Reverse Engineering

- ‘Trying to figure out the structure and behaviour of existing software by building general-level static and dynamic models’
- Links:
    - Compact information on reverse engineering
  - [http://users.ece.gatech.edu/~linda/revenge/revrepos.html](http://users.ece.gatech.edu/~linda/revenge/revrepos.html)
    - Reengineering Resource Repository
    - Listings of tools, literature, …
Applications

• Modifying software
  – Change of environment (software migration)
  – Re-designing software (re-engineering)
    • E.g. Y2K, €, e-commerce
• Design and implementation in forward engineering, e.g. debugging
• Program understanding/comprehension
• Program visualisation
• Software re-use

Data reverse engineering

• "Data reverse engineering focuses on data and data-relationships both among data structures within programs and data bases”
• For example: relational data bases (RDBs):

  flat/hierarchical files  ➔  RDB’s

  RDB’s  ➔  OO model
- OO model (objects, associations, inheritance, ...)
- keys
- optimizations
- ...

conceptual schema

abstraction
- reengineer

logical schema

analysis
- domain expert
- developer
- reengineer

physical schema

- data
- schema catalog
- code
- documentation

Data reverse engineering

• Redocumentation
• Restructuring
  – transforming a system from one representation to another, while preserving its external functional behavior
• Retargeting
  – transforming and hosting or porting the existing system in a new configuration

Other ’Re’ terms
More ’Re’ terms

• Business Process Reengineering
  – radical redesign of business processes to increase performance, such as cost, quality, service, and speed
  – reoptimization of organizational processes and structures
• Reverse specification
  – extracting a description of what the examined system does in terms of the application domain
  – a specification is abstracted from the source code or design description

Software reverse engineering

• Chikofsky & Cross: two-phase process
  – Collecting information
    • parsers, debuggers, profilers, event recorders
  – Abstracting information
    • Making understandable, high-level models
• “Programmers have become part historian, part detective, and part clairvoyant”
  (T.A.Corbì 1989)
Source code vs. binaries

- **Source code**
  - better form of representation
  - not always possible
  - result depends on the parser (notable differences)

- **Binaries**
  - faster information collection (e.g. Java byte code)
  - legality issues
Usage of binaries
(reverse engineering, decompilation, disassembly)

• Recovery of lost source code
• Migration of applications to a new hardware platform
• Translation of code written in obsolete languages not supported by compiler tools nowadays
• Determination of the existence of viruses or malicious code in the program
• Recovery of someone else's source code (to determine an algorithm for example)

Binary copyrights
(decompilation, disassembly)

• Not all countries implement the same laws!
• Commonly allowed by law
  – for the purposes of interoperability
  – for the purposes of error correction where the owner of the copyright is not available to make the correction
  – to determine parts of the program that are not protected by copyright (e.g. algorithms), without breach of other forms of protection (e.g. patents or trade secrets)
• The decompilation page:
Copyrights cont.

- EU: 1991 EC Copyright Directive on Legal Protection of Computer Programs provided extensions to copyright to permit decompilation in limited circumstances
- An example: Sony sued Connectix Corp (1999) for developing of its Virtual Game Station emulator, and emulator of the Sony developed PlayStation (Mac) -> a long fight over emulation rights and extent of copyright protection on computer programs

A decompilation example / 1

```java
public class MyTest {
    // This is a silly program.
    public static void main(String[] args) {
        int myInt1=1;
        int myInt2=2;
        for (int i=1;i<10;i++) {
            for (int j=2;j<8;j++)
                myInt1++;
            myInt2=myInt2+myInt1;
        }
        System.out.println("myInt1 is "+myInt1 + " and myInt2 is " + myInt2);
    }
}
```

-> Compiled with Sun's javac compiler and decompiled with DJ Java Decompiler, let's see what we got:
A decompilation example / 2

```java
import java.io.PrintStream;

public class MyTest {
    public MyTest() {
    }

    public static void main(String args[]) {
        int i = 1;
        int j = 2;
        for(int k = 1; k < 10; k++) {
            for(int l = 2; l < 8; l++) {
                i++;
            }
            j += i;
        }
        System.out.println("myInt1 is " + i + " and myInt2 is " + j);
    }
}
```

Static models

- Finding out the static structure, architecture
  - code (using a parser)
  - documents
  - interviews
- Visualisation:
  - class diagrams
  - (hierarchical) graphs
**Dynamic models**

- Finding out the run-time behaviour of software
  - debugger, profiler, source code instrumentation

- Visualisation:
  - scenarios (sequence diagrams)
  - State diagrams
  - (hierarchical) graphs

---

**Abstracting the static model**

- Abstracting the high-level components (like subsystems)

- The process can be made partly automatic
  - Automatic abstraction
    - Using the structure of the language
    - Using measurements

  - Manual abstraction
Metrics

- Numeric measurements from software (or software projects)
- *More on these later in this course*

**CodeCrawler:**
- a reverse engineering tool that combines metrics and graphs to visualize OO systems
- *http://www.iam.unibe.ch/~lanza/codecrawler/codecrawler.html*
Abstracting the dynamic model

- Finding behaviour patterns, repeating sequences of events
  - E.g. initialising a dialogue

- Using static abstractions
  - E.g. representing interactions between high-level software elements in sequence diagrams

- Dynamic information is combined with the high-level static model

<table>
<thead>
<tr>
<th>Merging static and dynamic information to a single view</th>
<th>Dynamic and static views</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Directly illustrates connections between static and dynamic info</td>
<td>- connections and correspondencies between the views need to be defined</td>
</tr>
<tr>
<td>+ Ensuring the quality of the view</td>
<td>+ both static and dynamic abstractions can be built</td>
</tr>
<tr>
<td>-polymorphism (OO) may cause confusion</td>
<td>+ static and dynamic views are separated also in forward engineering: support for re-engineering and round-trip engineering</td>
</tr>
<tr>
<td>- building abstractions becomes cumbersome and/or requires trade offs: behavioral patterns &lt;-&gt; subsystems</td>
<td>+ more information can be viewed</td>
</tr>
<tr>
<td>- sequential information is difficult to merge to a static view</td>
<td></td>
</tr>
<tr>
<td>- the more information a view contains, the less readable it gets</td>
<td></td>
</tr>
</tbody>
</table>
Analysing the static model

- Syntax, type checking, interfaces
- Control and data flow analysis
- Structure analysis
- Slicing and dicing (different ways to partition the software)
- Measuring the complexity
- Navigation

Analysing the dynamic model

- Object creation and related dependencies
- Dynamic binding, polymorphism
- Method calls
- Looking for dead code/reachability analysis
- Memory management
- Performance and related problems
- Concurrency
Reverse engineering for OO software

- Dynamic behavior may be hard to detect from static model (creating and deleting objects, garbage collection, dynamic binding,…)  
  -> this emphasises dynamic modelling
- Pure object languages support encapsulation (classes, packages,…)  
  -> helps in static reverse engineering  
  -> increases usability of metrics
- OO paradigm supports the use of design patterns  
  -> reusability applications (pattern recognition)

Round-trip engineering

- Forward and backward (reverse) engineering combined
- Most typical OO example: producing source code from class diagrams and class diagrams from source code.
- As another example, a design tool may support automatic (or mostly automatic) translation from ER-model to relational model and back.
Why round-trip engineering? / 2

- Assume that you first model your software using UML.
- Typically, it is possible to automatically generate source code files (say, Java) from a class diagram.
- Eventually someone will touch the source code in such a way that the class diagram is no longer valid and the classes are not to be re-generated from the class diagram.
- After that, you will just spend the rest of the project hoping that no-one will have a look at the class diagrams 😞
- Of course, you may manually update your class diagrams 😞 😞

Why round-trip engineering? / 3

- Some software development tools automatically generate source code.
- However, it may be that they do not generate the UML diagrams.
- Or, if they do, they may be in a format, which your UML design tools do not know how to read.
- Again, of course, you may manually update your class diagrams 😞
Tools

• Tools supporting creation of high-level models
• Tools supporting metrics
• Forward & reverse engineering
  – re-engineering & round-trip-engineering &testing
• Other tools
  – parser generators
  – design pattern recognition

Tools

• Rigi (University of Victoria, Canada)
  – a research prototype that represents an open and public
domain reverse engineering tool
  – user programmable
  – analysis for: C, C++, COBOL, PL/AS, LaTeX
• SNIFF+ (TakeFive Software)
  – a software development environment that also provides
reverse engineering capabilities
Tools

- McCabe’s Visual Reengineering Toolset and Visual Quality Toolset
  - various views
  - software metrics (complexity and structuredness)
    - shown as specific colors on the views
- Logiscope (CS Verilog)
  - reverse eng, code testing, static and dynamic testing, metrics
  - analysis for: C, C++, Java, ADA
- ESW (Viasoft Inc.)
  - forward and reverse engineering (maintenance), metrics, testing

Tools

- Refine (Reasoning Systems Inc.)
  - an open and programmable tool that works in the Refinery environment
    - tools for generating source code parsing and conversion tools
  - features for analyzing and re-engineering code
  - analysis for: Ada, C, Cobol
- Imagix4D (Imagix Corp.)
  - a closed tool that provides a large set of built-in functionalities
  - several views (also 3D)
  - analysis for: C/C++
Tools for OO languages

• Produce a class diagram from code
  – Rational Rose (Rational Software Corp.)
  – Paradigm Plus (Computer Associates International)
  – OEW (Innovative Software GmbH)
  – Graphical Designer (Advanced Software Technologies Inc.)
  – Domain Objects (Domain Objects Inc.)
  – COOL:Jex (Sterling Software Inc.)
  – Fujaba (Paderborn University)
  – ...