of **backward slicing** is **forward slicing.**
- With respect to a variable v and a point p in a program, a forward slide comprises all the instructions and predicates which may depend on the value of v at p.
- **Forward slicing** answers the question “***What program components might be effected by a selected computation?***”

Example – Computation of backward slice using PDG

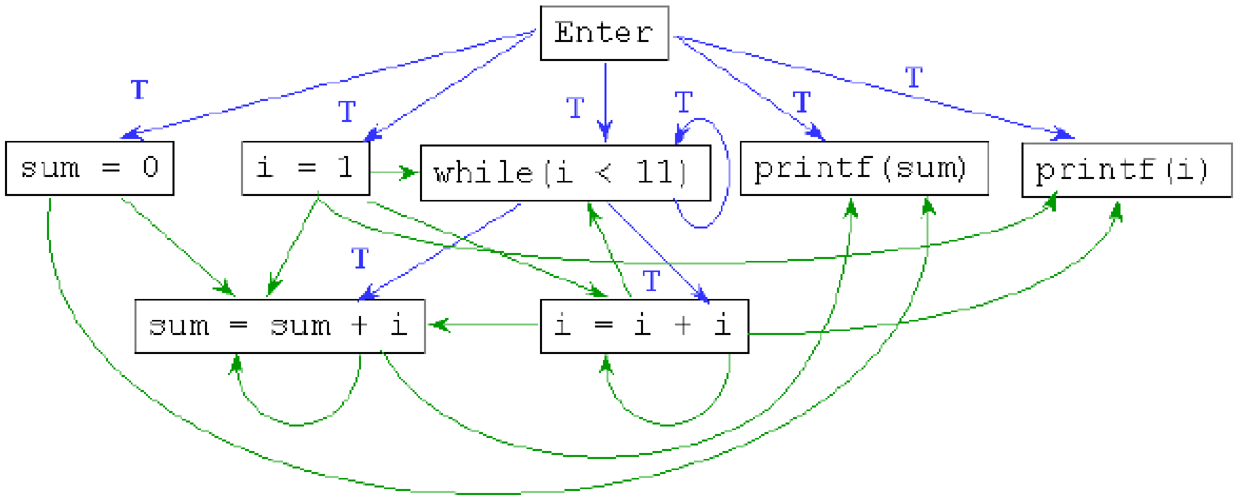
Backward slice from

`System.out.println(i);` :-

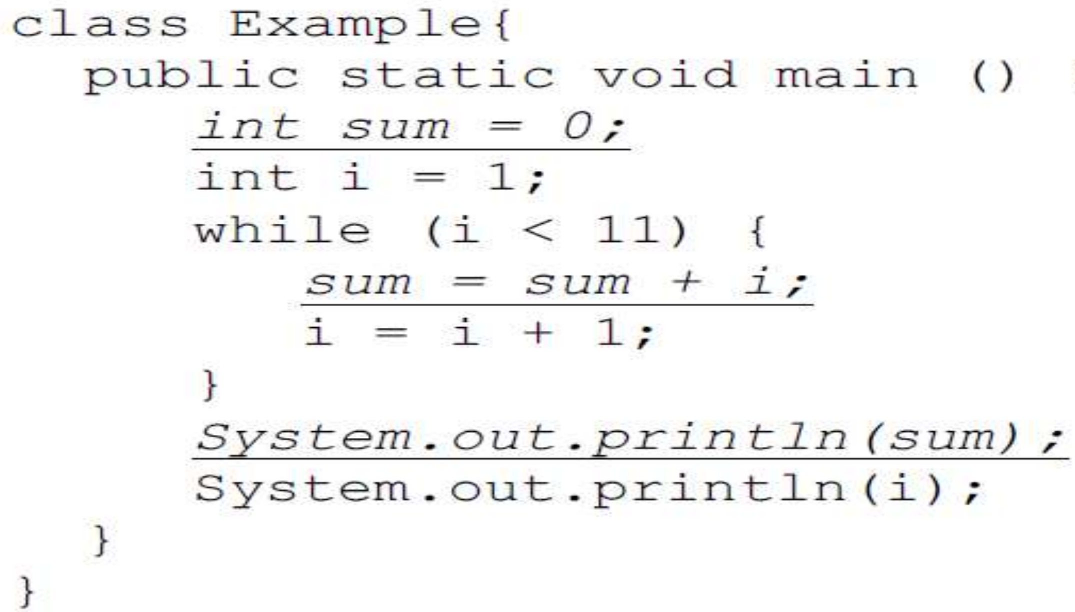
```
class Example{
  public static void main () {
        int sum = 0;
        int i = 1;
        while (i < 11) {
            sum = sum + i;
            i = i + 1;
        }
        System.out.println(sum);
        System.out.println(i);
    }
}
```

Control dependence

Flow dependence



Forward slice from the statement
`int sum = 0;` :-



```
class Example{
   public static void main () {
        int sum = 0;
        int i = 1;
        while (i < 11) {
            sum = sum + i;
            i = i + 1;
        }
        System.out.println(sum);
        System.out.println(i);
   }
}
```