Testing of Evolving Software: Achievements, Challenges, and Promises

Mary Jean Harrold
ADVANCE Professor of Computing
College of Computing
Georgia Institute of Technology

High Cost of Software Failure
Airplane entertainment system (2008)
- Failed for me and most passengers
- 16 hour flight—Atlanta to Mumbai

"The world is fully IT dependent"
[Annie Combelles]
→ increased demand for higher-quality software

Resulted in Increased Research In
Software testing, verification, validation

<table>
<thead>
<tr>
<th>Software Engineering</th>
<th>Languages/ Compilers</th>
<th>Testing, Verification, Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSE</td>
<td>PLDI</td>
<td>ISSTA (annual)</td>
</tr>
<tr>
<td>ESEC/FSE</td>
<td>POPL</td>
<td>ICST</td>
</tr>
<tr>
<td>ICSE</td>
<td>OOPSLA</td>
<td></td>
</tr>
<tr>
<td>ICSM</td>
<td>ECOOP</td>
<td></td>
</tr>
</tbody>
</table>
Resulted in Increased Research In

**Software testing, verification, validation**

Uniqueness of ICST
- reporting good results in testing, verification, validation
- and
- bridging research and practice

**Testing, Verification, Validation**
- ISSTA (annual)
- ICST

Testing Evolving Software
- Interest
- Problems
  - Achievements
  - Challenges
- Industry—academic collaboration

---

**Initial Interest by Chance**
- Searched for problem
- Discovered data-flow testing
- Led to
  - using data-flow information to select test cases to rerun
  - researching other regression testing problems

**The Real Eye Opener**

**Early 1990s**
- Industry/academic testing conference
- Gave invited talk on regression testing

**At Microsoft**
- Collaborated with developers and testers
- Integrated my algorithm into MS tools
- Solved some problems, identified new problems

**In the audience**
Roger Sherman, Director of Test, Microsoft

**After talk**
- Discussed my work
- Invited me to visit
Collaboration With Industry

Common Problem
- Changes require rapid modification and testing for quick release (time to market pressures)
- Causing released software to have many defects

Research Question
- How can we test well to gain confidence in the changes in an efficient way before release of changed software?

Approach
- Concentrate testing around the changes
- Automate (if possible) the regression testing process

Testing Evolving Software

Modify P→P′
Select subset of T to rerun
Execute
Assess adequacy
Assess outcome
Identify faults
Augment T for untested adequacy requirements

Add features
Improve performance

Select Subset of T to Rerun

Which test cases in T should be rerun to test P′?

Solution
Partition T into two subsets
- run one on P′
- don’t run the other
General Test Selection Technique

1. Construct representation $G$ for $P$

$P$ can be
- procedural
- object-oriented
- database
- Web-based
- other

$G$ can be
- low level—e.g., code
- higher level—e.g., UML diagrams, state-transition diagrams, architectural diagrams
- other information—e.g., CVS repository

2. Associate test cases in $T$ with entities in $G$

3. Build $G'$ and compare $G$ and $G'$ to find dangerous entities

4. Select test cases based on dangerous entities

General Test Selection Technique

Empirical Evaluation

Goal: To determine savings in execution time

Subjects

<table>
<thead>
<tr>
<th>C Program</th>
<th>Versions</th>
<th>Procedures</th>
<th>KLOC</th>
<th>Test Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empire</td>
<td>5</td>
<td>766</td>
<td>50</td>
<td>1033</td>
</tr>
<tr>
<td>Java Programs</td>
<td>Versions</td>
<td>Classes</td>
<td>KLOC</td>
<td>Test Cases</td>
</tr>
<tr>
<td>Daikon</td>
<td>5</td>
<td>824</td>
<td>167</td>
<td>200</td>
</tr>
<tr>
<td>JBoss</td>
<td>5</td>
<td>2,403</td>
<td>~1K</td>
<td>639</td>
</tr>
</tbody>
</table>

Procedure

For each pair of versions $v_i$, $v_{i+1}$, measured
- Time to re-run $v_{i+1}$ on all test cases in $T$
- Time to select $T'$ (using DejaVu) + Time to run $T'$ on $v_{i+1}$

Compare times
- Save if $B < A$

Savings in Testing Time Using DejaVu

- Empire: 93%
- Daikon: 36%
- Jboss: 63%
What if T’ has too many test cases for allotted time? want to run most important test cases in T’ first?

Solution
Order (prioritize) T’
• find faults earlier
• get coverage earlier
• etc.

Select Subset of T to Rerun

Program P

P’ Version of P

T-T’

T’

T-T’

T’

T

Select Subset of T to Rerun

Achievements: Research

• Application to
  • models of the system
  • kinds of programs
  • programming languages

• Representations using
  • types of program information

• Empirical evidence of
  • effectiveness
  • efficiency
  • etc.

many examples
Achievements: Industry

In 2002 OOPSLA talk, Bill Gates
• in-house testing tool—selects and prioritizes regression tests
• "the impact had been very dramatic."

In 2004 talk at Georgia Tech, Jim Gray
• prioritizing regression tests being used throughout Microsoft with significant impact

Challenges: Assumptions

Selection/prioritization algorithms assume
1. obsolete test cases removed from T
2. deterministic behavior of P, P’
3. non-distributed development of P, P’

Challenges: Process

Regression testing
1. performed at system level
2. performed manually
3. part of a larger process that is difficult to change
**Assess Adequacy**

How well do T, T’, T” or any test suites exercise P’ with respect to changes?

**Solution**

Change-impact criteria
- to assess existing test suite
- augment test suite

---

**Program and Control-flow Graph (CFG)**

Program Example
1. read X
2. read Y
3. if (X > 2)
   4. Y ++
   5. Y --
4. else
5. if (Y > 2)
6. print “1”
7. print “-1”

**Program and Control-flow Graph (CFG)**

X, Y in [1,10] \(\rightarrow\) 100 test cases

<table>
<thead>
<tr>
<th>Paths</th>
<th>Conditions</th>
<th>Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1: 1,2,3,4,6,7</td>
<td>X&gt;2, Y&gt;1</td>
<td>72</td>
</tr>
<tr>
<td>P2: 1,2,3,4,6,8</td>
<td>X&gt;2, Y≤1</td>
<td>8</td>
</tr>
<tr>
<td>P3: 1,2,3,5,6,7</td>
<td>X≤2, Y&gt;3</td>
<td>14</td>
</tr>
<tr>
<td>P4: 1,2,3,5,6,8</td>
<td>X≤2, Y≤3</td>
<td>6</td>
</tr>
</tbody>
</table>
Program and Control-flow Graph (CFG)

Assess Adequacy

Want

- inputs that show difference between $P$ and $P'$
- criteria that require such test cases

→ Change-impact criteria

Change-Impact Criteria

Want

- inputs that show difference between $P$ and $P'$
- criteria that require such test cases

→ Change-impact criteria

Criteria for change-impact propagation (PIE model [Voas] for change propagation)

- Execution of the change
- Infection of the state after change
- Propagation of state to output where it can be observed

Change-Impact Criteria

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X&gt;2, Y&gt;1$</td>
<td>$16/72$</td>
</tr>
<tr>
<td>$X&gt;2, Y\leq1$</td>
<td>$0/8$</td>
</tr>
<tr>
<td>$X\leq2, Y&gt;3$</td>
<td>$0/14$</td>
</tr>
<tr>
<td>$X\leq2, Y\leq3$</td>
<td>$4/6$</td>
</tr>
</tbody>
</table>

$Y$ must be 2 or 3

20/100 test cases propagate difference
## Change-Impact Criteria

### Criteria for change-impact propagation (PIE model for change propagation)
- **Execution of the change**
- **Infection of the state after change**
- **Propagation of state to output where it can be observed**

### Coverage Criteria
- **DU-Pairs**

<table>
<thead>
<tr>
<th>DU-Pairs</th>
<th>P1, P2</th>
<th>P3, P4</th>
<th>P5, P3</th>
<th>P4, P6</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,3,4,6)</td>
<td>0/14</td>
<td>0/14</td>
<td>0/14</td>
<td>0/14</td>
</tr>
<tr>
<td>(1,3,5,6)</td>
<td>0/14</td>
<td>0/14</td>
<td>0/14</td>
<td>0/14</td>
</tr>
<tr>
<td>(2,5,6)</td>
<td>0/14</td>
<td>0/14</td>
<td>0/14</td>
<td>0/14</td>
</tr>
<tr>
<td>(1,3,4,5)</td>
<td>0/14</td>
<td>0/14</td>
<td>0/14</td>
<td>0/14</td>
</tr>
<tr>
<td>(1,3,5,6)</td>
<td>0/14</td>
<td>0/14</td>
<td>0/14</td>
<td>0/14</td>
</tr>
<tr>
<td>(2,5,6)</td>
<td>0/14</td>
<td>0/14</td>
<td>0/14</td>
<td>0/14</td>
</tr>
</tbody>
</table>

### Example Diagram

1. **Read X**
2. **Read Y**
3. **X > 2**
4. **Y++**
5. **Y--**
6. **Y > 2**
7. **Print “1”**
8. **Print “-1”**

<table>
<thead>
<tr>
<th>Change</th>
<th>DU-Pairs</th>
<th>P1, P2</th>
<th>P3, P4</th>
<th>P4, P6</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1,2,3,5,6,7</td>
<td>16/72</td>
<td>0/14</td>
<td>4/6</td>
</tr>
<tr>
<td>P2</td>
<td>1,2,3,4,6,8</td>
<td>0/14</td>
<td>4/6</td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>1,2,3,5,6,7</td>
<td>0/14</td>
<td>4/6</td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>1,2,3,5,6,8</td>
<td>4/6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

## Change-Impact Criteria

### Setup

- **Random**
  - 20% chance of finding a fault

### Example

- **Read X**
- **Read Y**
- **X > 2**
- **Y++**
- **Y--**
- **Y > 2**
- **Print “1”**
- **Print “-1”**

### 20/100 Test Cases
- **Propagate difference**
  - 0/14
  - 0/8
  - 16/72

### Coverage Criteria
- **Change or statement**
  - 1 → 20%
  - >1 → increase chances but not because of this criterion
- **Generally, better than random**
Change-Impact Criteria

Criteria for change-impact propagation (PIE model for change propagation)

- Execution of the change
- Infection of the state after change
- Propagation of state to a point where it can be observed

Coverage Criteria

- Dependence chains (all paths)

<table>
<thead>
<tr>
<th>Path Condition</th>
<th>Infection Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1: 1,2,3,4,6,7</td>
<td>(X ≤ 2) and not (X &gt; 2) → any value of X shows difference</td>
</tr>
<tr>
<td>P2: 1,2,3,4,6,8</td>
<td></td>
</tr>
<tr>
<td>P3: 1,2,3,5,6,7</td>
<td></td>
</tr>
<tr>
<td>P4: 1,2,3,5,6,8</td>
<td></td>
</tr>
</tbody>
</table>

S3
Infection:
Path condition in P after S3 and path condition in P' after S3' differ

Condition for infection:
(X ≤ 2) and not (X > 2) → any value of X shows difference

S4
Infection:
Value of state after execution of S4 in P and S4' in P' must differ

Condition for infection:
- S4 in P, Y=Y+1
- corresponding location in P', Y=Y
  - Y=Y+1
  - any value of Y

P—path condition
SS—symbolic state
Perform symbolic execution from before change to get conditions
Change-Impact Criteria

### Propagation

<table>
<thead>
<tr>
<th>PC</th>
<th>SS(n)</th>
<th>PC'</th>
<th>SS(n')</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>$X_0 Y_0$</td>
<td>true</td>
<td>$X_0 Y_0$</td>
</tr>
<tr>
<td>$X_0 &gt; 2$</td>
<td>$X_0 Y_0$</td>
<td>$X_0 &lt; 2$</td>
<td>$X_0 Y_0$</td>
</tr>
<tr>
<td>$X_0 \leq 2$</td>
<td>$X_0 Y_0$</td>
<td>$X_0 &gt; 2$</td>
<td>$X_0 Y_0$</td>
</tr>
<tr>
<td>$Y_0 + 1 &gt; 2$</td>
<td>$Y_0 - 1$</td>
<td>$Y_0 + 1 &gt; 2$</td>
<td>$Y_0 - 1$</td>
</tr>
<tr>
<td>$Y_0 \leq 2$</td>
<td>$Y_0 - 1$</td>
<td>$Y_0 &gt; 2$</td>
<td>$Y_0 - 1$</td>
</tr>
</tbody>
</table>

### Resulting conditions that show differences

- **C1**: $Y_0 + 1 > 2$ and $Y_0 - 1 \leq 2$
- **C2**: $Y_0 + 1 \leq 2$ and $Y_0 - 1 > 2$

1. Performs Partial Symbolic Execution

   **Beginning immediately before change**
   - computes conditions in terms of relative variables immediately before change
   - avoids symbolic execution from beginning of program to change

   Don’t need to solve conditions—can still monitor for their satisfaction
2. Performs Partial Symbolic Execution

For some distance instead of to output statements

- computes conditions on states at intermediate points (i.e., distances)
- bounds depth using slicing-like dependences, avoids symbolic execution to outputs

Greater distances improve confidence in propagation to output

---

Insert Probes to Record Coverage

**Program Example**
1. read X
2. read Y
3. if (X ≤ 2)
4.    Y ++
5.   else
6.   Y --
7. if (Y > 2)
8.   print "1"
9.   else
10.  print "-1"

To record adequacy (coverage of conditions)

Instrument modified program so that probes check for satisfaction of condition before change (e.g., before S3)
Challenges

1. more research—e.g.,
   - handling multiple changes
   - tracking actual propagation
   - handling false positives
2. more engineering—e.g.,
   - efficient partial symbolic evaluation
   - effective condition representation
   - various analyses
3. more experimentation—e.g.,
   - determining good distance

Augment T for ... Requirements

How can we get test cases to satisfy unsatisfied conditions?

Solution
Provide assistance for generating test cases for these conditions

- Developers use unsatisfied conditions to create new test cases
- Automatically generate test cases to satisfy conditions
  - use regression test suite and apply existing techniques
  - ensure that the techniques are efficient for large programs
  - accommodate large numbers of infeasible paths
**Create Regression Test Suite**

Program P → Test Suite T

- Modify P → P'
- Select subset of T to rerun
- Execute
- Assess adequacy
- Identify faults
- Assess outcome
- Add features
- Improve performance

What if no regression test suite?

- Assess adequacy
- Augment T for untested adequacy requirements
- Identify faults
- Modify P → P'
- Select subset of T to rerun
- Execute
- Assess outcome
- Assess adequacy
- Assess adequacy

**Assess Outcome**

How can time for checking result be reduced?

Modify P → P'

- Select subset of T to rerun
- Execute
- Assess adequacy
- Identify faults
- Assess outcome
- Assess adequacy
- Assess adequacy

Improve performance

Add features

**Identify Faults**

How can faulty parts of the program be located?

Modify P → P'

- Select subset of T to rerun
- Execute
- Assess adequacy
- Identify faults
- Assess outcome
- Assess adequacy
- Assess adequacy

Improve performance

Add features

**Automate Execution**

How can the execution of the regression test suite be automated?

Modify P → P'

- Select subset of T to rerun
- Execute
- Assess adequacy
- Identify faults
- Assess outcome
- Assess adequacy
- Assess adequacy

Improve performance

Add features
**Industry-Academic Collaboration**

**Lessons Learned**

To **solve** important problems—have **impact**
- interact with industry
- many ideas come from collaborations

To **transfer** results to industry
- show effectiveness on real systems (their systems)
- requires extensive change to process

To **evaluate** research in practice
- integrate prototype into industrial environment
- make prototype usable for developers
- identify internal champion to support evaluation

**Promise: What We Can Expect**

**Questions?**