Towards the Prioritization of Regression Test Suites with Data Flow Information

Matthew J. Rummel
Gregory M. Kapfhammer
Andrew Thall

Symposium on Applied Computing
Santa Fe, New Mexico   March 13-17 2005
Definitions

- **Test Case** – An individual unit test
- **Test Suite** – A tuple of test cases
- **Regression Testing** – Testing that occurs after the completion of development or maintenance activities when a test suite comprised of all accumulated unit tests is executed
- **Test Prioritization** – The process of arranging test cases in a given test suite to facilitate the detection of defects earlier in the execution of the test suite
Motivation

- Regression testing may account for as much as one-half the cost of software maintenance
- Prioritization is often more feasible than test selection
- Tests that fulfill the all-

\textit{DUs} test adequacy criteria are more likely to reveal defects than those that satisfy control flow based criteria
Dataflow

- Model each method in a program as a control flow graph
- Control flow flow family of test criteria (ex: *all-nodes*, *all edges*, *all-paths*)
- Data flow criteria evolved from control flow (ex: *all-DUs*, *all-P-U ses*, *all-C-U ses*)
- Focus on intraprocedural def-use associations
Metrics

- **APFD** – The rate of fault detection per percentage of test suite execution

  \[ APFD(T, P) = 1 - \frac{\sum_{i=1}^{g} \text{reveal}(i, T)}{rg} + \frac{1}{2r} \]

- **PTR** – Percentage of a given test suite that must be executed for all faults to be detected

  \[ PTR(T, P) = \frac{rg}{r} \]
# Metrics Example

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Faults</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$f_1$</td>
<td>$f_2$</td>
</tr>
<tr>
<td>$T_1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_2$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$T_3$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$T_4$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_5$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
</tbody>
</table>

\( \sigma_1 = \langle T_1, T_2, T_3, T_4, T_5 \rangle \) \hspace{1cm} \( \sigma_2 = \langle T_3, T_4, T_1, T_2, T_5 \rangle \)

- \( APFD(T_1, P) = 1 - .4 + .1 = .7 \)
- \( PTR(T_1, P) = \frac{4}{5} \)
- \( APFD(T_2, P) = 1 - .2 + .1 = .9 \)
- \( PTR(T_2, P) = \frac{2}{5} \)
Experiment Design

- Calculate the set of test requirements for program $P$
- Introduce test coverage monitoring instrumentation
- Execute test suites and report $APFD$ and $PTR$ calculations
Cumulative Adequacy of a Test Case

- When a test case has covered both a \textit{def} and corresponding \textit{use} statement, the coverage of that association is stored.

- **Test case adequacy** – The ratio between the number of covered test requirements and the total number of test requirements for all of the methods under test.

\[
a_{\text{adeq}}(T_f) = \frac{\sum_{k=1}^{h} | R_c(m_k) |}{\sum_{k=1}^{h} | R(m_k) |}
\]
Cumulative Adequacy Example

- Model each method in a program as a control flow graph.
- $T_f$ enters method $m$ and executes the true branch of node 3.
- $\text{adeq}(T_f) = \frac{7}{16} = 43.75\%$
Experiments conducted on a GNU/Linux workstation with dual 1GHz Pentium III Xeon processors, 512 MB of main memory

Case study applications:

- **Bank** – 1 class, 53 def-use associations, 5 methods, 7 test cases, 4 seeded errors
- **Identifier** – 3 classes, 81 def-use associations, 13 methods, 11 test cases, 2 sets of 3 seeded errors
- **Money** – 3 classes, 302 def-use associations, 33 methods, 21 test cases, 3 sets of 3 seeded errors
Bank APFD and PTR Measurements

- Prioritized suite has best PTR value
- Prioritized suite has the best APFD value, slightly better than Random1
Identifier *APFD* and *PTR* Measurements

- Prioritized suite has the worst *PTR* value
- Prioritized suite has the worst *APFD* value
Money **APFD** and **PTR** Measurements

- Prioritized suite has best **APFD** for 3 errors, worst for 6 errors, medium for 9 errors.
- Prioritized suite has medium **APFD** for 3 errors, slightly worse than Random1, worst for 6 errors, medium for 9 errors.
Time and Memory Requirements

<table>
<thead>
<tr>
<th>Program</th>
<th>Time (ms)</th>
<th>Space (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bank</td>
<td>3,210</td>
<td>1,084,648</td>
</tr>
<tr>
<td>Identifier</td>
<td>3,351</td>
<td>2,170,871</td>
</tr>
<tr>
<td>Money</td>
<td>9,176</td>
<td>4,984,648</td>
</tr>
</tbody>
</table>

Time and Memory for *InstrumentandEnumerate* Algorithm

- Test case monitoring did not cause significant increases in the time required to execute test cases
Conclusions

- Test suites can be prioritized according to *all-DUs* with minimal time and space overhead.
- Preliminary results indicate that data flow-based prioritizations are not always more effective than random prioritizations.
- Successfully created a low-overhead framework for performing test prioritization which can be used in future studies.
Future Work

- Incorporation of control flow-based and mutation-based adequacy into **Kanonizo**
- The comparison of our prioritization approach to other prioritization schemes beyond random
- The calculation of **APFD** and **PTR** for all permutations of an application’s test suite
- Experimentation with additional case studies that have larger program segments and test suites
- The investigation of prioritization techniques for test suites that must be executed within a specified time constraint
Related Work


Resources