



Software Analysis and Testing (II)

Lezione 1

Filippo Ricca

ITC-Irst

Istituto per la ricerca
Scientifica e Tecnologica

ricca@itc.it



Legacy systems

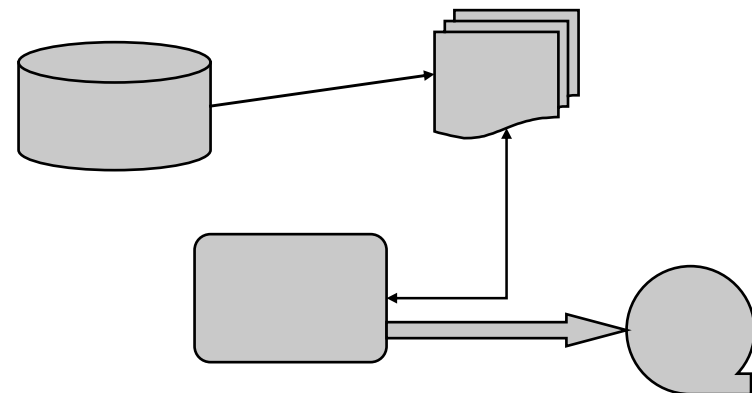
- They were implemented years ago
- Their technology became obsolete (obsolete languages, hardware)
- They have been maintained for a long time
- Their structure is deteriorated and does not facilitate understanding
- Their documentation (if it exists) became obsolete
- Original authors are not available
- They contain business rules not recorded elsewhere
- They can not be easily replaced
- They represent a large investment
- ...

Reverse engineering

- **Reverse engineering** is a process that helps understanding the software system. It is a **process of examination**, of extracting information, not a process of change or replication.

Software -----> “Abstract representation”

Software ----->



Restructuring

Restructuring is the transformation from one representation to another at the same relative abstraction level, while preserving the system external behavior (functionality and semantics).

Examples:

- **Code-to-code transform:** from an unstructured (“spaghetti”) form to a structure form (goto-less), or conversion of set of “if-statements” into a “case structure”.
- **Design level:** to improve data structures or algorithms (time complexity).



Re-engineering



Re-engineering is the **examination** (reverse engineering) of a system to **reconstitute** it (forward engineering) in a new form. This process may include modifications with respect to new requirements not met by the original system.

The re-engineering process takes many forms, depending on its objectives. Sample objectives are: code migration (ex. C to C++) reengineering code for reuse, ...



Objectives

➤ This course aims at providing the practical skills involved in software analysis and testing. Algorithms and techniques described during the theoretical lessons of the basic course (*Software Analysis and Testing I*) are applied to real cases of software systems to be re-engineered and tested.



Last Year Project

➤ Dato un programma P scritto in linguaggio C effettuare un operazione di “**code migration**” (Re-engineering) trasformando P in un nuovo programma P' scritto in linguaggio Java. P e P' dovranno avere la stessa semantica (*per ogni i :input $P[i] = P'[i]$*).



Last Year Steps

1. Instrumentare il codice C usando TXL. Scrivere Testcases t.c. vi sia la copertura dei comandi/branch.
2. Fase di reverse engineering. Ricavare alcuni diagrammi (call graph, dipendenza tra funzioni e strutture dati, ...).
3. Fase di **Object identification** (tecniche di clustering e concept analysis).
4. UML design.
5. Generazione di codice Java (usare TXL per trasformazioni ...).
6. Regression testing con i Testcases generati al punto 1.
7. Instrumentare il codice Java e aggiungere nuovi Testcases al fine di avere la copertura dei comandi/branch del nuovo codice.

What is TXL

◆ TXL is a programming language specifically designed to support software analysis and program transformation.

Loop

x := a + b;

y := x;

a := y + 3;

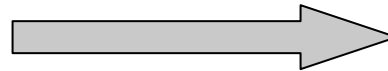
x := b + c;

y := x - 2;

z := x + a * y;

End loop

Code motion optimization



x4 := b + c;

y2 := x4 - 2;

Loop

x := a + b;

y := x;

a := y + 3;

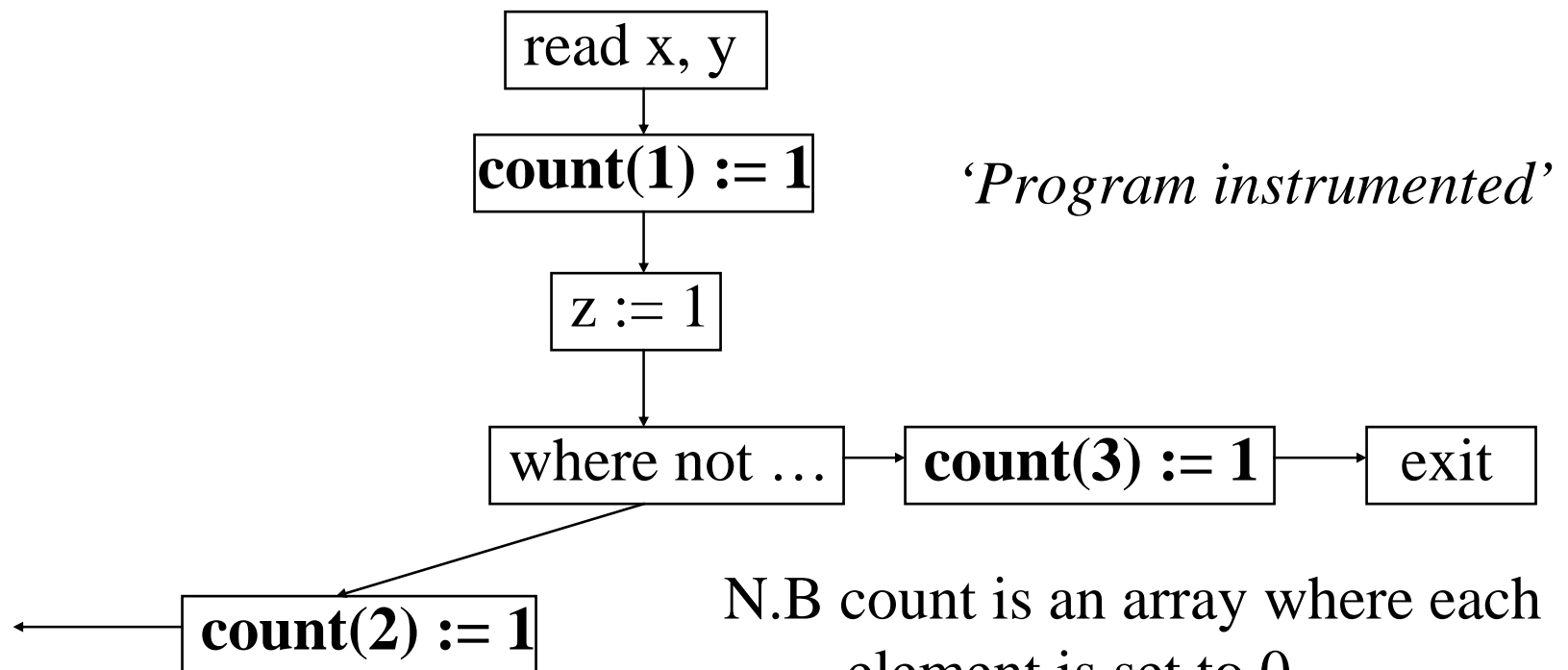
z := x4 + a * y2;

End loop

Example: `_` moves all loop-independent assignment statements outside of loops.

Code instrumented

To determine whether or not each branch is traversed, we can place a 'counter' on each branch.





Last Year Examination

The examination consists of the discussion of the course project.

Admission to the examination requires (at least) the production of the following documents:

- Architecture of the existing system.
- Object-oriented design of the new system.
- Implementation in Java of the new system.
- Testcases and testing results for both old and new system.



Idea for this Year ...

- Dato un programma P scritto in linguaggio JAVA effettuare un operazione di Re-structuring aggiungendo gli aspetti (Aspect Oriented Programming).
- L'idea è quella di trasformare P in un nuovo programma P' scritto in linguaggio **AspectJ** (estensione ad aspetti di Java). Il risultato dovrebbe essere un programma più semplice e leggibile.
- P e P' dovranno avere la stessa semantica (*per ogni i :input $P[i] = P'[i]$*).

AOP

Suppose a e-commerce project convention demands all developers to do logging by calling `Logger.entry(string)` and `Logger.exit(string)` for all function they write.

```
/** main without AOP */
```

```
Public class Main {  
    public void foo() {  
        Logger.entry("foo()")  
        .... Something ...  
        Logger.exit("foo()")  
    }  
    public void foo(int i) {  
        Logger.entry("foo(int)")  
        .... Something ...  
        Logger.exit("foo(int)")  
    }  
    public static void main(String [] args) {  
        Logger.entry("main()")  
        .... Something ...  
        Logger.exit("main()")  
    }  
}
```

```
/** main with AOP */
```

```
Public class Main {  
    public void foo() {  
        .... Something ...  
    }  
    public void foo(int i) {  
        .... Something ...  
    }  
    public static void main(String [] args) {  
        .... Something ...  
    }  
}
```

```
Public aspect autolog {  
    pointcut publicMethods(): ....
```

```
    Before(): publicMethods() { ... }
```

```
    After(): publicMethods() { ... }
```

```
}
```



Steps



1. Instrumentare il codice JAVA usando TXL. Scrivere Testcases t.c. vi sia la copertura dei comandi/branch.
2. Fase di reverse engineering. Ricavare alcuni diagrammi (class diagram, call graph, ...).
3. Aspect mining (individuare quelle parti di software che possono essere trasformate in aspetti)
4. Scrivere in TXL la grammatica AspectJ.
5. Generazione di codice AspectJ (usare TXL per trasformazioni ...).
6. Regression testing con i Testcases generati al punto 1.



Dependencies

---> Formal languages and compilers, Programming I and II, Software Engineering, Software Analysis and Testing.



Tools

- **TXL**: instrumentare il codice e trasformazioni
- **Dotty**: diagrammi/viste/astrazioni del codice
- **Junit**: Testcases e testing
- ...



Material

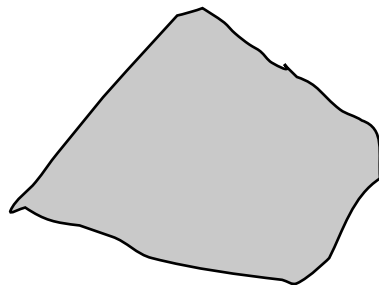
- Slides
- Papers
- Manuals of laboratory tools

Last Year Project

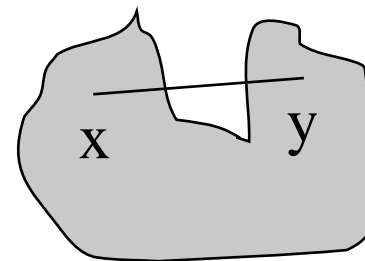
Il codice C scelto risolve il problema dell'involuppo convesso (convex hulls) in 3D.

Definiamo **involuppo convesso** di un insieme di punti P_i nello spazio come la più piccola regione convessa che contiene tutti i punti dati.

Un set S di punti è convesso se per ogni x e y appartenenti a S il segmento xy è contenuto in S .

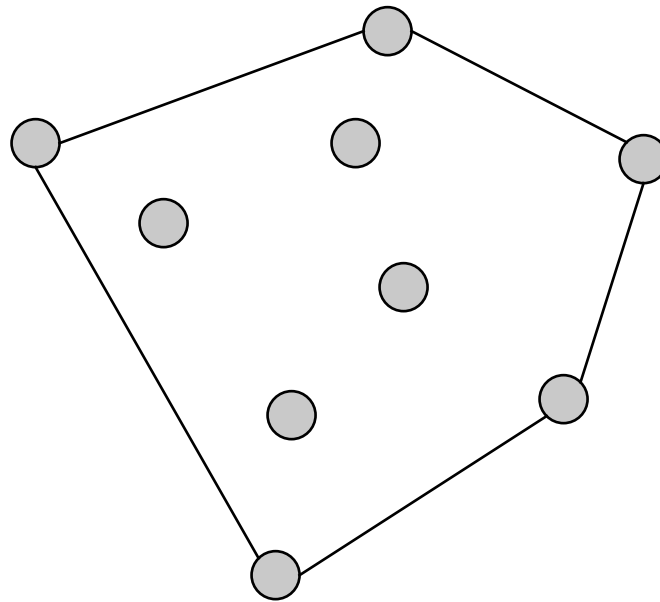


convex



No-convex

Example in 2D



---> Esistono moltissimi algoritmi che risolvono il problema dell'involuppo convesso.