The Measured Performance of Database-Aware Test Coverage Monitoring

Gregory M. Kapfhammer†

Department of Computer Science
Allegheny College
http://cs.allegheny.edu/~gkapfham/

University of Pittsburgh, 2007

† In Conjunction with Mary Lou Soffa (UVa/CS), Panos Chrysanthi (Pitt/CS), Bruce Childers (Pitt/CS)
Outline

1. Introduction to Database Applications
   - Motivation
   - What is a Database Application?

2. Introduction to Software Testing
   - Traditional Software Testing
   - A New Testing Paradigm

3. Database-Aware Test Coverage Monitoring
   - Coverage Monitoring Basics
   - Fundamentals of Coverage Monitoring
   - Instrumentation Probes

4. Experimental Study
   - Experiment Design
   - Instrumentation Costs
   - Coverage Monitoring Costs
An Interesting Defect Report

Database Server Crashes
When you run a complex query against Microsoft SQL Server 2000, the SQL Server scheduler may stop responding. Additionally, you receive an error message that resembles the following: **Date Time server Error: 17883 Severity: 1, State: 0 Date Time server Process 52:0 (94c) ...**

An Input-Dependent Defect
This problem occurs when one or more of the following conditions are true: The query contains a `UNION` clause or a `UNION ALL` clause that affects many columns. The query contains several `JOIN` statements. The query has a large estimated cost. **BUG 473858 (SQL Server 8.0)**
An Interesting Defect Report

Database Server Crashes

When you run a complex query against Microsoft SQL Server 2000, the SQL Server scheduler may stop responding. Additionally, you receive an error message that resembles the following: Date Time server Error: 17883 Severity: 1, State: 0 Date Time server Process 52:0 (94c) ...

An Input-Dependent Defect

This problem occurs when one or more of the following conditions are true: The query contains a UNION clause or a UNION ALL clause that affects many columns. The query contains several JOIN statements. The query has a large estimated cost. BUG 473858 (SQL Server 8.0)
A Severe Defect

The Risks Digest, Volume 22, Issue 64, 2003

**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.

An Important Point

Practically all use of databases occurs from within application programs [Silberschatz et al., 2006, pg. 311].
Real World Example

A Severe Defect

The Risks Digest, Volume 22, Issue 64, 2003

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.

An Important Point

Practically all use of databases occurs from within application programs [Silberschatz et al., 2006, pg. 311].
Program and Database Interactions

Program $P$ creates SQL statements in order to view and/or modify the state of the relational database.

Basic Operation

- select
- insert
- update
- delete

$P$ interacts with a set of databases $D_l, \ldots, D_e$.
Database Interaction Granularity

Database Interactions

Program $P$ interacts with two relational databases $D_k$ and $D_l$ at different levels of granularity (relation, record, attribute, ...)

Gregory M. Kapfhammer
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 an HSQLDB relational database using a JDBC driver
Outline

1. Introduction to Database Applications
   - Motivation
   - What is a Database Application?

2. Introduction to Software Testing
   - Traditional Software Testing
   - A New Testing Paradigm

3. Database-Aware Test Coverage Monitoring
   - Coverage Monitoring Basics
   - Fundamentals of Coverage Monitoring
   - Instrumentation Probes

4. Experimental Study
   - Experiment Design
   - Instrumentation Costs
   - Coverage Monitoring Costs
Focus on Testing Individual Components

Traditional Assumption
Defects may exist in program $P$ and/or $P$’s execution environment
Various Approaches to Software Testing

- **Structural Testing**
- **Random Testing**
- **Software Reliability**
- **Specification Testing**

**Techniques and Supporting Tools**

**Structural testing** requires a test coverage monitor!
A New Direction in Software Testing

Defects may exist in $P$'s interaction with its environment. This suggests the need for a database-aware test coverage monitor!
Outline

1. Introduction to Database Applications
   - Motivation
   - What is a Database Application?

2. Introduction to Software Testing
   - Traditional Software Testing
   - A New Testing Paradigm

3. Database-Aware Test Coverage Monitoring
   - Coverage Monitoring Basics
   - Fundamentals of Coverage Monitoring
   - Instrumentation Probes

4. Experimental Study
   - Experiment Design
   - Instrumentation Costs
   - Coverage Monitoring Costs
Coverage Criteria for Database Applications

Find defects in the database interactions by ensuring that the test suite covers all of the possible **def-use associations** and/or **calling contexts**
Challenges of Database-Aware Monitoring

SQL Statement

```sql
select Path
from Files
where ucase(Path) like '%/usr/bin/bi%'
```

Testing Challenges

Traditional coverage monitoring **does not reveal** how the test case causes the method to **interact** with the database.
Overview of the Coverage Monitoring Process

Program → Instrumentation → Instrumented Program → Instrumented Test Suite → Test Coverage Monitoring → Test Requirements → Adequacy Calculation → Adequacy Measurements → Adequacy Criterion

Current Considerations
Focus on the design, implementation, and performance evaluation of the instrumentation and coverage monitoring components.
Instrumentation Probes

Use **static** and **dynamic** (load-time) instrumentation techniques to insert coverage monitoring probes.

Coverage Trees

Store the coverage results in a tree in order to support the calculation of many types of coverage (e.g., data flow or call tree).
## Comparing the Coverage Trees

### Tree Characteristics

<table>
<thead>
<tr>
<th>Tree</th>
<th>DB?</th>
<th>Context</th>
<th>Probe Time</th>
<th>Tree Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCT</td>
<td>×</td>
<td>Partial</td>
<td>Low - Moderate</td>
<td>Low</td>
</tr>
<tr>
<td>DCT</td>
<td>×</td>
<td>Full</td>
<td>Low</td>
<td>Moderate - High</td>
</tr>
<tr>
<td>DI-CCT</td>
<td>✓</td>
<td>Partial</td>
<td>Moderate</td>
<td>Moderate</td>
</tr>
<tr>
<td>DI-DCT</td>
<td>✓</td>
<td>Full</td>
<td>Moderate</td>
<td>High</td>
</tr>
</tbody>
</table>

### Table Legend

- **Database?** ∈ \{×, ✓\}
- **Context** ∈ \{Partial, Full\}
- **Probe Time Overhead** ∈ \{Low, Moderate, High\}
- **Tree Space Overhead** ∈ \{Low, Moderate, High\}
**Important Goal**

**Efficiently** monitor coverage of database state and structure without changing the behavior of the program under test.
Phases of Coverage Monitoring

Monitoring Operations

Database-aware probes:
- Capture the SQL String
- Consult the database schema and result set meta-data
- Extract and analyze portions of the database state
- Update the coverage tree
Relational Differencing

Handling Database Modifications

The probes use relational differencing to determine that record $t_2$ and attribute value $t_2[2]$ were modified by the SQL \texttt{UPDATE} command.
Outline

1. Introduction to Database Applications
   - Motivation
   - What is a Database Application?

2. Introduction to Software Testing
   - Traditional Software Testing
   - A New Testing Paradigm

3. Database-Aware Test Coverage Monitoring
   - Coverage Monitoring Basics
   - Fundamentals of Coverage Monitoring
   - Instrumentation Probes

4. Experimental Study
   - Experiment Design
   - Instrumentation Costs
   - Coverage Monitoring Costs
Characterizing the Case Study Applications

<table>
<thead>
<tr>
<th>Application</th>
<th># Tests</th>
<th>Test NCSS / Total NCSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>R M</td>
<td>13</td>
<td>227/548 = 50.5%</td>
</tr>
<tr>
<td>F F</td>
<td>16</td>
<td>330/558 = 59.1%</td>
</tr>
<tr>
<td>P I</td>
<td>15</td>
<td>203/579 = 35.1%</td>
</tr>
<tr>
<td>S T</td>
<td>25</td>
<td>365/620 = 58.9%</td>
</tr>
<tr>
<td>T M</td>
<td>27</td>
<td>355/748 = 47.5%</td>
</tr>
<tr>
<td>G B</td>
<td>51</td>
<td>769/1455 = 52.8%</td>
</tr>
</tbody>
</table>
## Details about the Database Interactions

### Interaction Counts

<table>
<thead>
<tr>
<th>Application</th>
<th><code>executeUpdate</code></th>
<th><code>executeQuery</code></th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>RM</td>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>FF</td>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>PI</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>ST</td>
<td>4</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>TM</td>
<td>36</td>
<td>9</td>
<td>45</td>
</tr>
<tr>
<td>GB</td>
<td>11</td>
<td>23</td>
<td>34</td>
</tr>
</tbody>
</table>
Attach probes to all of the applications in less than nine seconds

Statically inserting probes increases space overhead
### Coverage Monitoring Time: Static Versus Dynamic

<table>
<thead>
<tr>
<th>Instr</th>
<th>Tree</th>
<th>TCM Time (sec)</th>
<th>Per Incr (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>CCT</td>
<td>7.44</td>
<td>12.5</td>
</tr>
<tr>
<td>Static</td>
<td>DCT</td>
<td>8.35</td>
<td>26.1</td>
</tr>
<tr>
<td>Dynamic</td>
<td>CCT</td>
<td>10.17</td>
<td>53.0</td>
</tr>
<tr>
<td>Dynamic</td>
<td>DCT</td>
<td>11.0</td>
<td>66.0</td>
</tr>
</tbody>
</table>

**Discussion**

Static has poor space overhead but leads to a minimal increase in testing time. Static is less flexible than dynamic.
Further Comparison of Static Versus Dynamic

Discussion
Static is faster than dynamic / CCT is faster than DCT
## Varying Database Interaction Granularity

### Time Overhead

<table>
<thead>
<tr>
<th>DB Level</th>
<th>TCM Time (sec)</th>
<th>Per Incr (%)</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>

### Discussion

Static supports **efficient** monitoring since there is a 53% increase in testing time at the **finest** level of interaction.
Conclusions and Future Work

Concluding Remarks

- A new **perspective** on software testing and an **efficient** and **effective** database-aware test coverage monitor

Future Work

- Perform demand-driven instrumentation
- Use the coverage tree to **reduce** or **prioritize** a test suite
- Conduct experiments with larger database applications

Resources

- [http://cs.allegheny.edu/~gkapfham/research/diatoms/](http://cs.allegheny.edu/~gkapfham/research/diatoms/)