Testing Evolving Software

Alessandro (Alex) Orso
School of Computer Science - College of Computing

Software Engineering
Static/Dynamic Program Analysis,
Software Testing, Security

Partially supported by: NSF, IBM Research, TCS Ltd., Boeing Aerospace Corporation

Tuesday, June 22, 2010
[...] the outage was due to an upgrade of the company’s Web site [...]
Regression Testing
Process and Issues
Regression Testing
Process and Issues

- Test suite T
  - Test-suite maintenance
  - Regression test selection
  - Test-suite prioritization
  - Prioritized Test suite T'

  - Test suite Tval
    - Regression test selection
    - Test-suite prioritization
    - Prioritized Test suite T'

  - Obsolete test cases
  - Redundant test cases

- Modified test suite
  - Test-case manipulation
  - Minimized test suite
    - Test-suite minimization
    - Test suite Taug
  - Prioritized Test suite T'

Tuesday, June 22, 2010
Regression Testing Process and Issues

Test suite T → Test-suit maintenance → Test suite Tval → Regression test selection → Test suite T' → Test-suite prioritization → Prioritized Test suite T'

Modified test suite → Test-case manipulation → Minimized test suite → Test-suite minimization → Test suite Taug

Obsolete test cases → Redundant test cases

Tuesday, June 22, 2010
Regression Testing
Process and Issues

Test suite T

- Test-suite maintenance
  - Obsolete test cases
- Test suite Tval
  - Regression test selection
  - Redundant test cases
  - Test-suite augmentation

- Test suite T'
  - Test-suite prioritization
  - Test suite Taug
    - Test-case manipulation
    - Minimized test suite
    - Test-suite minimization

Modified test suite

Prioritized Test suite T'

Tuesday, June 22, 2010
Regression Testing Process and Issues

Modified test suite

Test-suite maintenance

Test suite T

Test suite Tval

Regression test selection

Test suite T'

Test-suite prioritization

Prioritized Test suite T'

Test-case manipulation

Test-suite augmentation

Minimized test suite

Test-suite minimization

Test suite Taug

Obsolete test cases

Redundant test cases
Outline

- Introduction
- Regression test selection
- Test suite augmentation
- Test suite minimization
- Conclusion
Outline

• Introduction
• Regression test selection
• Test suite augmentation
• Test suite minimization
• Conclusion
Outline

- Introduction
- Regression test selection
- Test suite augmentation
- Test suite minimization
- Conclusion
Regression Test Selection

Test suite Tval

Regression test selection

Test suite T'

Time to rerun Tval

Analysis time

Time to rerun T'

Savings

Time

Tuesday, June 22, 2010
class A {
    void foo() {}
}
class B extends A {
}
class C extends B {
}
class D {
    void bar() {
        A ref=null;
        switch(somevar) {
            case '1': ref=new A(); break;
            case '2': ref=new B(); break;
            case '3': ref=new C(); break;
        }
        ref.foo();
    }
}
class E extends D {
}

class F {
    void bar(D d) {
    }
}
Motivating Example

class A {
    void foo() {...}
}
class B extends A {
}
class C extends B {}

class D {
    void bar() {
        A ref=null;
        switch(somevar) {
            case '1': ref=new A(); break;
            case '2': ref=new B(); break;
            case '3': ref=new C(); break;
        }
        ref.foo();
    }
}
class E extends D {}

class F {
    void bar(D d) {...}
}
Motivating Example

class A {
    void foo() {...}
}
class B extends A {
}
class C extends B {
}
class D {
    void bar() {
        A ref=null;
        switch(somevar) {
            case '1': ref=new A(); break;
            case '2': ref=new B(); break;
            case '3': ref=new C(); break;
        }
        ref.foo();
    }
}
class E extends D {
}
class F {
    void bar(D d) {...}
}

class A {
    void foo() {...}
}
class B extends A {
    void foo() {...}
}
class C extends B {
}
class D {
    void bar() {
        A ref=null;
        switch(somevar) {
            case '1': ref=new A(); break;
            case '2': ref=new B(); break;
            case '3': ref=new C(); break;
        }
        ref.foo();
    }
}
class E extends D {
}
class F {
    void bar(D d) {...}
}
Motivating Example

class A {
    void foo() {...} }
class B extends A {
}
class C extends B {}
class D {
    void bar() {
        A ref=null;
        switch(somevar) {
            case '1': ref=new A(); break;
            case '2': ref=new B(); break;
            case '3': ref=new C(); break;
        }
        ref.foo();
    }
}
class E extends D {}
class F {
    void bar(D d) {...} }

class A {
    void foo() {...} }
class B extends A {
    void foo() {...} }
class C extends B {
    void foo() {...} }
class D {
    void bar() {
        A ref=null;
        switch(somevar) {
            case '1': ref=new A(); break;
            case '2': ref=new B(); break;
            case '3': ref=new C(); break;
        }
        ref.foo();
    }
}
class E extends D {}
class F {
    void bar(D d) {...} }
Our Approach

• Handle Java features by suitably modeling them in the Java Interclass Graph (JIG)
• Use an algorithm that operates on the JIG to perform safe RTS
• Make some assumptions for safety
RTS Algorithm

1. Build JIG for P

2. Collect coverage data

3. Build G' and compare

4. Select affected tests

G

if()
e1

doA
e2
doB

G'

if()
e1

doA

e2
doC

tc1
tc2
tc3
edges

e1
X
e2
X X

test cases

tc1
tc2
tc3

edges

e1
X
e2
X X
Low-level, precise

Ideal solution: two-phase approach

- Class-Level analysis \( \mapsto \) subset of \( P \)
- Stmt-Level analysis on the subset \( \mapsto T \)

Low-level, precise Stmt-level Analysis

Program \( P \)

Test suite \( T_{val} \)

Program \( P' \)

Test suite \( T' \)

Tuesday, June 22, 2010
Low-level, precise

Several medium-sized subjects (up to 40KLOC)
Ideal solution: two-phase approach

Class-Level analysis → subset of P

Stmt-Level analysis on the subset → T

Low-level, precise

Program P → Stmt-level Analysis → Test suite Tval

Program P' → Test suite T'

Several medium-sized subjects (up to 40KLOC)

Time to rerun Tval

Analysis time

Time to rerun T'

Savings

Time
Low-level, precise

Ideal solution: two-phase approach

Class-Level analysis → subset of P

Stmt-Level analysis on the subset → T

erun

Program P

Program P’

Stmt-level Analysis

Test suite Tval

Test suite T’

JBoss – web application server, 1 million LOC
Low-level, precise

Program P

Program P'

Stmt-level Analysis

Test suite Tval

Test suite T'

Analysis time

Time to rerun T'

Time to rerun Tval

JBoss – web application server, 1 million LOC

Time

Tuesday, June 22, 2010
High-level, imprecise

- Ideal solution: two-phase approach
  - Class-Level analysis: subset of P
  - Stmt-Level analysis on the subset

Program P \rightarrow High-level Analysis \rightarrow Program P'

Test suite Tval \rightarrow High-level Analysis

Test suite T' \rightarrow Time to rerun T'

Analysis time \rightarrow Time to rerun T'

Time
High-level, imprecise

Related Work

- Efficient, less precise techniques
  - White and Leung [CSM92]
  - Chen, Rosenblum, and Vo [ICSE94]
  - Hsia et al. [SMRP97]
  - White and Abdullah [QW97]
  - Ren et al. [OOPSLA04]
  - ...

- Expensive, more precise techniques
  - Binkley [TSE97]
  - Rothermel and Harrold [TOSEM97]
  - Vokolos and Frankl [RQSSIS97]
  - Ball [ISSTA'98]
  - Rothermel, Harrold, and Dedhia [JSTVR00]
  - Harrold et al. [OOPSLA01]
  - Bible, Rothermel, and Rosenblum [TOSEM01]
  - ....
Our solution

Two-phase approach

1. Class-Level analysis $\rightarrow$ subset of P and P'
2. Stmt-Level analysis on the subset $\rightarrow$ T'

Tuesday, June 22, 2010
I. Class-level Analysis

P/P'

class A {
    void foo() {...}
}
class B extends A {
    void foo() {...}
}
class C extends B {}
class D {
    void bar() {
        A ref=null;
        switch(somevar) {
            case ‘1’: ref=new A(); break;
            case ‘2’: ref=new B(); break;
            case ‘3’: ref=new C(); break;
        }
        ref.foo();
    }
}
class E extends D {}
class F {
    void bar(D d) {...}
}
2. Stmt-level Analysis

Subset of P

```cpp
class A
class B {...}
class C
class D {
   void bar() {...; ref.foo(); ...}
}
```

Subset of P'

```cpp
class A
class B {... void foo() {...} ...}
class C
class D {
   void bar() {...; ref.foo(); ...}
}
```

G (excerpt)

```
... ref.foo()
   A.foo()
```

G' (excerpt)

```
... ref.foo()
   A.foo()
   B.foo()...
## 2. Stmt-level Analysis

### Subset of P
```
class A
class B {...}
class C
class D {
    void bar() {...; ref.foo(); ...}
}
```

### Subset of P’
```
class A
class B {... void foo() {...} ...
class C
class D {
    void bar() {...; ref.foo(); ...}
}
```

### G (excerpt)
```
ref.foo()
...  A
    A.foo()
    B
    C
...  B
    C
...  C
```

### G’ (excerpt)
```
Test cases to be rerun:
Test cases in Tval that execute the call node with ref’s dynamic type being B or C
```

Tuesday, June 22, 2010
Empirical Evaluation

- **Tool**: DejaVoo
- **Subjects**:

<table>
<thead>
<tr>
<th>Program</th>
<th>#versions</th>
<th>#classes</th>
<th>KLOC</th>
<th>#test cases</th>
<th>retest time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jaba</td>
<td>5</td>
<td>525</td>
<td>70</td>
<td>707</td>
<td>54 min</td>
</tr>
<tr>
<td>Daikon</td>
<td>5</td>
<td>824</td>
<td>167</td>
<td>200</td>
<td>74 min</td>
</tr>
<tr>
<td>Jboss</td>
<td>5</td>
<td>2,403</td>
<td>1,000</td>
<td>639</td>
<td>32 min</td>
</tr>
</tbody>
</table>

- **RQ**: What are the savings in testing time we can achieve using DejaVoo?

Tuesday, June 22, 2010
Retesting time (percentage)

<table>
<thead>
<tr>
<th>Retesting Time</th>
<th>RerunAll</th>
<th>DejaVOO</th>
</tr>
</thead>
<tbody>
<tr>
<td>v2</td>
<td>110%</td>
<td>0%</td>
</tr>
<tr>
<td>v3</td>
<td>110%</td>
<td>0%</td>
</tr>
<tr>
<td>v4</td>
<td>83%</td>
<td>0%</td>
</tr>
<tr>
<td>v5</td>
<td>55%</td>
<td>0%</td>
</tr>
<tr>
<td>Jaba</td>
<td>55%</td>
<td>0%</td>
</tr>
<tr>
<td>Daikon</td>
<td>55%</td>
<td>0%</td>
</tr>
<tr>
<td>Jboss</td>
<td>55%</td>
<td>0%</td>
</tr>
</tbody>
</table>

Tuesday, June 22, 2010
Results

Savings in regression testing time: DejaVOO vs. RerunAll
- Jaba: 19%
- Daikon: 36%
- Jboss: 63%
Regression Test Selection

Summary

• DejaVOO
  • Based on the Interclass Relation Graph and Java Interclass Graph
  • First phase identifies affected classes
  • Second phase performs low-level analysis

• Benefits of our technique
  • Handles Java features
  • Handles subsystems without analyzing external classes
  • Safe (under some assumptions)
Outline

• Introduction
• Regression test selection
  • Test suite augmentation
• Test suite minimization
• Conclusion
Test Suite Augmentation

- **Test suite T**
  - Test-suite maintenance
  - Obsolete test cases
  - Modified test suite
  - Test-case manipulation

- **Test suite Tval**
  - Regression test selection
  - Redundant test cases
  - Minimized test suite
  - Test-suite minimization

- **Test suite T'**
  - Test-suite prioritization
  - Prioritized Test suite T'
  - Test-suite augmentation

Tuesday, June 22, 2010
Test Suite Augmentation
Traditional regression testing
Regression testing

Traditional regression testing

Class BankAccount

```java
class BankAccount
{
    double balance;

    public boolean deposit(double amount)
    {
        if (amount > 0.00)
        {
            balance = balance + amount;
            return true;
        }
        else
        {
            print("negative amount");
            return false;
        }
    }

    public boolean withdraw(double amount)
    {
        if (amount <= 0)
        {
            print("account overdraft");
            return false;
        }
        if (balance < 0)
        {
            print("negative amount");
            return false;
        }
        else
        {
            balance = balance - amount;
            return true;
        }
    }

    public void print(String message)
    {
        System.out.println(message);
    }
}
```
class BankAccount {

double balance;

bool deposit(double amount) {
    if (amount > 0.00) {
        balance = balance + amount;
        return true;
    } else {
        print("negative amount");
        return false;
    }
}

bool withdraw(double amount) {
    if (amount <= 0) {
        print("negative amount");
        return false;
    }
    if (balance < 0)
        print("account overdraft");
    return true;
}

return true;
}

}
Where is the fault?
class BankAccount {
    double balance;
    bool isOverdraft;

    bool deposit(double amount) {
        if (amount > 0.00) {
            balance = balance + amount;
            return true;
        } else {
            print("negative amount");
            return false;
        }
    }

    bool withdraw(double amount) {
        if (amount <= 0) {
            print("negative amount");
            return false;
        }
        if (isOverdraft) {
            print("account overdraft");
            return false;
        }
        balance = balance - amount;
        if (balance < 0)
            isOverdraft = true;
        return true;
    }
}
class BankAccount {
    double balance;
    bool isOverdraft;

    bool deposit(double amount) {
        if (amount > 0.00) {
            balance = balance + amount;
            return true;
        } else {
            print("negative amount");
            return false;
        }
    }

    bool withdraw(double amount) {
        if (amount <= 0) {
            print("negative amount");
            return false;
        }
        if (isOverdraft) {
            print("account overdraft");
            return false;
        }
        balance = balance - amount;
        if (balance < 0) {
            isOverdraft = true;
        }
        return true;
    }
}

Class BankAccountTest {
}
class BankAccount {
    double balance;
    bool isOverdraft;

    bool deposit(double amount) {
        if (amount > 0.00) {
            balance = balance + amount;
            return true;
        } else {
            print("negative amount");
            return false;
        }
    }

    bool withdraw(double amount) {
        if (amount <= 0) {
            print("negative amount");
            return false;
        }
        if (isOverdraft) {
            print("account overdraft");
            return false;
        }
        balance = balance - amount;
        if (balance < 0)
            isOverdraft = true;
        return true;
    }
}

Class BankAccountTest {
    ...
    void test1() {
        BankAccount a=new BankAccount();
        bool result = a.deposit(-1.00);
        assertEquals(result, false);
    }
}

Tuesday, June 22, 2010
class BankAccount {
    double balance;
    bool isOverdraft;

    bool deposit(double amount) {
        if (amount > 0.00) {
            balance = balance + amount;
            return true;
        } else {
            print("negative amount");
            return false;
        }
    }

    bool withdraw(double amount) {
        if (amount <= 0) {
            print("negative amount");
            return false;
        }
        if (isOverdraft) {
            print("account overdraft");
            return false;
        }
        balance = balance - amount;
        if (balance < 0)
            isOverdraft = true;
        return true;
    }
}

class BankAccountTest {
    ...
    void test1() {
        BankAccount a = new BankAccount();
        bool result = a.deposit(-1.00);
        assertEquals(result, false);
    }
}

Tuesday, June 22, 2010
```java
import java.util.

class BankAccount {
    double balance;
    boolean isOverdraft;

    boolean deposit(double amount) {
        if (amount > 0.00) {
            balance = balance + amount;
            return true;
        } else {
            println("negative amount");
            return false;
        }
    }

    boolean withdraw(double amount) {
        if (amount <= 0) {
            println("negative amount");
            return false;
        }
        if (isOverdraft) {
            println("account overdraft");
            return false;
        }
        balance = balance - amount;
        if (balance < 0)
            isOverdraft = true;
        return true;
    }
}
```
class BankAccount {
    double balance;
    bool isOverdraft;

    bool deposit(double amount) {
        if (amount > 0.00) {
            balance = balance + amount;
            return true;
        } else {
            print("negative amount");
            return false;
        }
    }

    bool withdraw(double amount) {
        if (amount <= 0) {
            print("negative amount");
            return false;
        } else {
            print("account overdraft");
            return false;
        }
        balance = balance - amount;
        if (balance < 0) {
            isOverdraft = true;
        }
        return true;
    }
}

Class BankAccountTest {
...}

void test1() {
    BankAccount a=new BankAccount();
    bool result = a.deposit(-1.00);
    assertEquals(result, false);
}

void test2() {
    BankAccount a=new BankAccount();
    bool result = a.withdraw(-1.00);
    assertEquals(result, false);
}

Tuesday, June 22, 2010
class BankAccount {
  double balance;
  bool isOverdraft;

  bool deposit(double amount) {
    if (amount > 0.00) {
      balance = balance + amount;
      return true;
    } else {
      print("negative amount");
      return false;
    }
  }

  bool withdraw(double amount) {
    if (amount <= 0) {
      print("negative amount");
      return false;
    } else {
      if (isOverdraft) {
        print("account overdraft");
        return false;
      }
      balance = balance - amount;
      if (balance < 0)
        isOverdraft = true;
      return true;
    }
  }
}

Class BankAccountTest {
  ...

  void test1() {
    BankAccount a = new BankAccount();
    bool result = a.deposit(-1.00);
    assertEquals(result, false);
  }

  void test2() {
    BankAccount a = new BankAccount();
    bool result = a.withdraw(-1.00);
    assertEquals(result, false);
  }
}
class BankAccount {
    double balance;
    bool isOverdraft;

    bool deposit(double amount) {
        if (amount > 0.00) {
            balance = balance + amount;
            return true;
        } else {
            print("negative amount");
            return false;
        }
    }

    bool withdraw(double amount) {
        if (amount <= 0) {
            print("negative amount");
            return false;
        } else {
            if (isOverdraft) {
                print("account overdraft");
                return false;
            }
            balance = balance - amount;
            if (balance < 0)
                isOverdraft = true;
            return true;
        }
    }
}

class BankAccountTest {
    ...

    void test1() {
        BankAccount a=new BankAccount();
        bool result = a.deposit(-1.00);
        assertEquals(result, false);
    }

    void test2() {
        BankAccount a=new BankAccount();
        bool result = a.withdraw(-1.00);
        assertEquals(result, false);
    }

    ...
}
class BankAccount
{
    double balance;
    bool isOverdraft;

    bool deposit(double amount) {
        if (amount > 0.00) {
            balance = balance + amount;
            return true;
        } else {
            print("negative amount");
            return false;
        }
    }

    bool withdraw(double amount) {
        if (amount <= 0) {
            print("negative amount");
            return false;
        } else {
            if (isOverdraft) {
                print("account overdraft");
                return false;
            }
            balance = balance - amount;
            if (balance < 0)
                isOverdraft = true;
            return true;
        }
    }
}

Class BankAccountTest {
...

void test1() {
    BankAccount a=new BankAccount();
    bool result = a.deposit(-1.00);
    assertEquals(result, false);
}

void test2() {
    BankAccount a=new BankAccount();
    bool result = a.withdraw(-1.00);
    assertEquals(result, false);
}

void test3() {
    BankAccount a=new BankAccount();
    a.deposit(100.00);
    bool result = a.withdraw(50.00);
    assertEquals(result, true);

    if (isOverdraft) {
        print("account overdraft");
        return false;
    }
    balance = balance - amount;
    if (balance < 0)
        isOverdraft = true;
    return true;
}
}
class BankAccount {
    double balance;
    bool isOverdraft;

    bool deposit(double amount) {
        if (amount > 0.00) {
            balance = balance + amount;
            return true;
        } else {
            print("negative amount");
            return false;
        }
    }

    bool withdraw(double amount) {
        if (amount <= 0) {
            print("negative amount");
            return false;
        }
        if (isOverdraft) {
            print("account overdraft");
            return false;
        }
        balance = balance - amount;
        if (balance < 0)
            isOverdraft = true;
        return true;
    }
}


Class BankAccountTest {
...
void test1() {
    BankAccount a=new BankAccount();
    bool result = a.deposit(-1.00);
    assertEquals(result, false);
}
void test2() {
    BankAccount a=new BankAccount();
    bool result = a.withdraw(-1.00);
    assertEquals(result, false);
}
void test3() {
    BankAccount a=new BankAccount();
    a.deposit(100.00);
    bool result = a.withdraw(50.00);
    assertEquals(result, true);
}
class BankAccount {

double balance;
bool isOverdraft;

bool deposit(double amount) {
    if (amount > 0.00) {
        balance = balance + amount;
        return true;
    } else {
        print("negative amount");
        return false;
    }
}

bool withdraw(double amount) {
    if (amount <= 0) {
        print("negative amount");
        return false;
    } else {
        if (isOverdraft) {
            print("account overdraft");
            return false;
        }
        balance = balance - amount;
        if (balance < 0)
            isOverdraft = true;
        return true;
    }
}

}

Class BankAccountTest {
...

void test1() {
    BankAccount a = new BankAccount();
    bool result = a.deposit(-1.00);
    assertEquals(result, false);
}

void test2() {
    BankAccount a = new BankAccount();
    bool result = a.withdraw(-1.00);
    assertEquals(result, false);
}

void test3() {
    BankAccount a = new BankAccount();
    a.deposit(100.00);
    bool result = a.withdraw(50.00);
    assertEquals(result, true);
}


Tuesday, June 22, 2010
```java
class BankAccount {
    double balance;
    boolean isOverdraft;

    boolean deposit(double amount) {
        if (amount > 0.00) {
            balance = balance + amount;
            return true;
        } else {
            print("negative amount");
            return false;
        }
    }

    boolean withdraw(double amount) {
        if (amount <= 0) {
            print("negative amount");
            return false;
        }
        if (isOverdraft) {
            print("account overdraft");
            return false;
        }
        balance = balance - amount;
        if (balance < 0)
            isOverdraft = true;
        return true;
    }
}
```

```java
Class BankAccountTest {
...

void test1() {
    BankAccount a = new BankAccount();
    boolean result = a.deposit(-1.00);
    assertEquals(result, false);
}

void test2() {
    BankAccount a = new BankAccount();
    boolean result = a.withdraw(-1.00);
    assertEquals(result, false);
}

void test3() {
    BankAccount a = new BankAccount();
    a.deposit(100.00);
    boolean result = a.withdraw(50.00);
    assertEquals(result, true);
}

void test4() {
    BankAccount a = new BankAccount();
    a.deposit(100.00);
    a.withdraw(200.00);
    boolean result = a.withdraw(50.00);
    assertEquals(result, false);
}
```

Tuesday, June 22, 2010
class BankAccount {

double balance;
bool isOverdraft;

bool deposit(double amount) {
    if (amount > 0.00) {
        balance = balance + amount;
        return true;
    } else {
        print("negative amount");
        return false;
    }
}

bool withdraw(double amount) {
    if (amount <= 0) { // Error: if (amount <= 0)
        print("negative amount");
        return false;
    }
    if (isOverdraft) {
        print("account overdraft");
        return false;
    }
    balance = balance - amount;
    if (balance < 0)
        isOverdraft = true;
    return true;
}
}

Class BankAccountTest {
...
void test1() {
    BankAccount a=new BankAccount();
    bool result = a.deposit(-1.00);
    assertEquals(result, false);
}
void test2() {
    BankAccount a=new BankAccount();
    bool result = a.withdraw(-1.00);
    assertEquals(result, false);
}
void test3() {
    BankAccount a=new BankAccount();
    a.deposit(100.00);
    bool result = a.withdraw(50.00);
    assertEquals(result, true);
}
void test4() {
    BankAccount a=new BankAccount();
    a.deposit(100.00);
    a.withdraw(200.00);
    bool result = a.withdraw(50.00);
    assertEquals(result, false);
}
}

Tuesday, June 22, 2010
class BankAccount {
  double balance;
  bool isOverdraft;

  bool deposit(double amount) {
    if (amount > 0.00) {
      balance = balance + amount;
      return true;
    } else {
      print("negative amount");
      return false;
    }
  }

  bool withdraw(double amount) {
    if (amount <= 0) {
      print("negative amount");
      return false;
    } else if (isOverdraft) {
      print("account overdraft");
      return false;
    }
    balance = balance - amount;
    if (balance < 0) {
      isOverdraft = true;
    }
    return true;
  }
}

class BankAccountTest {
  ...

  void test1() {
    BankAccount a = new BankAccount();
    bool result = a.deposit(-1.00);
    assertEquals(result, false);
  }

  void test2() {
    BankAccount a = new BankAccount();
    bool result = a.withdraw(-1.00);
    assertEquals(result, false);
  }

  void test3() {
    BankAccount a = new BankAccount();
    a.deposit(100.00);
    bool result = a.withdraw(50.00);
    assertEquals(result, true);
  }

  void test4() {
    BankAccount a = new BankAccount();
    a.deposit(100.00);
    a.withdraw(200.00);
    bool result = a.withdraw(50.00);
    assertEquals(result, false);
    result = a.deposit(200.00);
    assertEquals(result, true);
  }
  ...

  ...
}
Class BankAccountTest {
... 
void test1() {
    BankAccount a=new BankAccount();
    bool result = a.deposit(-1.00);
    assertEquals(result, false);
}
void test2() {
    BankAccount a=new BankAccount();
    bool result = a.withdraw(-1.00);
    assertEquals(result, false);
}
void test3() {
    BankAccount a=new BankAccount();
    a.deposit(100.00);
    bool result = a.withdraw(50.00);
    assertEquals(result, true);
}
void test4() {
    BankAccount a=new BankAccount();
    a.deposit(100.00);
    a.withdraw(200.00);
    bool result = a.withdraw(50.00);
    assertEquals(result, false);
    result = a.deposit(200.00);
    assertEquals(result, true);
    ...
}

class BankAccount {
    double balance;
    bool isOverdraft;

    bool deposit(double amount) {
        if (amount > 0.00) {
            balance = balance + amount;
            return true;
        } else {
            print("negative amount");
            return false;
        }
    }

    bool withdraw(double amount) {
        if (amount <= 0) {
            if (isOverdraft) {
                print("account overdraft");
                return false;
            }
            balance = balance - amount;
            if (balance < 0)
                isOverdraft = true;
            return true;
        }
    }
}
class BankAccount {

double balance;
bool isOverdraft;

bool deposit(double amount) {
    if (amount > 0.00) {
        balance = balance + amount;
        return true;
    } else {
        print("negative amount");
        return false;
    }
}

bool withdraw(double amount) {
    if (amount <= 0) {
        print("negative amount");
        return false;
    } else {
        print("account overdraft");
        return false;
    }
    balance = balance - amount;
    if (balance < 0)
        isOverdraft = true;
    return true;
}
}

Class BankAccountTest {

void test1() {
    BankAccount a = new BankAccount();
    bool result = a.deposit(-1.00);
    assertEquals(result, false);
}

void test2() {
    BankAccount a = new BankAccount();
    bool result = a.withdraw(-1.00);
    assertEquals(result, false);
}

void test3() {
    BankAccount a = new BankAccount();
    a.deposit(100.00);
    bool result = a.withdraw(50.00);
    assertEquals(result, true);
}

void test4() {
    BankAccount a = new BankAccount();
    a.deposit(100.00);
    a.withdraw(200.00);
    bool result = a.withdraw(50.00);
    assertEquals(result, false);
    result = a.deposit(200.00);
    assertEquals(result, true);
}
...
class BankAccount {
    double balance;
    bool isOverdraft;

    bool deposit(double amount) {
        if (amount > 0.00) {
            balance = balance + amount;
            return true;
        } else {
            print("negative amount");
            return false;
        }
    }

    bool withdraw(double amount) {
        if (amount <= 0) {
            print("negative amount");
            return false;
        }
        if (isOverdraft) {
            print("account overdraft");
            return false;
        }
        balance = balance - amount;
        if (balance < 0)
            isOverdraft = true;
        return true;
    }
}

...
class BankAccount {

double balance;
bool isOverdraft;

bool deposit(double amount) {
if (amount > 0.00) {
balance = balance + amount;
return true;
} else {
print("negative amount");
return false;
}
}

bool withdraw(double amount) {
if (amount <= 0) {
print("negative amount");
return false;
}
if (isOverdraft) {
print("account overdraft");
return false;
}
balance = balance - amount;
if (balance < 0)
isOverdraft = true;
return true;
}

...

void testBehavioralDifference() {
BankAccount a = new BankAccount();
a.deposit(10.00);
a.withdraw(20.00);
a.deposit(50.00);
bool result = a.withdraw(20.00);
assertEquals(result, true);
}

...
Such a test may not be in T
• 100% stmt coverage without it
• Specific sequence of calls/params
• Or its oracle may be inadequate
Traditional regression testing

Existing test suites typically target a small subset of the program behavior

- Tests focus on core functionality
- Oracles often approximated

Tuesday, June 22, 2010
Traditional regression testing

Program P

Test suite T

Test runner & Oracle checker

Regression errors

Program P'

Program P

Program P'

Test suite T

BERT

Tuesday, June 22, 2010
Program P

Program P'

Test suite T
Phase I:
Generation of test cases for changed code
Phase I: Generation of test cases for changed code

Program P

Program P'

Test suite T

Change analyzer

Code changes C
**Phase I:**
Generation of test cases for changed code

**Change analyzer**

- Given two versions, produces a list of changed classes
- Can use any differencing tool
- Currently: Eclipse’s change information

Tuesday, June 22, 2010
Phase I: Generation of test cases for changed code

Program P

Program P'

Tests for C TC

Test suite T

Change analyzer

Code changes C

Test case generator

Tuesday, June 22, 2010
Phase I: Generation of test cases for changed code

Test case generator

- Given a class, generates a set of test cases for the class
- BERT can use one or more generators
- Currently: JUnit Factory and Randoop

Tuesday, June 22, 2010
Phase I: Generation of test cases for changed code

Test case generator

- Given a class, generates a set of test cases for the class
- BERT can use one or more generators
- Currently: JUnit Factory and Randoop
BERT

Phase II: Behavioral comparison

Program P

Program P'

Tests for C TC

Test suite T

Change analyzer

Code changes C

Test case generator

Tuesday, June 22, 2010
Phase II: Behavioral comparison
Test runner & Behavioral comparator

- ∀ c and t for c, runs t on old and new versions of c, ∀ call within t to m in c, logs
Test runner & Behavioral comparator

∀ c and t for c, runs t on old and new versions of c, ∀ call within t to m in c, logs

• **State** (∀ field):
  
  `<seq_id, m_sig, name, value>`

**Phase II**: Behavioral comparison
Test runner & Behavioral comparator

- ∀ c and t for c, runs t on old and new versions of c, ∀ call within t to m in c, logs

  - **State** (∀ field):
    - <seq_id, m_sig, name, value>
  
  - **Return values**:
    - <seq_id, m_sig, value>

*Phase II*: Behavioral comparison
Test runner & Behavioral comparator

- ∀ c and t for c, runs t on old and new versions of c, ∀ call within t to m in c, logs

  - **State** (∀ field):
    <seq_id, m_sig, name, value>

  - **Return values**:
    <seq_id, m_sig, value>

  - **Outputs**:
    <seq_id, m_sig, dest, data>

**Phase II**: Behavioral comparison
Test runner & Behavioral comparator

- ∀ c and t for c, runs t on old and new versions of c, ∀ call within t to m in c, logs
  - **State** (∀ field):
    <seq_id, m_sig, name, value>
  - **Return values**:
    <seq_id, m_sig, value>
  - **Outputs**:
    <seq_id, m_sig, dest, data>
  - **Distance**

Phase II: Behavioral comparison
Test runner & Behavioral comparator

\( \forall c \text{ and } t \text{ for } c, \text{ runs } t \text{ on old and new versions of } c, \forall \text{ call within } t \text{ to } m \text{ in } c, \text{ logs} \)

- **State** (\( \forall \) field):
  - \(<\text{seq}_\text{id}, \text{m}_\text{sig}, \text{name}, \text{value}>\>

- **Return values**:
  - \(<\text{seq}_\text{id}, \text{m}_\text{sig}, \text{value}>\>

- **Outputs**:
  - \(<\text{seq}_\text{id}, \text{m}_\text{sig}, \text{dest}, \text{data}>\>

- **Distance**
Test runner & Behavioral comparator

- ∀ \( c \) and \( t \) for \( c \), runs \( t \) on old and new versions of \( c \), ∀ call within \( t \) to \( m \) in \( c \), logs

- **State** (∀ field):
  \(<\text{seq}_\text{id}, \text{m}_\text{sig}, \text{name}, \text{value}>\>

- **Return values**:
  \(<\text{seq}_\text{id}, \text{m}_\text{sig}, \text{value}>\>

- **Outputs**:
  \(<\text{seq}_\text{id}, \text{m}_\text{sig}, \text{dest}, \text{data}>\>

- **Distance**
Test runner & Behavioral comparator

• \( \forall c \) and \( t \) for \( c \), runs \( t \) on old and new versions of \( c \), \( \forall \) call within \( t \) to \( m \) in \( c \), logs

• **State** (\( \forall \) field):
  \(<\text{seq}_{-}\text{id}}, \text{m}_{-}\text{sig}, \text{name}, \text{value}>\n
• **Return values**:
  \(<\text{seq}_{-}\text{id}}, \text{m}_{-}\text{sig}, \text{value}>\n
• **Outputs**:
  \(<\text{seq}_{-}\text{id}}, \text{m}_{-}\text{sig}, \text{dest}, \text{data}>\n
• **Distance**

---

**Phase II**

- **Dynamic call graph**

---

**Class C**

**Test case t**

- **Changed method**
- **Method showing behavioral differences**

---

Tuesday, June 22, 2010
Test runner & Behavioral comparator

- ∀ c and t for c, runs t on old and new versions of c, ∀ call within t to m in c, logs
  - **State** (∀ field):
    <seq_id, m_sig, name, value>
  - **Return values**:
    <seq_id, m_sig, value>
  - **Outputs**:
    <seq_id, m_sig, dest, data>
  - **Distance**

Phase II:

Class C

Test case t

Dynamic call graph

- Changed method
- Method showing behavioral differences

Tuesday, June 22, 2010
Test runner & Behavioral comparator

- ∀ c and t for c, runs t on old and new versions of c, ∀ call within t to m in c, logs

- **State** (∀ field):
  - <seq_id, m_sig, name, value>

- **Return values**:
  - <seq_id, m_sig, value>

- **Outputs**:
  - <seq_id, m_sig, dest, data>

- **Distance**

Phase II:

Class C

Test case t

Dynamic call graph

- Changed method
- Method showing behavioral differences
**Test runner & Behavioral comparator**

- ∀ \( c \) and \( t \) for \( c \), runs \( t \) on old and new versions of \( c \), ∀ call within \( t \) to \( m \) in \( c \), logs

- **State** (∀ field):
  - \(<\text{seq}\_\text{id}, \text{m}\_\text{sig}, \text{name}, \text{value}>\>

- **Return values**:
  - \(<\text{seq}\_\text{id}, \text{m}\_\text{sig}, \text{value}>\>

- **Outputs**:
  - \(<\text{seq}\_\text{id}, \text{m}\_\text{sig}, \text{dest}, \text{data}>\>

- **Distance**

**Phase II**:

- Changed method
- Method showing behavioral differences

Tuesday, June 22, 2010
Test runner & Behavioral comparator

- \(\forall c\) and \(t\) for \(c\), runs \(t\) on old and new versions of \(c\), \(\forall\) call within \(t\) to \(m\) in \(c\), logs

  - **State** (\(\forall\) field):
    \(<\text{seq}_\text{id}, \text{m}_\text{sig}, \text{name}, \text{value}>\>
  
  - **Return values**:
    \(<\text{seq}_\text{id}, \text{m}_\text{sig}, \text{value}>\>
  
  - **Outputs**:
    \(<\text{seq}_\text{id}, \text{m}_\text{sig}, \text{dest}, \text{data}>\>

  - **Distance**

- Compares and stores differences and relevant context

Phase II: Behavioral comparison

Tuesday, June 22, 2010
Phase III: Differential behavior analysis and reporting
**Phase III:**
Differential behavior analysis and reporting
Behavioral differences analyzer

Phase III: Differential behavior analysis and reporting

Tuesday, June 22, 2010
Behavioral differences analyzer

- Simplifies and refines raw data through abstraction and redundancy elimination

Phases:

**Phase III:**
Differential behavior analysis and reporting
Behavioral differences analyzer

- Simplifies and refines raw data through abstraction and redundancy elimination
- Reports behavioral differences between $c_{v0}$ and $c_{v1}$ and test cases that reveal them
  - fields with $\neq$ values
  - methods returning $\neq$ values
  - differences in output

Phase III: Differential behavior analysis and reporting
Behavioral differences analyzer

- Simplifies and refines raw data through abstraction and redundancy elimination
- Reports behavioral differences between $c_v_0$ and $c_v_1$ and test cases that reveal them
  - fields with $\neq$ values
  - methods returning $\neq$ values
  - differences in output
- Ranks reports based on distance

Phase III: Differential behavior analysis and reporting
Evaluation

- **RQ**: Can BERT reveal regression faults automatically w/o generating too many false positives?
- **Prototype** (partial) implementation
  - Standalone
  - Eclipse plug-in
- **Two studies**
  - Proof of concept
  - Preliminary evaluation on a real program
Study 1: Proof of Concept

- Applied BERT to BankAccount example
- Fed BankAccount to BERT
- Generated 2,569 test inputs (< 1 sec to execute)
- 60% of the inputs (1,557) showed a behavioral difference that revealed the regression error
  - withdraw returned different values
  - withdraw resulted in a different state
- No false positives generated
Study 1: Proof of Concept

- Applied BERT to BankAccount example
- Fed BankAccount to BERT
- Generated 2,569 test inputs (<1 sec to execute)
- 60% of the inputs revealed a behavioral difference, indicating regression error
- withdraw returned different values
- withdraw resulted in a different state
- No false positives generated
Study 2: Real Program

- Subject program: JodaTime
- Java library (~60KLOC) that extends Java’s JDK
- SVN on sourceforge
- Versions: 54 pairs of versions from SVN
  - Start from a “stable” point
  - Select first 60 versions
  - Eliminate all versions that include interface changes
- Run BERT on all 54 pairs ➡ identified 36 behavioral differences
  - No differences: 21 pairs
  - One difference: 30 pairs
  - Two differences: 3 pairs
Study 2: Analysis

- Manual check of the reports is in most cases not feasible (without involving the developers)

- Two subsets:
  - Study of false positives: 21 versions that showed no behavioral differences
  - Study of effectiveness: Highest ranked reports based on distance
    - 22 reports with distance 0
    - 10 reports with distance 1
    - 4 reports with distance > 1
Study 2: Results

- 21 versions that showed no behavioral differences
- 6 unknowns/uncovered
- 15 of them are refactorings
  ➡ No false positives
- 4 reports with distance > 1
  - 2 unknowns (ranked #1 and #4)
  - 1 sure true positive (ranked #2)
  - 1 sure false positive (ranked #3)
Study 2: Results

21 versions that showed no behavioral differences
6 unknowns/uncovered
15 of them are refactorings
➡ No false positives
4 reports with distance > 1
2 unknowns (ranked #1 and #4)
1 sure true positive (ranked #2)
1 sure false positive (ranked #3)

---

```java
//r916:
class BaseGJChronology {
    private transient YearInfo[] iYearInfoCache;
    private transient int iYearInfoCacheMask;
    ...
}
```

```java
//r917:
class BaseGJChronology {
    private static final int CACHE_SIZE = 1;
    private static final int CACHE_MASK = CACHE_SIZE - 1;
    private final YearInfo[] iYearInfoCache =
        new YearInfo[CACHE_SIZE];
    ...
}
```
Study 2: Results

// Study 2: Results

// r916:
class BaseGJChronology {
    private transient YearInfo[] iYearInfoCache;
    private transient int iYearInfoCacheMask;
    ...

// r917:
class BaseGJChronology {
    private static final int CACHE_SIZE = 1;
    private static final int CACHE_MASK = CACHE_SIZE - 1;
    private final YearInfo[] iYearInfoCache =
        new YearInfo[CACHE_SIZE];
    ...

Study 2: Results

Study case: class BaseGJChronology

//r916:
    class BaseGJChronology {
        private transient YearInfo[] iYearInfoCache;
        private transient int iYearInfoCacheMask;
        ...

    }

//r917:
    class BaseGJChronology {
        private static final int CACHE_SIZE = 1;
        private static final int CACHE_MASK = CACHE_SIZE - 1;
        private final YearInfo[] iYearInfoCache =
            new YearInfo[CACHE_SIZE];
        ...

    }

- 21 versions that showed no behavioral differences
- 6 unknowns/uncovered
- 15 of them are refactorings
- No false positives
- 4 reports with distance > 1
- 2 unknowns (ranked #1 and #4)
- 1 sure true positive (ranked #2)
- 1 sure false positive (ranked #3)
Study 2: Results

//r916:
class BaseGJChronology {
    private transient YearInfo[] iYearInfoCache;
    private transient int iYearInfoCacheMask;
    ...

//r917:
class BaseGJChronology {
    private static final int CACHE_SIZE = 1;
    private static final int CACHE_MASK = CACHE_SIZE - 1;
    private final YearInfo[] iYearInfoCache =
        new YearInfo[CACHE_SIZE];
    ...

Fixed three days later
Study 2: Results

- 21 versions that showed no behavioral differences
- 6 unknowns/uncovered
- 15 of them are refactorings
  - No false positives
- 4 reports with distance > 1
  - 2 unknowns (ranked #1 and #4)
  - 1 sure true positive (ranked #2)
  - 1 sure false positive (ranked #3)
Phase I: Generation of test cases for changed code

- Change analyzer
- Code changes C
- Program P
- Program P'
- Tests for C TC

Phase II: Behavioral comparison

- Test suite T
- Test runner & Behavioral comparator
- Raw behavioral differences

Phase III: Differential behavior analysis and reporting

- Behavioral differences
- Behavioral differences analyzer
Focus on a small code fraction ➞ **thorough**

Analyze differential behavior ➞ **no oracles**

**BERT**

- Change analyzer
- Test case generator
- Program P
- Program P'
- Tests for C TC
- Test runner & Behavioral comparator
- Test suite T
- Raw behavioral differences
- Behavioral differences analyzer

Tuesday, June 22, 2010
BERT

Encouraging **initial** results

- Identified real regression errors
- No behavioral differences reported for refactorings

Tuesday, June 22, 2010
Future work

• Tool release
• More extensive studies
  • User studies
  • Studies of false positives
• Reducing false positives
• Leveraging change analysis
• Using automated debugging
• Change-based test case generation
Outline

- Introduction
- Regression test selection
- Test suite augmentation
- Test suite minimization
- Conclusion
Test Suite Minimization

Test suite $T$ → Test-suite maintenance → Test suite $T_{val}$ → Regression test selection → Test suite $T'$ → Test-suite prioritization → Prioritized Test suite $T'$

Test case manipulation → Modified test suite

Test-suite augmentation → Test suite $T_{aug}$ → Test-suite minimization → Minimized test suite

Obsolete test cases → Redundant test cases
Test Suite Minimization

- Test suite Taug
- Minimized test suite
- Redundant test cases
  - Test-suite minimization

Tuesday, June 22, 2010
Motivating Scenario

Regression test selection

Test suite $T'$

Test-suite augmentation

Test suite $T_{aug}$

Test suite $T$

Program $P_0$

Program $P_1$
Motivating Scenario

Test suite $T$

Regression test selection

Test suite $T'$

Test-suite augmentation

Test suite $T_{aug}$

Program $P_0$

Program $P_n$
Test Suite Minimization

Test suite Tau

Criteria:
- coverage
- fault-detection ability
- time
- cost
- ...

Test-suite minimization

Minimized test suite

Redundant test cases
A Simple Example

Minimize test suite while maintaining the same level of coverage
A More Realistic Example

Relevant parameters:
1. Test suite to minimize: $T = \{t_1, t_2, t_3, t_4\}$
2. Requirements to cover: $R = \{\text{stmt1}, \text{stmt2}, \text{stmt3}\}$
3. Test-related data: cost and fault-detection data

<table>
<thead>
<tr>
<th>stmt1</th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
</tr>
</thead>
<tbody>
<tr>
<td>stmt2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stmt3</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Criteria of interest:
C1 – maintain coverage
C2 – minimize time to run
C3 – minimize setup effort
C4 – maximize fault detection
State of the Art

Several approaches in the literature (e.g., [HGS93], [H99], [MB03], [BMK04], [TG05])

Two main limitations:

- **Single criterion**
  (typically, coverage)

- **Approximated**
  (problem is NP-complete)

Only exception is [BMK04]: two criteria, but still limited in terms of expressiveness
Our Contribution

MINTS – novel technique (and freely-available tool) for test-suite minimization that:

- Lets testers specify a wide range of multi-criteria test-suite minimization problems
- Automatically encodes problems in binary ILP form
- Leverages different ILP solvers to find optimal solutions in a “reasonable” time
Overview of MINTS

Test suite

Test-related data
- Coverage data
- Cost data
- Fault detection data

Minimization criteria
- Criterion #1
- Criterion #2
- Criterion #n
- Minimization policy

MINTS tool

Minimized Test suite

Solver 1

Solver n

Solution (or timeout)

Minimization problem (suitably encoded)

Testing team
RQ1: How often can mints find an optimal solution “quickly”?

Subjects:

<table>
<thead>
<tr>
<th>Subject</th>
<th>LOC</th>
<th>COV</th>
<th>#Test Cases</th>
<th>#Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcas</td>
<td>173</td>
<td>72</td>
<td>1608</td>
<td>5</td>
</tr>
<tr>
<td>schedule2</td>
<td>307</td>
<td>146</td>
<td>2700</td>
<td>5</td>
</tr>
<tr>
<td>tot_info</td>
<td>406</td>
<td>136</td>
<td>1052</td>
<td>5</td>
</tr>
<tr>
<td>schedule</td>
<td>412</td>
<td>166</td>
<td>2650</td>
<td>5</td>
</tr>
<tr>
<td>replace</td>
<td>562</td>
<td>263</td>
<td>5542</td>
<td>5</td>
</tr>
<tr>
<td>print_tokens</td>
<td>563</td>
<td>194</td>
<td>4130</td>
<td>5</td>
</tr>
<tr>
<td>print_tokens2</td>
<td>570</td>
<td>197</td>
<td>4115</td>
<td>5</td>
</tr>
<tr>
<td>flex</td>
<td>12,421</td>
<td>567</td>
<td>548</td>
<td>5</td>
</tr>
<tr>
<td>LogicBlox</td>
<td>570,595</td>
<td>29204</td>
<td>393</td>
<td>5</td>
</tr>
<tr>
<td>Eclipse</td>
<td>1,892,226</td>
<td>35903</td>
<td>3621</td>
<td>5</td>
</tr>
</tbody>
</table>

Solvers:
Four SAT-based pseudo-Boolean and two pure ILP solvers
RQ1: How often can MINTS find an optimal solution quickly? (setup)

Test-related data
- Code coverage (gcov, cobertura)
- Running time (UNIX’s time utility)
- Fault-detection ability (#faults detected in previous version)

Minimization criteria
- One absolute: maintain statement coverage
- Three relatives: min size test suite, min execution time, max fault-detection capability

Minimization policies
- Seven weighted: same weight; 0.6, 0.3, 0.1 (all combinations)
- One prioritized: (1) min size test suite, (2) min execution time, (3) max fault-detection capability

Overall, 400 minimization problems covering a wide spectrum
RQ1: How often can MINTS find an optimal solution quickly? (Process and results)

MINTS encoded each problem, submitted it to all solvers, and measured the time required to get the first solution.

Ordered by complexity indicator – size of the subject x # test cases
RQ1: How often can MINTS find an optimal solution quickly? (Process and results)

MINTS encoded each problem, submitted it to all solvers, and measured the time required to get the first solution. MINTS always found an optimal solution. All solutions found within 40 sec. Less than 10 seconds for the majority of the most complex minimization problems. In most cases, less than two sec.

Ordered by complexity indicator – size of the subject x # test cases

Tuesday, June 22, 2010
RQ1: How often can MINTS find an optimal solution quickly? (Process and results)

MINTS encoded each problem, submitted it to all solvers, and measured the time required to get the first solution.

- MINTS always found an optimal solution.
- All solutions found within 40 sec.
- Less than 10 seconds for the majority of the most complex minimization problems.
- In most cases, less than two sec.
- Clear correlation between complexity and time required.
- Almost linear; promising wrt scalability.

Ordered by complexity indicator – size of the subject x # test cases.
Test Suite Minimization

Summary

- MINTS is a technique and tool for test suite minimization that
- Allows for specifying a wide range of multi-criteria minimization problems
- Computes (when successful) optimal solutions
- Empirical results show usefulness and applicability of the approach
Outline

• Introduction
• Regression test selection
• Test suite augmentation
• Test suite minimization
• Conclusion
Outline

• Introduction
• Regression test selection
• Test suite augmentation
• Test suite minimization

• Conclusion
Acknowledgements

• **Collaborators:**
  - Taweesup Apiwattanapong
  - Mary Jean Harrold
  - Hwa-You Hsu
  - Wei Jin
  - James Jones
  - Donglin Liang
  - Raul Santelices
  - Nanjuan Shi
  - Saurabh Sinha
  - Tao Xie

• **Funding:**
  - NSF, IBM Research, TCS Ltd., Boeing Aerospace Corporation
Summary

Test suite T → Test-suite maintenance → Test suite Tval → Regression test selection → Test suite T' → Test-suite prioritization → Prioritized Test suite T'

- Modified test suite
- Test-case manipulation
- Minimized test suite
- Test-suite minimization
- Redundant test cases
- Test-suite augmentation
- Obsolete test cases
Summary

Test suite T

Test-suite maintenance

Test suite Tval

Regression test selection

Test suite T'

Test-suite prioritization

Prioritized Test suite T'

Modified test suite

Obsolete test cases

Test-case manipulation

Minimized test suite

Redundant test cases

Test-suite minimization

Test suite Tauq
For more information

- **Web:**
  - Home page: http://www.cc.gatech.edu/~orso/
  - Tools: [http://www.cc.gatech.edu/~orso/software.html](http://www.cc.gatech.edu/~orso/software.html) (or by request)
  - Papers: [http://www.cc.gatech.edu/~orso/papers/](http://www.cc.gatech.edu/~orso/papers/)
- **Email:** orso@cc.gatech.edu