Software Testing: An Evolution-Centric Perspective

Gregg Rothermel

Dept. of Computer Science and Engineering
University of Nebraska - Lincoln

Supported by the National Science Foundation, Microsoft, Lockheed Martin, and Boeing Commercial Aircraft Group
Evolving Software

Palette CAD 1.0

1.0 Apollo

1.01 Patches

1.1 Features

Palette CAD 2.0

2.0 Apollo

2.01 Patches

2.1 Features
An Evolution-Centric Perspective on Software Testing

- Focus on evolution first
- Harness evolution
- Design for regression testability
Overview of Presentation

• Testing evolving software
• Regression test selection
  – Dejavu algorithm
  – Analytical and empirical evaluation
• Test case prioritization
  – Prioritization measures and techniques
  – Empirical results
• Ongoing and Future Work
Overview of Presentation

• Testing evolving software
• Regression test selection
  – Dejavu algorithm
  – Analytical and empirical evaluation
• Test case prioritization
  – Prioritization measures and techniques
  – Empirical results
• Ongoing and Future Work
Testing Evolving Software

Regression testing

\[
P \rightarrow P' \rightarrow T
\]
Testing Evolving Software

Retest-all

P

T

P'

T
Testing Evolving Software

Regression test selection

\[ P \rightarrow P' \]

\[ T - T' \]

\[ T' \]
Testing Evolving Software

Test case prioritization

\[ P \rightarrow P' \rightarrow t1, t4, t3, t5, t2 \]
Testing Evolving Software

Test suite augmentation and impact analysis

\[ P \rightarrow P' \]

\[ T-T' \]

\[ T' \]

\[ T'' \]

\[ T' \]
Testing Evolving Software

Test suite reduction

\[ P' \]

\[ P \]

\[ T \]
Overview of Presentation

• Testing evolving software
• Regression test selection
  – Dejavu algorithm
  – Analytical and empirical evaluation
• Test case prioritization
  – Prioritization measures and techniques
  – Empirical results
• Ongoing and Future Work
Testing Evolving Software

Regression test selection
Control Flow Graphs

Procedure Avg
S1  count = 0
S2  fread(fptr,n)
S3  while (not EOF) do
S4   if (n<0)
S5     return(error)
else
S6     nums[count] = n
S7     count++
endif
S8   fread(fptr,n)
endwhile
S9  avg = mean(nums,count)
S10 return(avg)
Procedure Avg
S1  count = 0
S2  fread(fptr,n)
S3  while (not EOF) do
S4   if (n<0)
S5     return(error)
else
S6     nums[count] = n
S7     count++
endif
S8    fread(fptr,n)
endwhile
S9  avg = mean(nums,count)
S10 return(avg)

<table>
<thead>
<tr>
<th>test</th>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>empty file</td>
<td>0</td>
</tr>
</tbody>
</table>
Execution Traces

Procedure Avg
S1  count = 0
S2  fread(fptr,n)
S3  while (not EOF) do
S4   if (n<0)
S5      return(error)
else
S6      nums[count] = n
S7      count++
endif
S8      fread(fptr,n)
endwhile
S9  avg = mean(nums,count)
S10 return(avg)
Test History Information

<table>
<thead>
<tr>
<th>test</th>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>empty file</td>
<td>0</td>
</tr>
<tr>
<td>t2</td>
<td>-1</td>
<td>error</td>
</tr>
<tr>
<td>t3</td>
<td>1 2 3</td>
<td>2</td>
</tr>
</tbody>
</table>
Program and Modified Version

Procedure Avg
S1    count = 0
S2    fread(fptr,n)
S3    while (not EOF) do
S4      if (n<0)
S5        return(error)
else
S6      nums[count] = n
S7      count++
endif
S8    fread(fptr,n)
endwhile
S9    avg = mean(nums,count)
S10   return(avg)

Procedure Avg’
S1’    count = 0
S2’    fread(fptr,n)
S3’    while (not EOF) do
S4’      if (n<=0)
S5a     print(“input error”)
else
S6’     nums[count] = n
S7’     count++
endif
S8’    fread(fptr,n)
endwhile
S9’    avg = mean(nums,count)
S10’   return(avg)
CFG and Modified CFG

enter \rightarrow S1
S1 \rightarrow S2
S2 \rightarrow \text{t1,t2,t3}
S3 \rightarrow \text{t2,t3}
S4 \rightarrow F
T
S5 \rightarrow t2
T
S6 \rightarrow \text{t3}
T
S7 \rightarrow S8
S8 \rightarrow S9
S9 \rightarrow S10

\text{exit}

enter' \rightarrow S1'
S1' \rightarrow S2'
S2' \rightarrow \text{t1,t2,t3}
S3' \rightarrow \text{t2,t3}
S4' \rightarrow F
T
S5a \rightarrow \text{t2,t3}
T
S6' \rightarrow S8'
S8' \rightarrow S9'
S9' \rightarrow S10'

\text{exit'}
Example 1

\[ T' = \{t2, t3\} \]
Example 2

\[ T' = \{t2, t3\} \]
Algorithm Dejavu

Input: P, P', T  Output: T'
1. Build CFGs G and G' for P and P'
2. Compare(G.EntryNode,G'.EntryNode)
3. Compare(N,N')
4. mark N “N'-visited”
5. for each pair of successors C and C' of N and N'
6. on equivalently labeled edges do
7. if C is not marked “C'-visited”
8. if C and C' are not lexically identical
10. else
11. Compare(C,C')
Interprocedural Methodologies

1. Compare all pairs of procedures
2. Create & walk interprocedural representation
3. Compare all pairs of procedures identified by configuration management system
Overview of Presentation

• Testing evolving software
• Regression test selection
  – Dejavu algorithm
  – Analytical and empirical evaluation
• Test case prioritization
  – Prioritization measures and techniques
  – Empirical results
• Ongoing and Future Work
Algorithm Efficiency

CFG construction: linear in program size

Graph walk (graph sizes $n, n'$; test set size $t$):

\[ O \left( t \times n \times n' \right) \]

(with multiply-visited nodes)

\[ O \left( t \times \min(n, n') \right) \]

(with no multiply-visited nodes)
Precision and Safety

Conditions:
1. P was correct for all tests in T
2. T contains no obsolete tests
3. Controlled regression testing
Regression Test Selection System

- Program analysis tools
- Dejavu tool
- Code instrumenter
- Analysis database
- Test database
- Test history builder

Signals:
- \( P, P' \)
- \( P, P', \text{cfgs} \)
- \( P, \text{cfg} \)
- \( \text{cfgs} \)
- \( \text{test history} \)
- \( \text{test database} \)
- \( \text{traces} \)
- \( \text{test} \)
- \( \text{selected tests} \)
## Study 1: Empire

<table>
<thead>
<tr>
<th>Program</th>
<th>Procs</th>
<th>LOC</th>
<th>Vers</th>
<th>Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>766</td>
<td>49316</td>
<td>5</td>
<td>1033</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version</th>
<th>Functions Modified</th>
<th>LOC Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>114</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>55</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>726</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>62</td>
</tr>
<tr>
<td>5</td>
<td>42</td>
<td>221</td>
</tr>
</tbody>
</table>
Study 1: Test Selection Percentages

% Tests Selected

Version Number

0 10 20 30 40 50 60 70 80 90 100

1 2 3 4 5
Study 1: Cost Effectiveness

![Bar chart showing time (hours) vs. version number.

Time (Hours) | Version Number
---|---
0:00 | 1
1:00 | 2
2:00 | 3
3:00 | 4
4:00 | 5
5:00 | Retest All
6:00 | Dejavu
7:00 | Dejavu

Legend:
- Yellow bar: Retest All
- Red bar: Dejavu}
### Study 2: Windows NT Calculator

<table>
<thead>
<tr>
<th>Program</th>
<th>Funcs</th>
<th>LOCs</th>
<th>Vers</th>
<th>Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>calculator</td>
<td>27</td>
<td>2145</td>
<td>9</td>
<td>3/388</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version</th>
<th>Functions Modified</th>
<th>LOCs Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>264</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>245</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>44</td>
</tr>
</tbody>
</table>
Study 2:
Test Selection Percentages

% Tests Selected

Version Number

3-test
388-test
Overview of Presentation

• Testing evolving software
• Regression test selection
  – Dejavu algorithm
  – Analytical and empirical evaluation
• Test case prioritization
  – Prioritization measures and techniques
  – Empirical results
• Ongoing and Future Work
Testing Evolving Software

Test case prioritization

P

P'

T

t1, t4, t3, t5, t2
Addressing the Problem
Requires:

(1) Objective function \( f \)
(2) Algorithms/heuristics for maximizing \( f \)
Addressing the Problem Requires:

1. Objective function \( f \)
2. Algorithms/heuristics for maximizing \( f \)
Prioritization Objectives

• Cover system components more quickly
• Build reliability estimates more quickly
• Reveal faults earlier in testing
• Reveal regression faults earlier in testing
• Reveal critical faults earlier in testing
**Prioritization Objectives**

- Cover system components more quickly
- Build reliability estimates more quickly
- Reveal faults earlier in testing
- Reveal regression faults earlier in testing
- Reveal critical faults earlier in testing
Rate of Fault Detection – APFD

<table>
<thead>
<tr>
<th>TESTS</th>
<th>Faults</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

- T1: A-B-C-D-E
- T2: E-D-C-B-A
- T3: C-E-B-A-D

Graphs show the percentage of faults detected at different test suite fractions:
- T1: Area = 50%
- T2: Area = 64%
- T3: Area = 84%
Addressing the Problem Requires:

(1) Objective function $f$
(2) Algorithms (or heuristics) for maximizing $f$
A Simple Technique: Total Statement Coverage Prioritization

table:

<table>
<thead>
<tr>
<th>test</th>
<th>stmts covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>s1,s2,s3,s9,s10,s11</td>
</tr>
<tr>
<td>t2</td>
<td>s1,s2,s3,s4,s5</td>
</tr>
<tr>
<td>t3</td>
<td>s1,s2,s3,s4,s6,s7,s8,s9,s10,s11</td>
</tr>
</tbody>
</table>

st-total: t3, t1, t2
Using Feedback:  
Additional Statement Coverage Prioritization

<table>
<thead>
<tr>
<th>test</th>
<th>stmts covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>s1, s2, s3, s9, s10, s11</td>
</tr>
<tr>
<td>t2</td>
<td>s1, s2, s3, s4, s5</td>
</tr>
<tr>
<td>t3</td>
<td>s1, s2, s3, s4, s6, s7 s8, s9, s10, s11</td>
</tr>
</tbody>
</table>

st-total:  t3, t1, t2

st-addtl:  t3, t2, t1
Using Feedback:  
*Total/Addt’l Function Coverage Prioritization*

<table>
<thead>
<tr>
<th>test</th>
<th>functions covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>f1, f2, f3, f9, f10, f11</td>
</tr>
<tr>
<td>t2</td>
<td>f1, f2, f3, f4, f5</td>
</tr>
<tr>
<td>t3</td>
<td>f1, f2, f3, f4, f6, f7, f8, f9, f10, f11</td>
</tr>
</tbody>
</table>

*func-total: t3, t1, t2
*func-addtl: t3, t2, t1*
Incorporating Modification Info:
Total/Addt’l Modified Function Coverage Prio.

<table>
<thead>
<tr>
<th>test</th>
<th>functs covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>f1, f2, f3, f9, f10, f11</td>
</tr>
<tr>
<td>t2</td>
<td>f1, f2, f3, f4, f5</td>
</tr>
<tr>
<td>t3</td>
<td>f1, f2, f3, f4, f6, f7, f8, f9, f10, f11</td>
</tr>
</tbody>
</table>

fn-mod-total: t2, t1, t3
fn-mod-addtl: t2, t1, t3
Sources of Prioritization Data

• Code Coverage Data
• Modification information
• Test cost data
• Test criticality estimates
• History information
Overview of Presentation

• Testing evolving software
• Regression test selection
  – Dejavu algorithm
  – Analytical and empirical evaluation
• Test case prioritization
  – Prioritization measures and techniques
  – Empirical results
• Ongoing and Future Work
Case Study

• Empire program (60K + LOC)
• 11 sequential versions, several faults each
• 1 large functional test suite
• Various techniques
  – 1 control technique – random (avg of 20 runs)
  – function-level granularity
  – with/without feedback
  – with/without modification information
Mean APFD Values for Empire

![Bar Chart]
Overview of Presentation

- Testing evolving software
- The test case prioritization problem
  - Measuring success
  - Prioritization techniques
- Empirical studies of test case prioritization
- Related work
- Ongoing work
Ongoing Work

• Continued empirical assessment
• Investigate sources of variation in techniques
• Develop models for evaluating costs/benefits
• Develop methods for choosing techniques
• Identify implications for development and testing processes
• Develop process models and guidance mechanisms for practitioners
An Evolution-Centric Perspective on Software Testing

• Focus on evolution first
• Harness evolution
• Design for regression testability
Acknowledgments

• **Sponsors:**
  – National Science Foundation
  – Boeing Commercial Airplane Group
  – Microsoft
  – Lockheed Martin

• **Ph.D. Students:**
  – Hyunsook Do
  – Marc Fisher
  – Jung-Min Kim (U. Md.)
  – Jim Law
  – Alexey Malishevsky
  – Joe Ruthruff

• **Colleagues:**
  – Sebastian Elbaum
  – Mary Jean Harrold
  – Alex Orso
  – Adam Porter
  – David Rosenblum
  – Mary Lou Soffa
  – Roland Untch
  – Elaine Weyuker
Software Testing: An Evolution-Centric Perspective

Gregg Rothermel
Dept. of Computer Science and Engineering
University of Nebraska - Lincoln

Supported by the National Science Foundation,
Microsoft, Lockheed Martin,
and Boeing Commercial Aircraft Group