Overview

• Motivation
• Problem
• Approach
• Experiments

Change Impact Analysis

• A collection of techniques for determining the effects of source code modifications
• It can improve programmer productivity by
  – determining tests whose behaviors may be affected
  – identifying portions of an edit that may affect such tests
  – potentially allowing developers to experiment with different portions of an edit

An Example

```java
public static void test1()
```

```java
A a = new A();
a.foo(); // A's foo
```

```java
public static void test2()
```

```java
A a = new B();
a.foo(); // B's foo
```

```java
public static void test3()
```

```java
A a = new C();
a.foo(); // A's foo
```

Chianti Approach Overview

• Given P, P', and regression test suites T
  – Derive atomic changes \( A = \{c_1, c_2, \ldots c_n\} \)
  – For each test in \( T \), construct dynamic call graph \( CG \) and \( CG' \).
  – If \( CG \neq CG' \) or \( CG \) has overlap with some changed entities, put \( t \) into \( T_a \)—a set of tests that are potentially affected by changes
  – For each test in \( T_a \), determine the subset of changes affecting the test

Non-locality of change impact in OO programs

• Small source code changes can have major and non-local effects in object-oriented systems
  – Due to subtyping and dynamic dispatch

```java
class A {
    public void foo(){ }
}
class B extends A {
    public static void bar(){y=17;}
    public static int y;
}
```

```java
public void foo() {
    B.bar();
}
```

```java
A x = new B();
```

```java
x.foo();
```

Chianti: A Tool for Change Impact Analysis of Java Programs [1]

Barbara G. Ryder
Imported by Na Meng
An Example

```java
class A {
    public void foo() {
    }
    public int x;
}
class B extends A {
    public void foo() {
        B.bar();
    }
    public static void bar() {
        y = 17;
    }
    public static int y;
}
class C extends A {
    public void foo() {
        x = 19;
    }
    public void baz() {
        z = 19;
    }
    public int z;
}
```

Questions:
(1) Which test is affected?
(2) For each affected test, what is the affecting change?

Derive Atomic Changes

- Categories of atomic changes: The granularity is roughly at the method level
  - AC Add an empty class
  - DC Delete an empty class
  - AM Add an empty method
  - DM Delete an empty method
  - LC Change virtual method lookup
  - AF Add a field
  - DF Delete a field

Atomic changes and their dependencies

- c2 depends on c1
  - c1 is prerequisite for c2 to guarantee syntactic correctness

```
class A {
    public void foo() {
    }
    public int x;
}
class B extends A {
    public void foo() {
        B.bar();
    }
    public static void bar() {
        y = 17;
    }
    public static int y;
}
```

- LC change: (Y, X.m())
  - A call to method X.m() on an object whose runtime type is Y, is dispatched differently.

Affected Tests

```
public static void test2() {
    A a = new B();
    a.foo(); //B's foo
}
```

```
public static void test3() {
    A a = new C();
    a.foo(); //C's foo
}
```

Atomic changes and their dependencies

```
class A {
    public void foo() {
    }
    public int x;
}
class B extends A {
    public void foo() {
        B.bar();
    }
    public static void bar() {
        y = 17;
    }
    public static int y;
}
class C extends A {
    public void foo() {
        x = 19;
    }
    public void baz() {
        z = 19;
    }
    public int z;
}
```

Affected Tests
Affecting Changes

```
public static void test2(){
    A a = new B();
    a.foo(); // B's foo
}
```

class B extends A {
    public void foo()
    {
        B.bar();
    }
    public static void bar()
    {
        y = 17;
    }
    public static int y;
}

Affecting Tests

\[\text{Affected Tests (T, A)} = \begin{cases}
    (T_i | T_i \in T, (\text{Nodes}(P, T_i) \cap (\text{CM} \cup \text{AM})) \neq \emptyset) \\
    (T_i | T_i \in T, n \in \text{Nodes}(P, T_i), n \rightarrow_b X, A.m \in \text{Edges}(P, T_i), \text{for some } n, A.m \in \text{Nodes}(P, T_i), a' < a)\end{cases}\]

Experiments

- Data: Daikon project (cf. M. Ernst, MIT)
  - Obtained CVS repository from 2002 with version history - an active debugging period
  - Grouped code updates within same week to form edit intervals
  - Obtained 39 intervals with code changes
- Measurements: numbers of affected tests and their affecting changes per edit interval
- Platform: Pentium 4 PC at 2.8Ghz with 1Gb RAM.

Daikon

- Code base growth in 2002
  - From 48K to 121K lines of code: 357 to 765 classes; 2409 to 6284 methods; 937 to 2885 fields
- Unit test suite used
  - 40-62 unit tests per version
  - Collected dynamic call graphs of tests
  - Achieved on average of 21% coverage of methods, but higher coverage of the mde library (47%)
Performance of Chianti

- Deriving atomic changes from 2 successive versions takes on average **87 secs**
  - Median 70 secs, max 343 secs
- Calculating the set of affected tests takes on average **5 secs**
  - Median of 2.5 secs, max of 35 secs

Reference

[1] Xiaoxia Ren, Fenil Shah, Frank Tip, Barbara G. Ryder, and Ophelia Chesley, Chianti: A Tool for Change Impact Analysis of Java Programs, OOPSLA '04