A Comprehensive Framework for Testing Database-Centric Software Applications

Gregory M. Kapfhammer
Department of Computer Science
University of Pittsburgh

PhD Dissertation Defense
April 19, 2007
Dissertation Committee

Director: Dr. Mary Lou Soffa (University of Virginia)

Dr. Panos Chrysanthis (University of Pittsburgh)
Dr. Bruce Childers (University of Pittsburgh)
Dr. Jeffrey Voas (SAIC)

With support and encouragement from countless individuals!
Jeppesen reports airspace boundary problems

About 350 airspace boundaries contained in Jeppesen NavData are incorrect, the FAA has warned. The error occurred at Jeppesen after a software upgrade when information was pulled from a database containing 20,000 airspace boundaries worldwide for the March NavData update, which takes effect March 20.

Important Point: Practically all use of databases occurs from within application programs [Silberschatz et al., 2006, pg. 311].
Research Contributions

- Comprehensive framework that tests a program’s interaction with the complex state and structure of a database
  - Database interaction fault model
  - Database-aware representations
  - Test adequacy
  - Test coverage monitoring
  - Regression testing
- Worst-case analysis of the algorithms and empirical evaluation with six case study applications
Traditional Software Testing

Defects (e.g., bugs, faults, errors) can exist in program $P$ and all aspects of $P$’s environment.

PhD Dissertation Defense, University of Pittsburgh, April 19, 2007 – p. 5
Testing Environment Interactions

Defects can also exist in $P$’s interaction with its environment.
Focus on Database Interactions

$P$ can view and/or modify the state of the database

Program $P$ can view and/or modify the state of the database
Types of Applications

- Database-Centric Applications
  - Interaction Approach
    - Embedded
    - Interface
  - Program Location
    - Inside DBMS
    - Outside DBMS

- Testing framework relevant to all types of applications
- Current tool support focuses on Interface-Outside applications
- Example: Java application that submits SQL Strings to HSQLDB relational database using JDBC drivers
Research Contributions

- Database Interaction Fault Model
- Test Adequacy Criteria
- Test Coverage Monitoring
- Regression Testing
  - Reduction
  - Prioritization
Database Interaction Faults: (1-v)

- $P$ uses update or insert to incorrectly modify items within database
- Commission fault that violates database validity
- Database-aware adequacy criteria can support fault isolation
Database Interaction Faults: (1-c)

- $P$ uses delete to remove incorrect items from database.
- Commission fault that violates database completeness.
- Database-aware adequacy criteria can support fault isolation.

Diagram:

- $P$ uses delete to remove incorrect items from database.
- Commission fault that violates database completeness.
- Database-aware adequacy criteria can support fault isolation.
Data Flow-Based Test Adequacy

→ The intraprocedural database interaction association \( \langle n_3, n_6, R \rangle \) exists within method \( m_i \).
Research Contributions

- Database Interaction Fault Model
- Test Adequacy Criteria
- Test Coverage Monitoring
- Regression Testing
  - Reduction
  - Prioritization
Process: Create a database-aware representation and perform data flow analysis

Purpose: Identify the database interaction associations (i.e., the test requirements)
Database interaction graphs (DIGs) are placed before interaction point

Multiple DIGs can be integrated into a single CFG

Analyze interaction in a control-flow sensitive fashion
Data Flow Time Overhead

2.7% increase in time overhead from $P$ to $P + A_v$ (TM)
Research Contributions

- Database Interaction Fault Model
- Test Adequacy Criteria
- Test Coverage Monitoring
- Regression Testing
  - Reduction
  - Prioritization
**Purpose**: Record how the program interacts with the database during test suite execution
Database-Aware Instrumentation

- Efficiently monitor coverage without changing the behavior of the program under test
- Record coverage information in a database interaction calling context tree (DI-CCT)
Flexible and efficient approach that fully supports both traditional and database-centric applications.
Attach probes to all of the applications in less than nine seconds

Static approach is less flexible than dynamic instrumentation
Static Instrumentation: Space

- Increase in bytecode size may be large (space vs. time trade-off)
Static is faster than dynamic / CCT is faster than DCT

The coverage monitor is both efficient and effective
Size of the Instrumented Applications

<table>
<thead>
<tr>
<th>Compr Tech</th>
<th>Before Instr (bytes)</th>
<th>After Instr (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>29275</td>
<td>887609</td>
</tr>
<tr>
<td>Zip</td>
<td>15623</td>
<td>41351</td>
</tr>
<tr>
<td>Gzip</td>
<td>10624</td>
<td>35594</td>
</tr>
<tr>
<td>Pack</td>
<td>5699</td>
<td>34497</td>
</tr>
</tbody>
</table>

- Average static size across all case study applications
- Compress the bytecodes with general purpose techniques
- Specialized compressor nicely reduces space overhead
## Database Interaction Levels

<table>
<thead>
<tr>
<th>CCT Interaction Level</th>
<th>TCM Time (sec)</th>
<th>Percent Increase (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>7.44</td>
<td>12.39</td>
</tr>
<tr>
<td>Database</td>
<td>7.51</td>
<td>13.44</td>
</tr>
<tr>
<td>Relation</td>
<td>7.56</td>
<td>14.20</td>
</tr>
<tr>
<td>Attribute</td>
<td>8.91</td>
<td>34.59</td>
</tr>
<tr>
<td>Record</td>
<td>8.90</td>
<td>34.44</td>
</tr>
<tr>
<td>Attribute Value</td>
<td>10.14</td>
<td>53.17</td>
</tr>
</tbody>
</table>

- Static instrumentation supports efficient monitoring
- 53% increase in testing time at finest level of interaction
Research Contributions

- Database Interaction Fault Model
- Test Adequacy Criteria
- Test Coverage Monitoring
- Regression Testing
- Reduction
- Prioritization
Database-Aware Regression Testing

- Version specific vs. general approach
- Use paths in the coverage tree as a test requirement
- Prioritization re-orders the test suite so that it is more effective
- Reduction identifies a smaller suite that still covers all of the requirements
Regression testing techniques can be used in the version specific model.
Research Contributions

- Database Interaction Fault Model
- Test Adequacy Criteria
- Test Coverage Monitoring
- Regression Testing
  - Reduction
- Prioritization
Finding the Overlap in Coverage

\( R_j \rightarrow T_i \) means that requirement \( R_j \) is covered by test \( T_i \)

\( T = \langle T_2, T_3, T_6, T_9 \rangle \) cover all of the test requirements
Reducing the Size of the Test Suite

<table>
<thead>
<tr>
<th>App</th>
<th>Rel</th>
<th>Attr</th>
<th>Rec</th>
<th>Attr Value</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>RM  (13)</td>
<td>(7, .462)</td>
<td>(7, .462)</td>
<td>(10, .300)</td>
<td>(9, .308)</td>
<td>(8.25, .365)</td>
</tr>
<tr>
<td>FF  (16)</td>
<td>(7, .563)</td>
<td>(7, .563)</td>
<td>(11, .313)</td>
<td>(11, .313)</td>
<td>(9, .438)</td>
</tr>
<tr>
<td>PI  (15)</td>
<td>(6, .600)</td>
<td>(6, .600)</td>
<td>(8, .700)</td>
<td>(7, .533)</td>
<td>(6.75, .550)</td>
</tr>
<tr>
<td>ST  (25)</td>
<td>(5, .800)</td>
<td>(5, .760)</td>
<td>(11, .560)</td>
<td>(10, .600)</td>
<td>(7.75, .690)</td>
</tr>
<tr>
<td>TM  (27)</td>
<td>(14, .481)</td>
<td>(14, .481)</td>
<td>(15, .449)</td>
<td>(14, .481)</td>
<td>(14.25, .472)</td>
</tr>
<tr>
<td>GB  (51)</td>
<td>(33, .352)</td>
<td>(33, .352)</td>
<td>(33, .352)</td>
<td>(32, .373)</td>
<td>(32.75, .358)</td>
</tr>
<tr>
<td>All (24.5)</td>
<td>(12, .510)</td>
<td>(12.17, .503)</td>
<td>(14.667, .401)</td>
<td>(13.83, .435)</td>
<td></td>
</tr>
</tbody>
</table>

Reduction factor for test suite size varies from .352 to .8
Reducing the Testing Time

GRO reduces test execution time even though it removes few tests.
Preserving Requirement Coverage

\[ \begin{array}{ccccccc}
\text{GRO} & \text{GRC} & \text{GRV} & \text{GRR} & \text{RVR} & \text{RAR} \\
0.98 & 0.96 & 0.91 & 1.0 & 0.99 & 0.98 & 0.94 \\
\end{array} \]

GRO guarantees coverage preservation - others do not
Research Contributions

- Database Interaction Fault Model
- Test Adequacy Criteria
- Test Coverage Monitoring
- Regression Testing
  - Reduction
  - Prioritization
Cumulative Coverage of a Test Suite

Calculate coverage effectiveness of a prioritization: \( \frac{\text{Actual}}{\text{Ideal}} \)
Improving Coverage Effectiveness

GRO is the best choice - original ordering is poor
Conclusions

- Practically all use of databases occurs from within application programs - must test these interactions!
- Incorporate the *state* and *structure* of the database during all testing phases
- Fault model, database-aware representations, test adequacy, test coverage monitoring, regression testing
- Experimental results suggest that the techniques are *efficient* and *effective* for the chosen case study applications
- A new perspective on software testing: it is important to test a program’s interaction with the execution environment
Future Work

- Avoiding database server restarts during test suite execution
- Time-aware regression testing
- Input simplification and fault localization during debugging
- Reverse engineering and mutation testing
- New environmental factors:
  - eXtensible markup language (XML) databases
  - Distributed hash tables
  - Tuple spaces
- Further empirical studies with additional database-centric applications and traditional programs
Further Resources


