Database-Aware Test Coverage Monitoring

†Gregory M. Kapfhammer and ‡Mary Lou Soffa

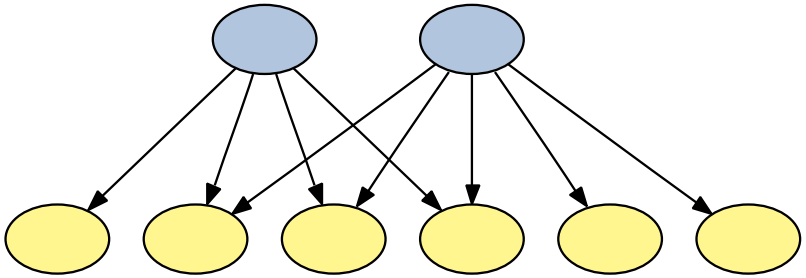
†Department of Computer Science
Allegheny College
http://www.cs.allegheny.edu/~gkapfham/

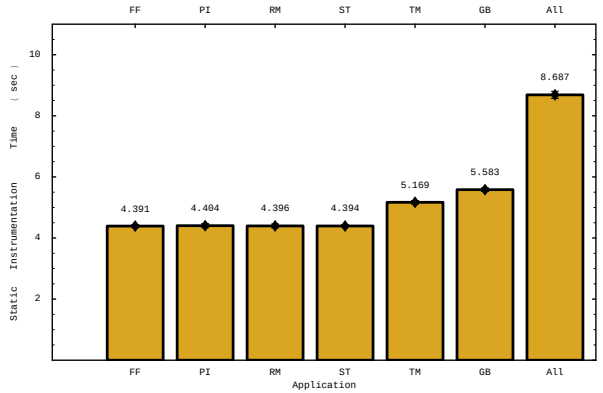
‡Department of Computer Science
University of Virginia
http://www.cs.virginia.edu/~soffa/

India Software Engineering Conference

February, 2008

Important Contributions





Test Coverage Monitors

Experimental Results

A **comprehensive framework** that supports test coverage monitoring for **database** applications

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) ...**

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)**

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) ...**

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

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.

Important Point

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

Real World Example

Severe Defect

The Risks Digest, Volume 22, Issue 64, 2003

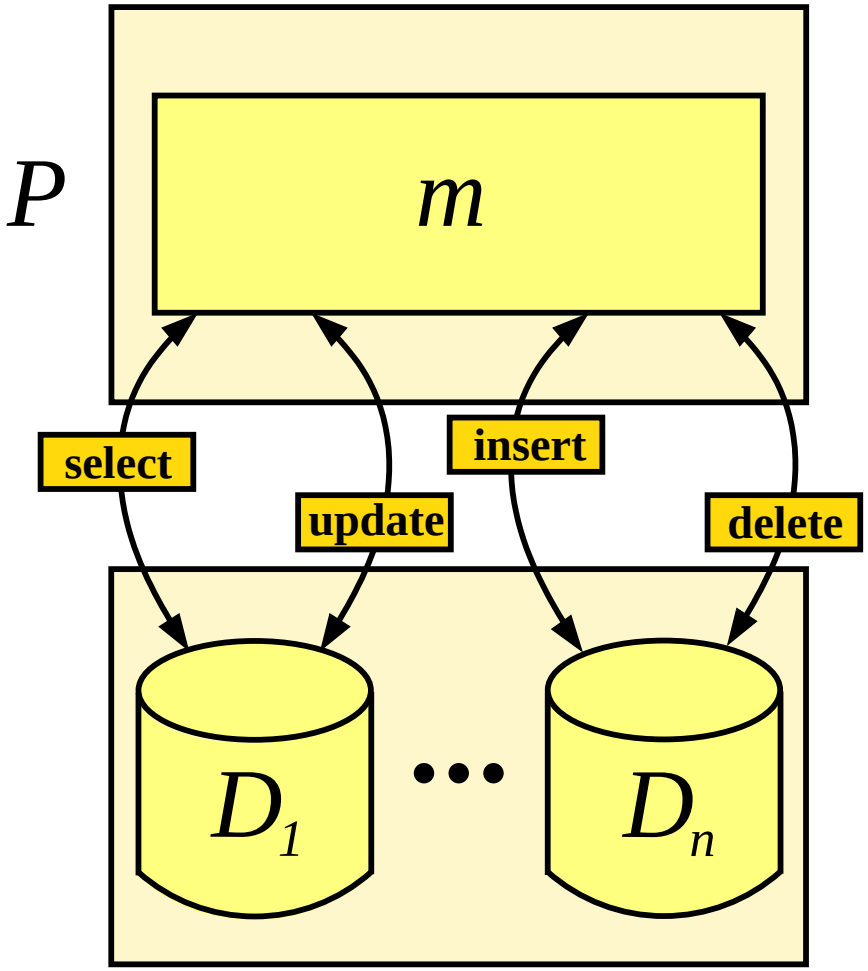
Jeppesen reports airspace boundary problems

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.

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

SQL Construction

Static analysis **does not** reveal the exact SQL command since the program **constructs** the full SQL statement at run-time

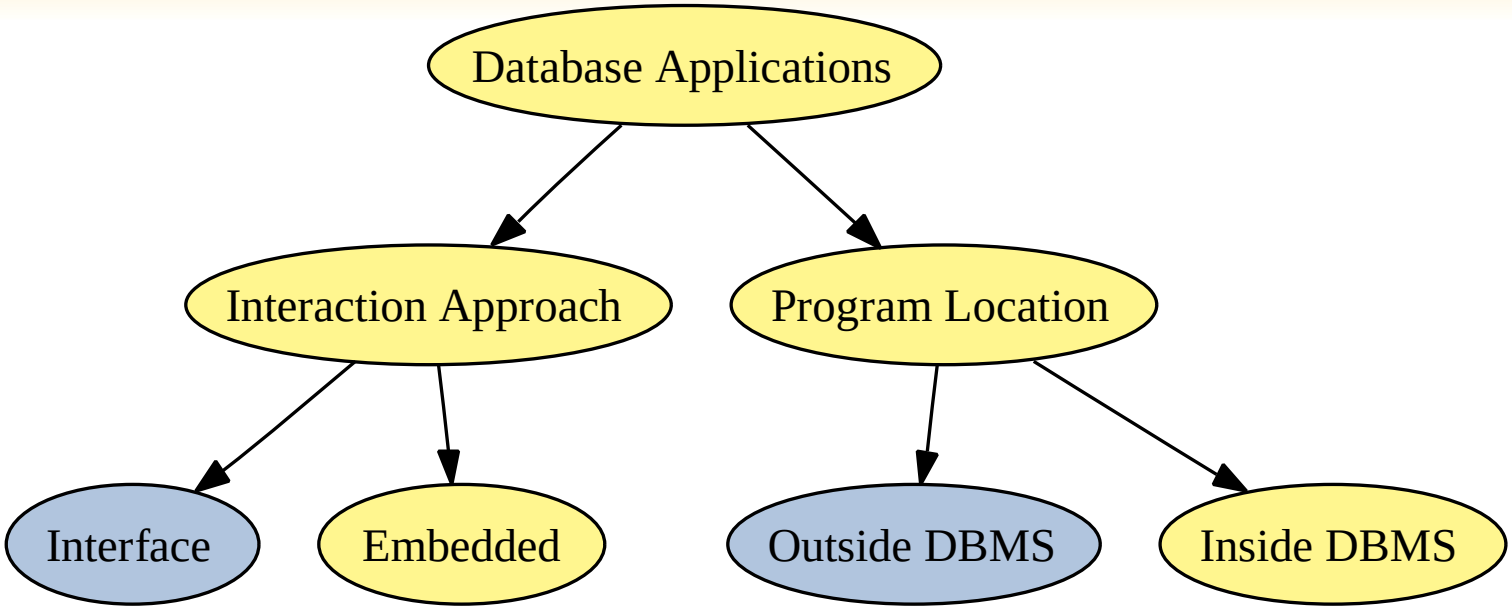
Program and Database Interactions

	A_1	A_2	...	A_n
t_1	2	3	4	5
t_2	1	5	9	12
t_3	2	3	4	5
t_4	2	4	0	1
t_5	4	4	2	5

Database Interactions

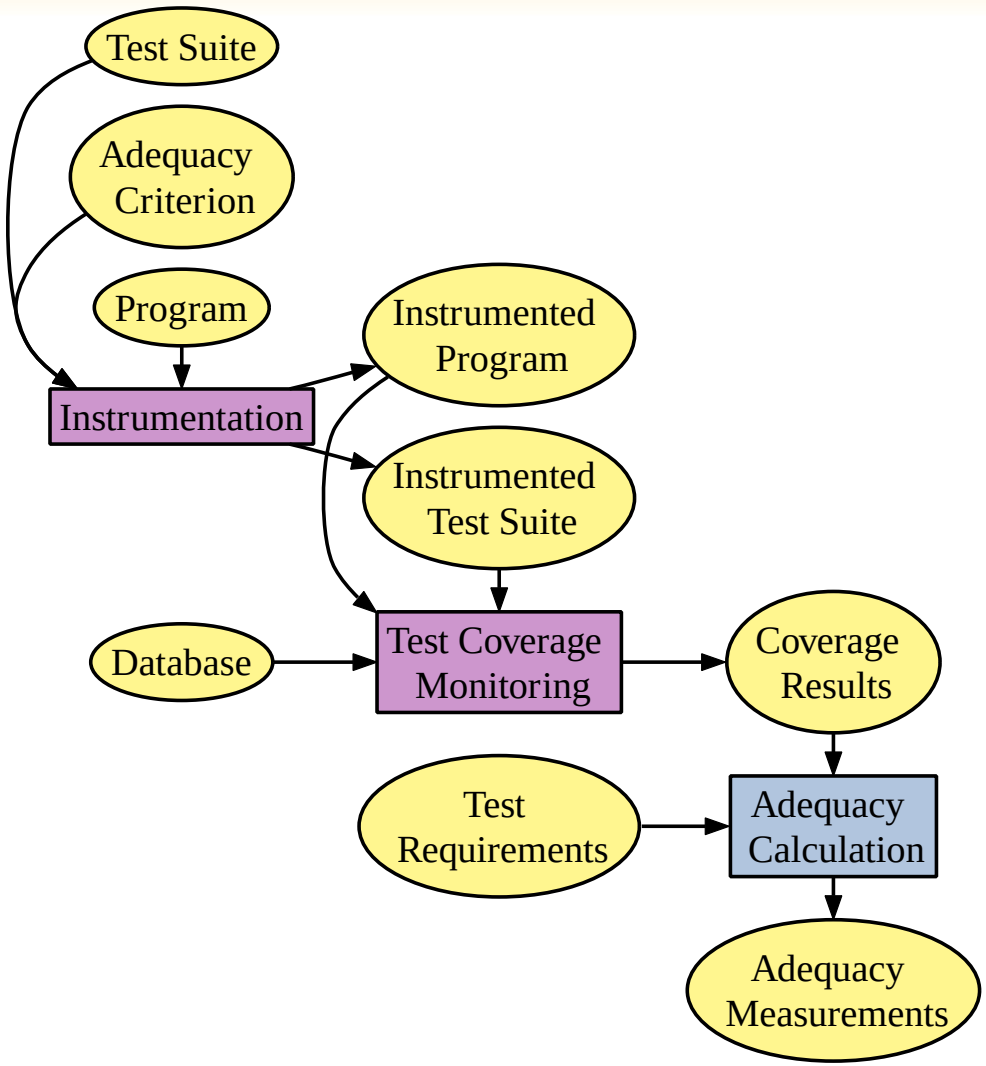
A **program** interacts with a **relational database** at different levels of granularity (database, relation, record, attribute, attribute value)

Types of Applications



- Monitoring framework is 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

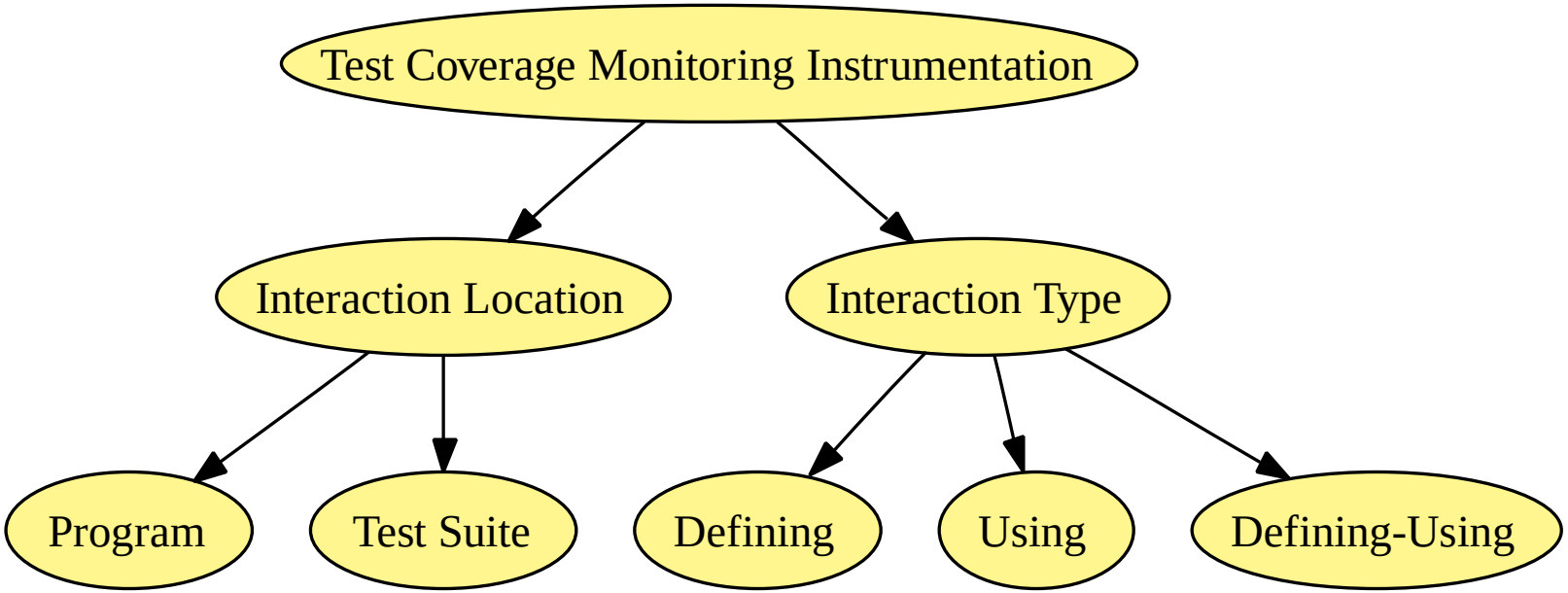
Coverage Monitoring Process



Use instrumentation **probes** to capture and analyze a program's **interaction** with the databases

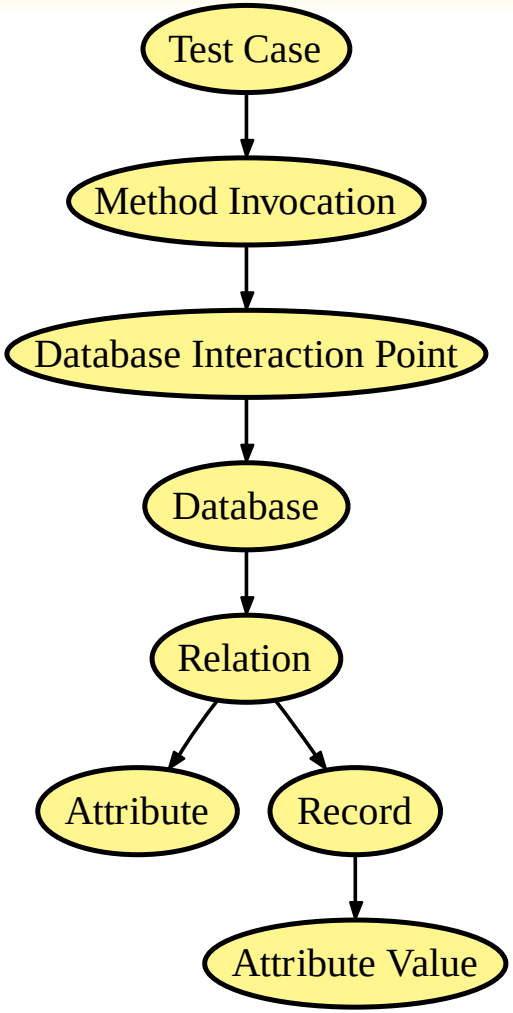
Use the **adequacy** measurements to support both test suite **reduction** and **prioritization**

Database-Aware Instrumentation



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

Database-Aware Coverage Trees



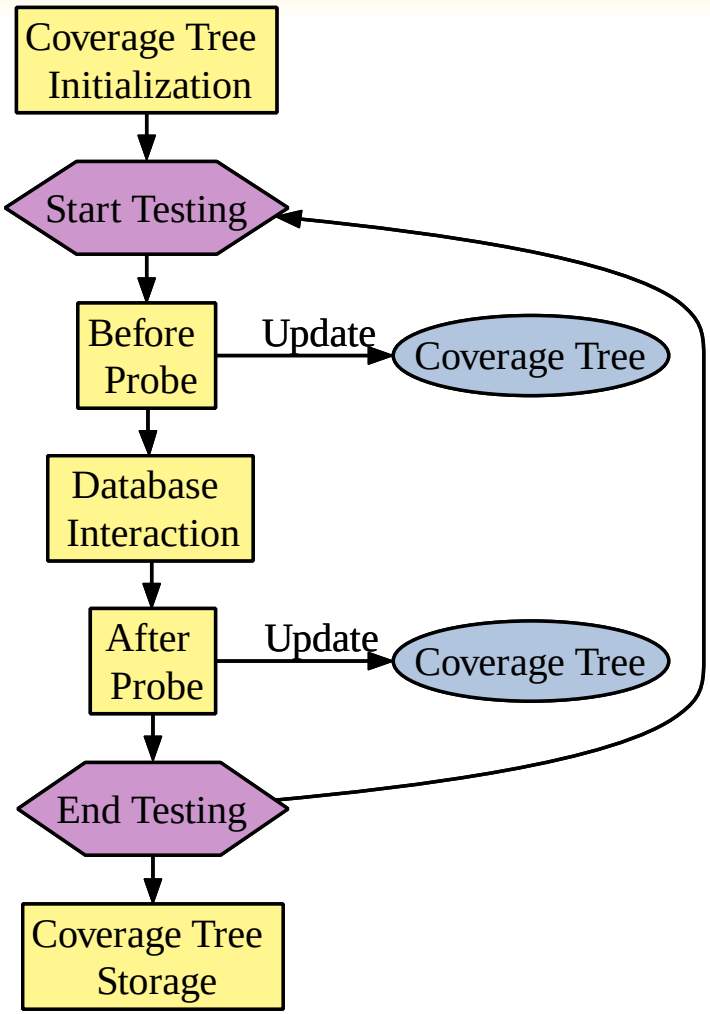
Instrumentation Probes

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**)

Phases of Coverage Monitoring



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

Comparing the Coverage Trees

Tree Characteristics

Tree	DB?	Context	Probe Time	Tree Space
CCT	×	Partial	Low - Moderate	Low
DCT	×	Full	Low	Moderate - High
DI-CCT	✓	Partial	Moderate	Moderate
DI-DCT	✓	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$\}$

Case Study Applications

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	$769/1455 = 52.8\%$

Future Work: replicate the study with larger database applications

Details About the Database Interactions

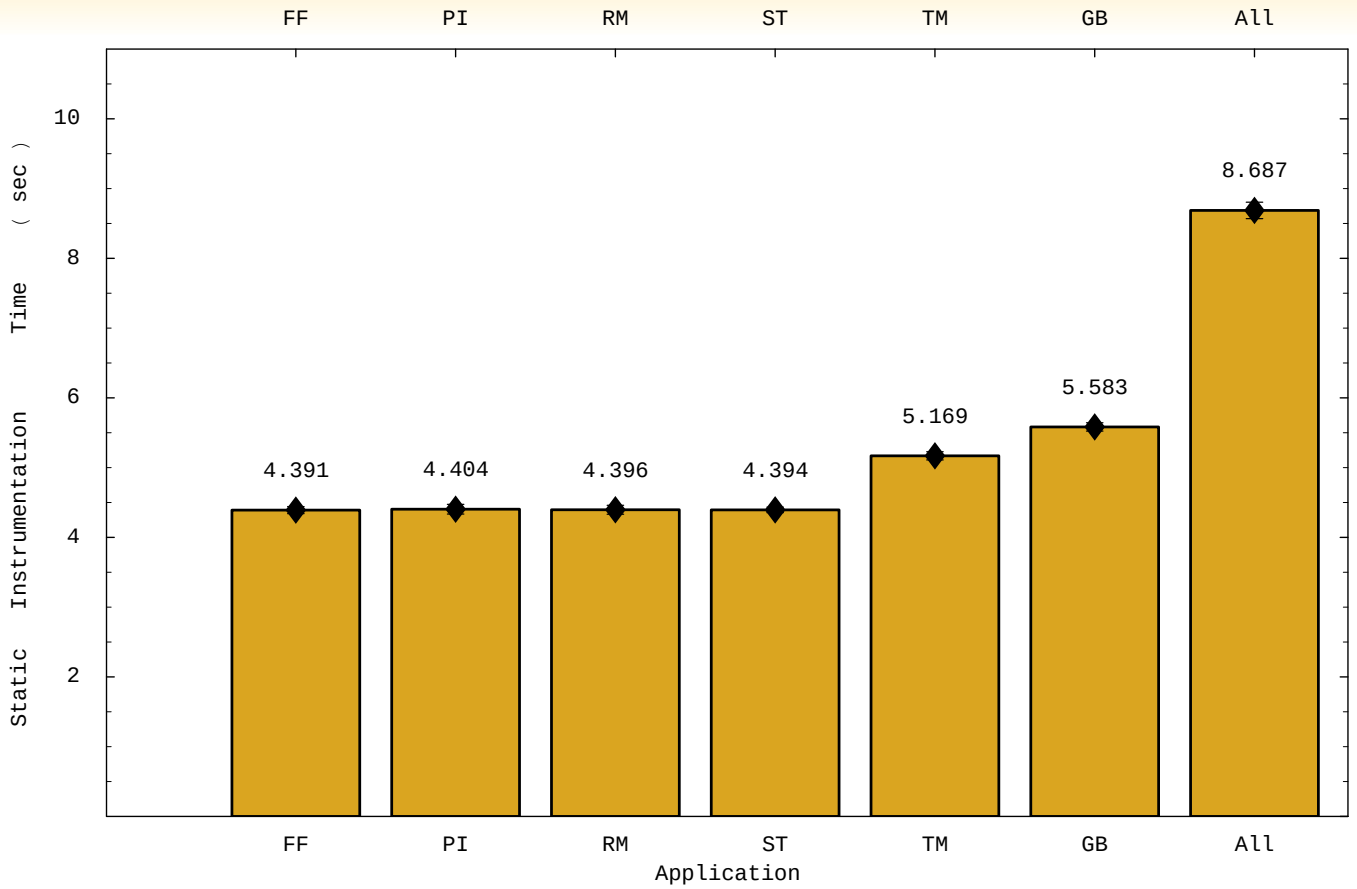
Static Interaction Counts

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

Dynamic Interaction Counts

Database interactions that occur in **iterative** or **recursive** computations are executed more frequently

Static Instrumentation Costs



Static instrumentation process incurs **low** time overhead

Static Versus Dynamic Instrumentation

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

- Static has **high** space overhead but it leads to a **minimal** increase in test coverage monitoring (TCM) time
- Time and space overhead trends are due to the use of **AspectJ**
- Static is **less flexible** than dynamic when program **changes**

Varying Database Interaction Granularity

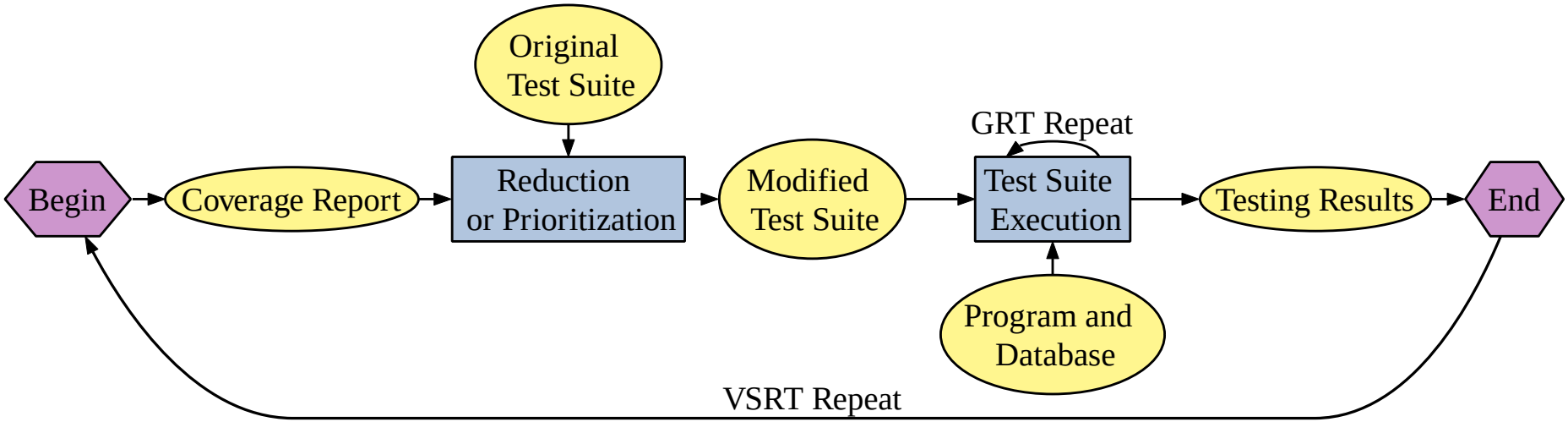
Time Overhead

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

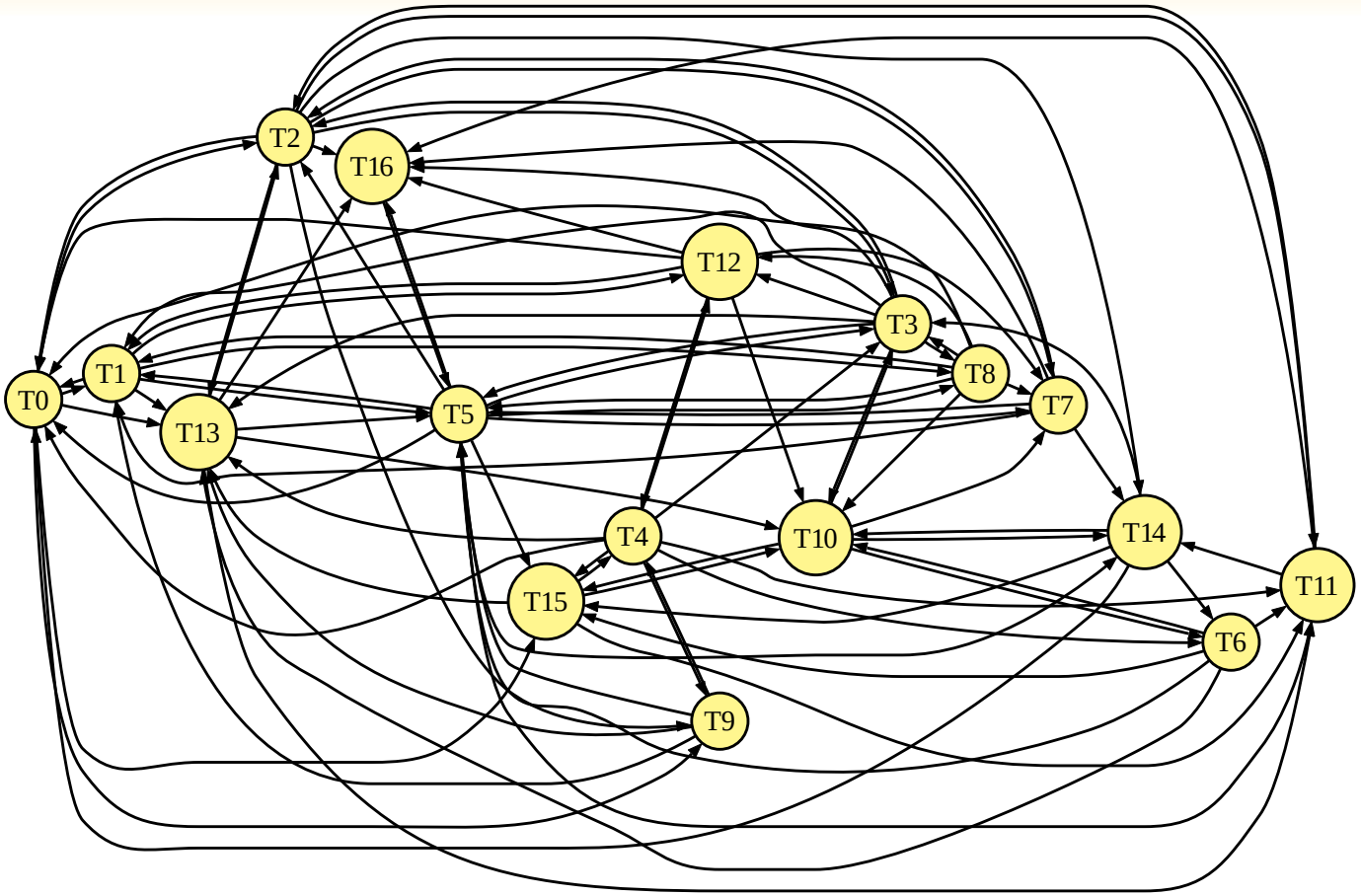
Database-Aware Regression Testing



Regression Testing Overview

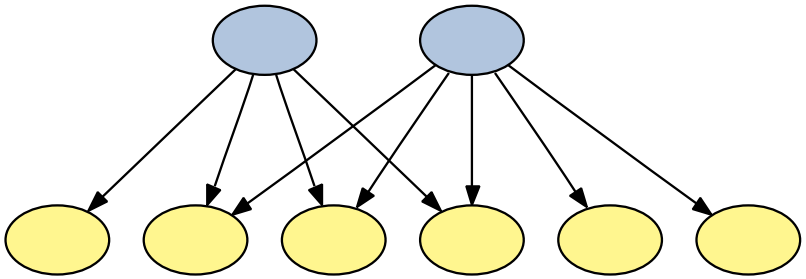
Reduction aims to find a **smaller** test suite that **covers** the same requirements as the original suite. **Prioritization** re-orders the tests so that they **cover** the requirements more **effectively**.

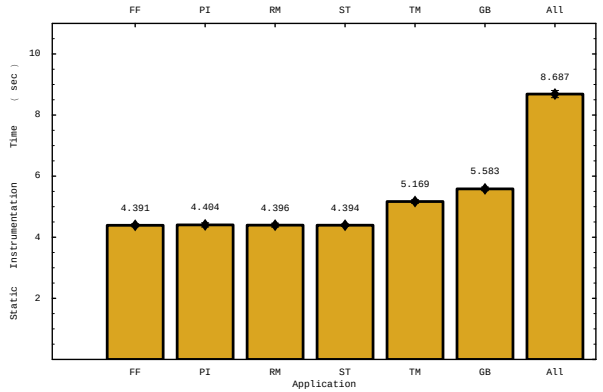
Avoiding Database Restarts



Use **prioritization** to **avoid** costly database **restarts**

Concluding Remarks





Test Coverage Monitors

Experimental Results

A **comprehensive framework** that supports test coverage monitoring for **database** applications

http://www.cs.allegheny.edu/~gkapfham/research/diatoms/