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 Chrysanthis (Pitt/CS), Bruce Childers (Pitt/CS)

Outline

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)**

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].

Real World Example

A Severe Defect

The Risks Digest, Volume 22, Issue 64, 2003

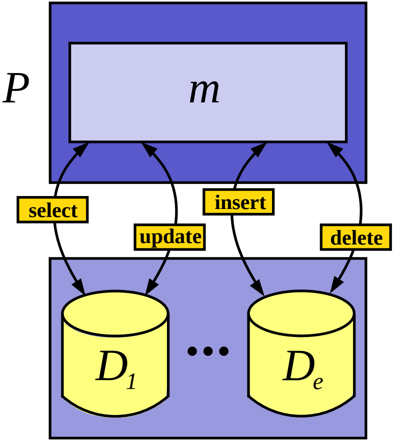
Jeppesen reports airspace boundary problems

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].

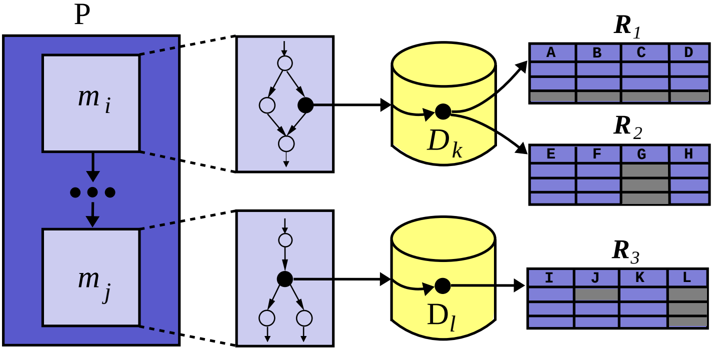
Program and Database Interactions



Basic Operation

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

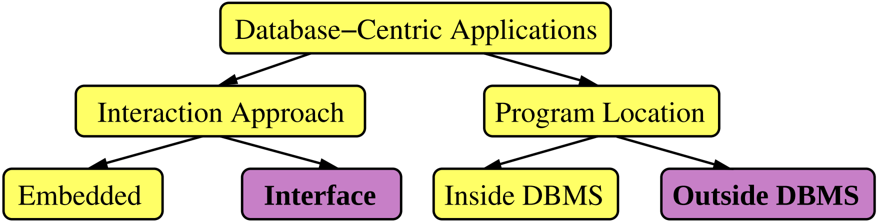
Database Interaction Granularity



Database Interactions

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

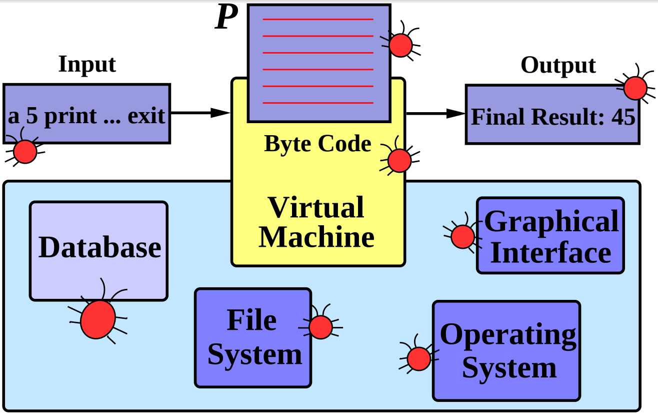
Types of Applications



- 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

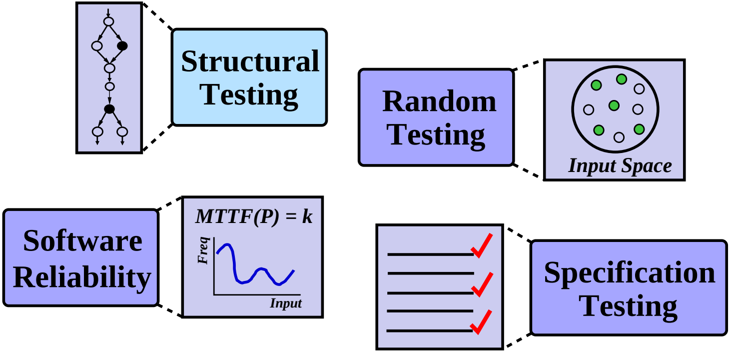
Focus on Testing Individual Components



Traditional Assumption

Defects may exist in program *P* and/or *P*'s execution environment

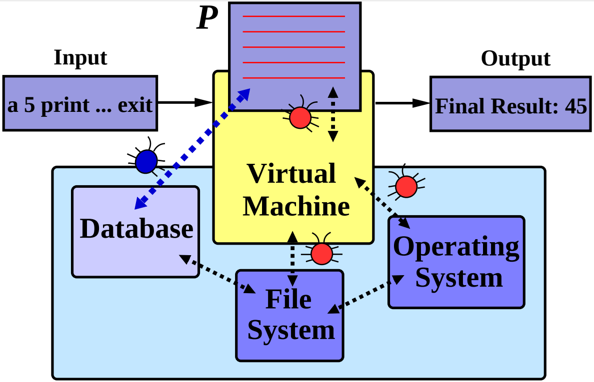
Various Approaches to Software Testing



Techniques and Supporting Tools

Structural testing requires a test coverage monitor!

Testing Environment Interactions

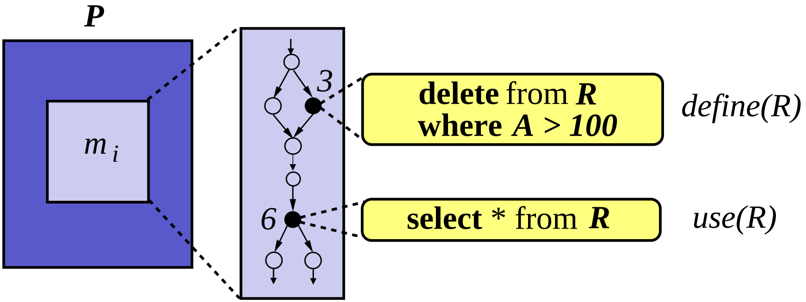


A New Direction in Software Testing

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

Outline

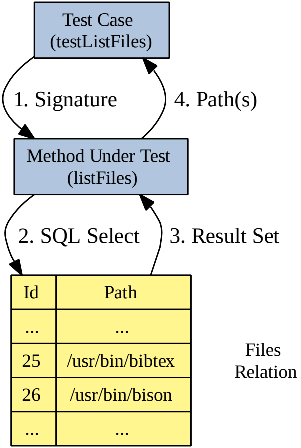
Coverage Criteria for Database Applications



Candidates for Coverage Monitoring

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



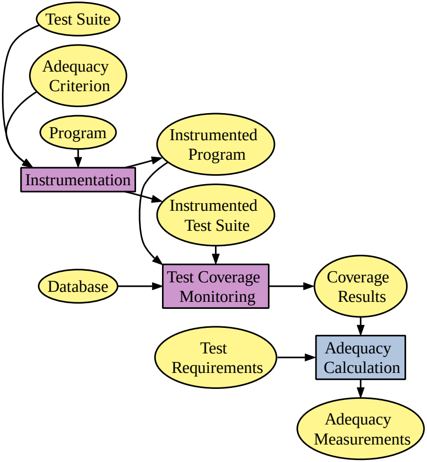
SQL Statement

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

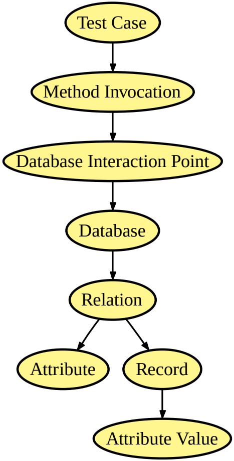
Overview of the Coverage Monitoring Process



Current Considerations

Focus on the design, implementation, and performance evaluation of the **instrumentation** and **coverage monitoring** components

Database-Aware Coverage Trees



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

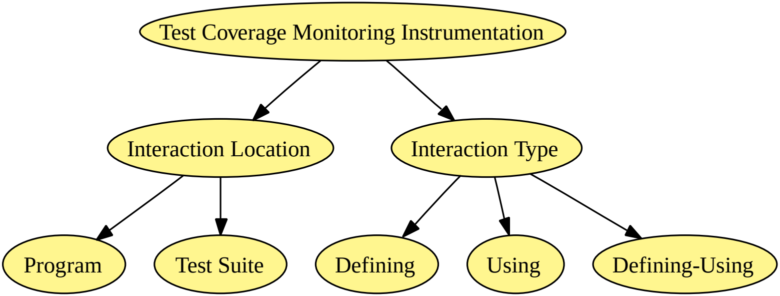
Tree Characteristics

Tree	DB?	Context	Probe Time	Tree Space
CCT	$\times$	Partial	Low - Moderate	Low
DCT	$\times$	Full	Low	Moderate - High
DI-CCT	$\checkmark$	Partial	Moderate	Moderate
DI-DCT	$\checkmark$	Full	Moderate	High

Table Legend

Database? $\in \{\times, \checkmark\}$
Context $\in$ {Partial, Full}
Probe Time Overhead $\in$ {Low, Moderate, High}
Tree Space Overhead $\in$ {Low, Moderate, High}

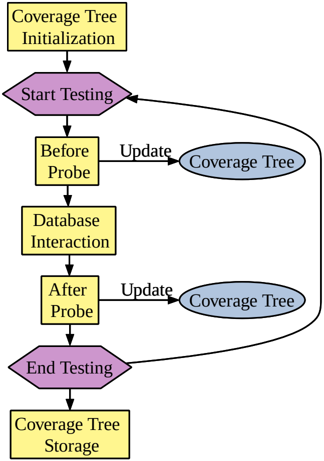
Database-Aware Instrumentation



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

rel_j

	A_1	A_2
t_1	1	2
t_2	2	3
t_3	3	4

Before

$rel_{j'}$

	A_1	A_2
t_1	1	2
t_2	2	4
t_3	3	4

After

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 `UPDATE` command

Outline

Characterizing the Case Study Applications

Test Suites

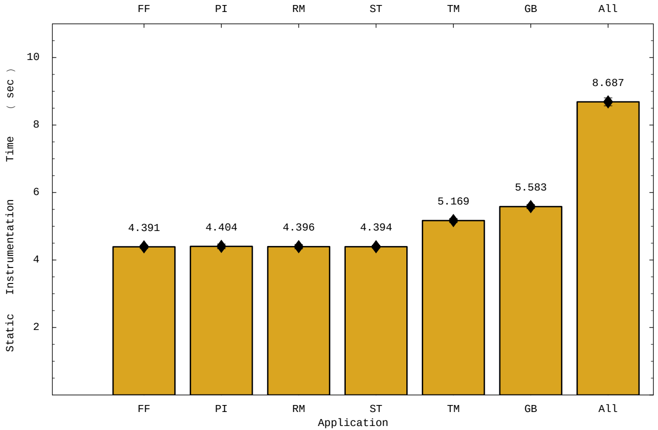
Application	# Tests	Test NCSS / Total NCSS
RM	13	$227/548 = 50.5\%$
FF	16	$330/558 = 59.1\%$
PI	15	$203/579 = 35.1\%$
ST	25	$365/620 = 58.9\%$
TM	27	$355/748 = 47.5\%$
GB	51	$769/1455 = 52.8\%$

Details about the Database Interactions

Interaction Counts

Application	executeUpdate	executeQuery	Total
RM	3	4	7
FF	3	4	7
PI	3	2	5
ST	4	3	7
TM	36	9	45
GB	11	23	34

Static Instrumentation Costs



- Attach probes to all of the applications in less than nine seconds
- Statically inserting probes increases space overhead

Coverage Monitoring Time: Static Versus Dynamic

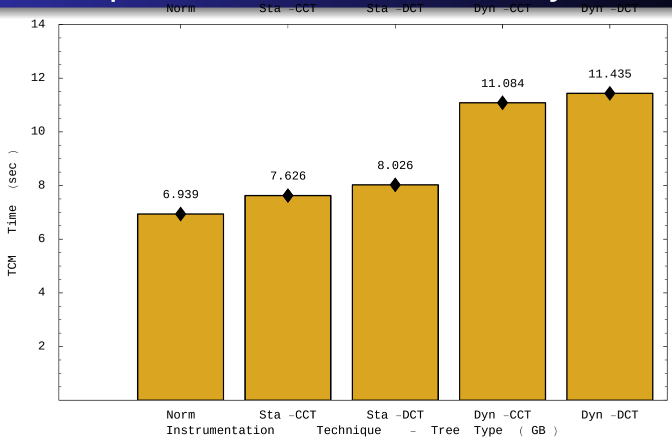
Time Overhead

Instr	Tree	TCM Time (sec)	Per Incr (%)
Static	CCT	7.44	12.5
Static	DCT	8.35	26.1
Dynamic	CCT	10.17	53.0
Dynamic	DCT	11.0	66.0

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

DB Level	TCM Time (sec)	Per Incr (%)
Program	7.44	12.39
Database	7.51	13.44
Relation	7.56	14.20
Attribute	8.91	34.59
Record	8.90	34.44
Attribute Value	10.14	53.17

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/