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!

Motivation

The Risks Digest, Volume 22, Issue 64, 2003

Jeppesen reports airspace boundary problems

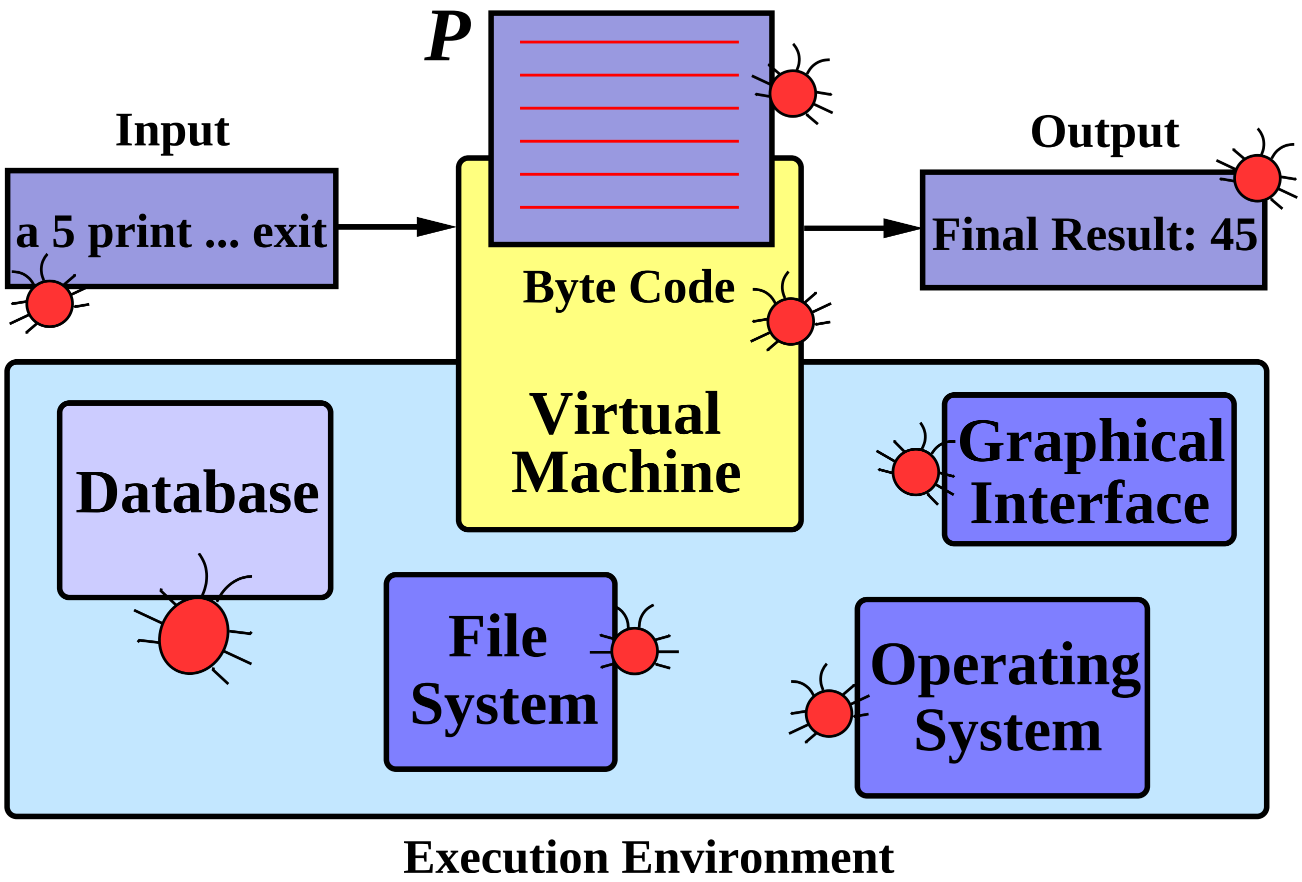
About 350 airspace boundaries contained in Jeppesen NavData are incorrect, the FAA has warned. The error occurred at Jeppesen after a software upgrade when information was pulled from a database containing 20,000 airspace boundaries worldwide for the March NavData update, which takes effect March 20.

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

Research Contributions

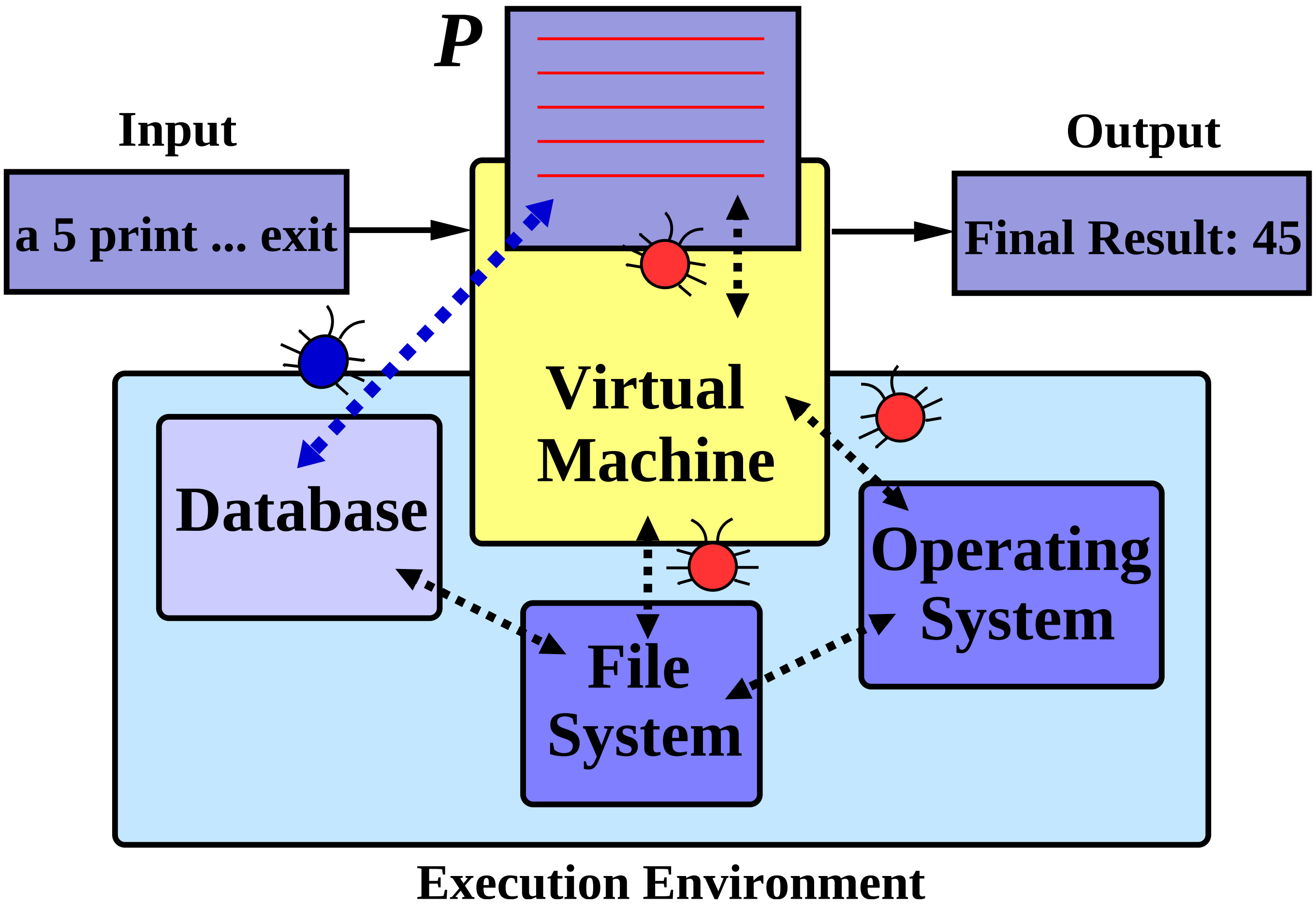
- Comprehensive framework that tests a program's interaction with the complex *state* and *structure* of a database
  - Database interaction fault model
  - Database-aware representations
  - Test adequacy
  - Test coverage monitoring
  - Regression testing
- Worst-case analysis of the algorithms and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

Traditional Software Testing



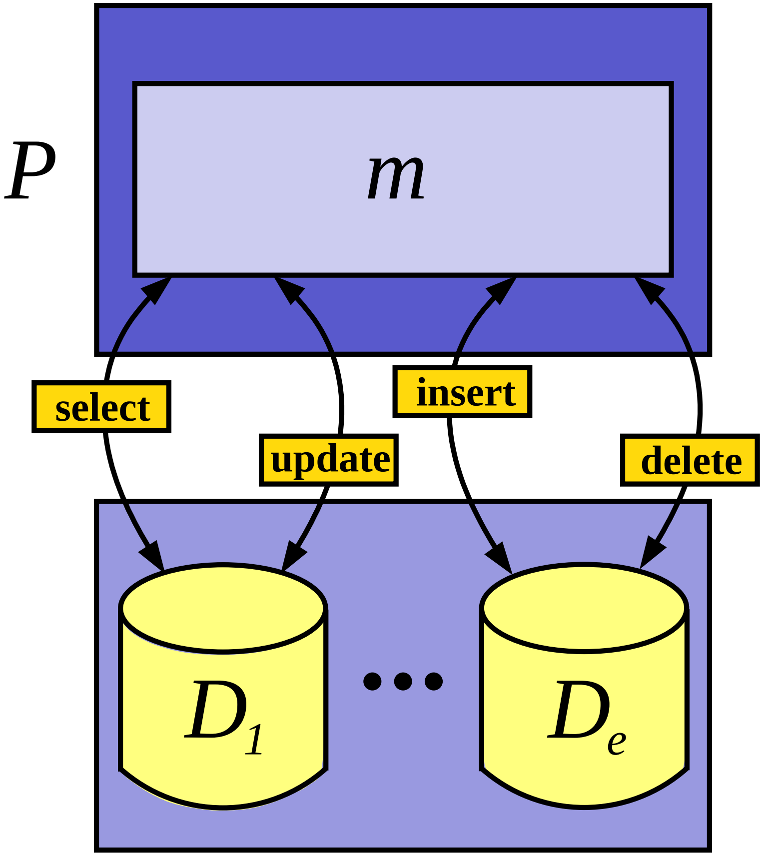
- Defects (e.g., bugs, faults, errors) can exist in program P and all aspects of P's environment

Testing Environment Interactions



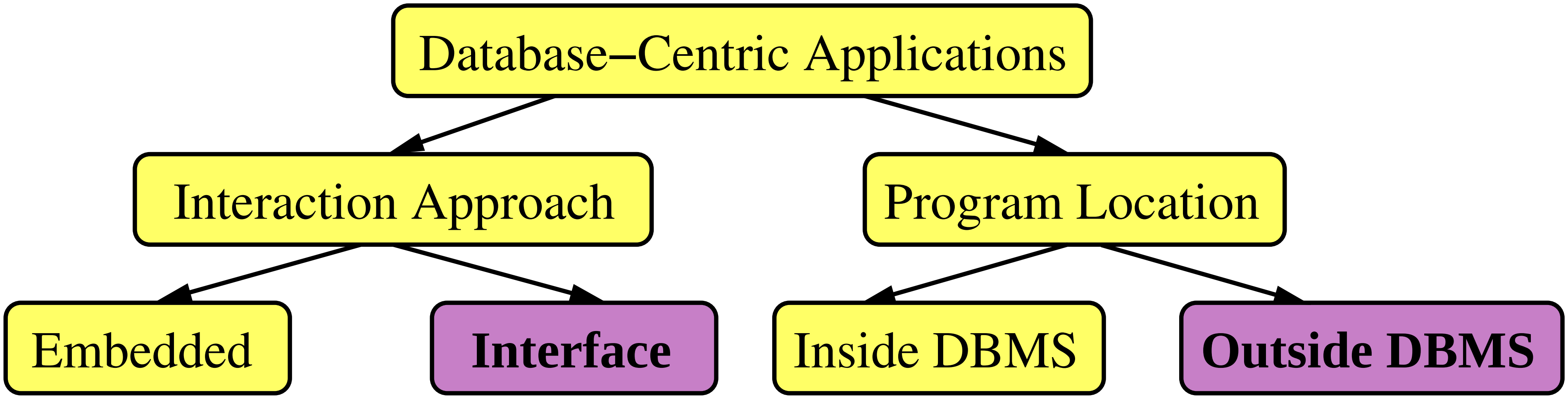
- Defects can also exist in P's interaction with its environment

Focus on Database Interactions



- Program P can view and/or modify the state of the database

Types of Applications

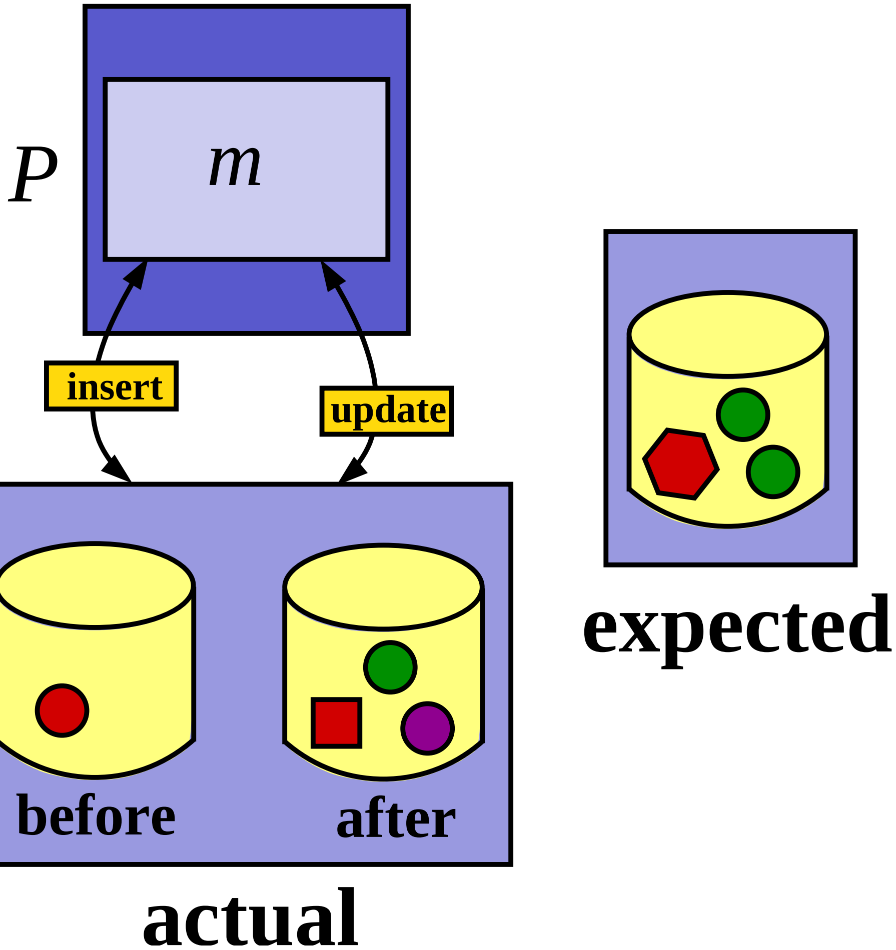


- Testing framework relevant to all types of applications
- Current tool support focuses on Interface-Outside applications
- **Example:** Java application that submits SQL `String`s to HSQLDB relational database using JDBC drivers

Research Contributions

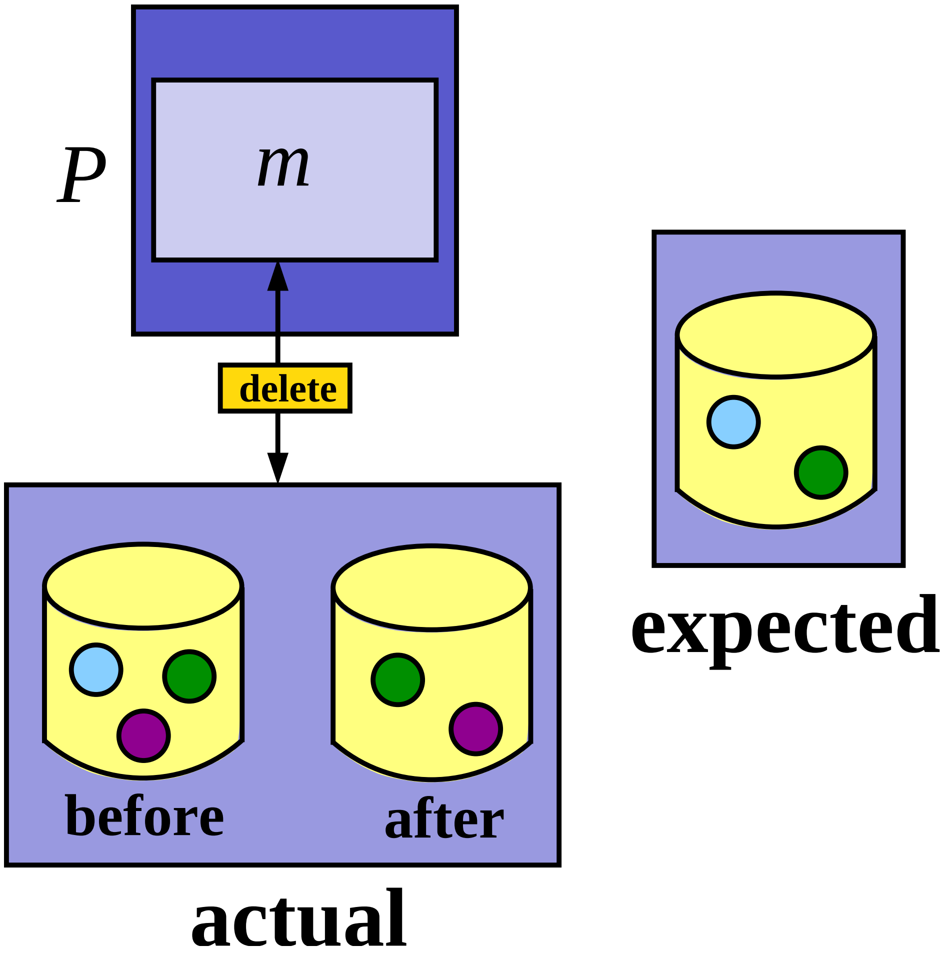
- $\boxed{\text{Database Interaction Fault Model}}$
- Test Adequacy Criteria
- Test Coverage Monitoring
- Regression Testing
  - Reduction
  - Prioritization

Database Interaction Faults: (1-v)



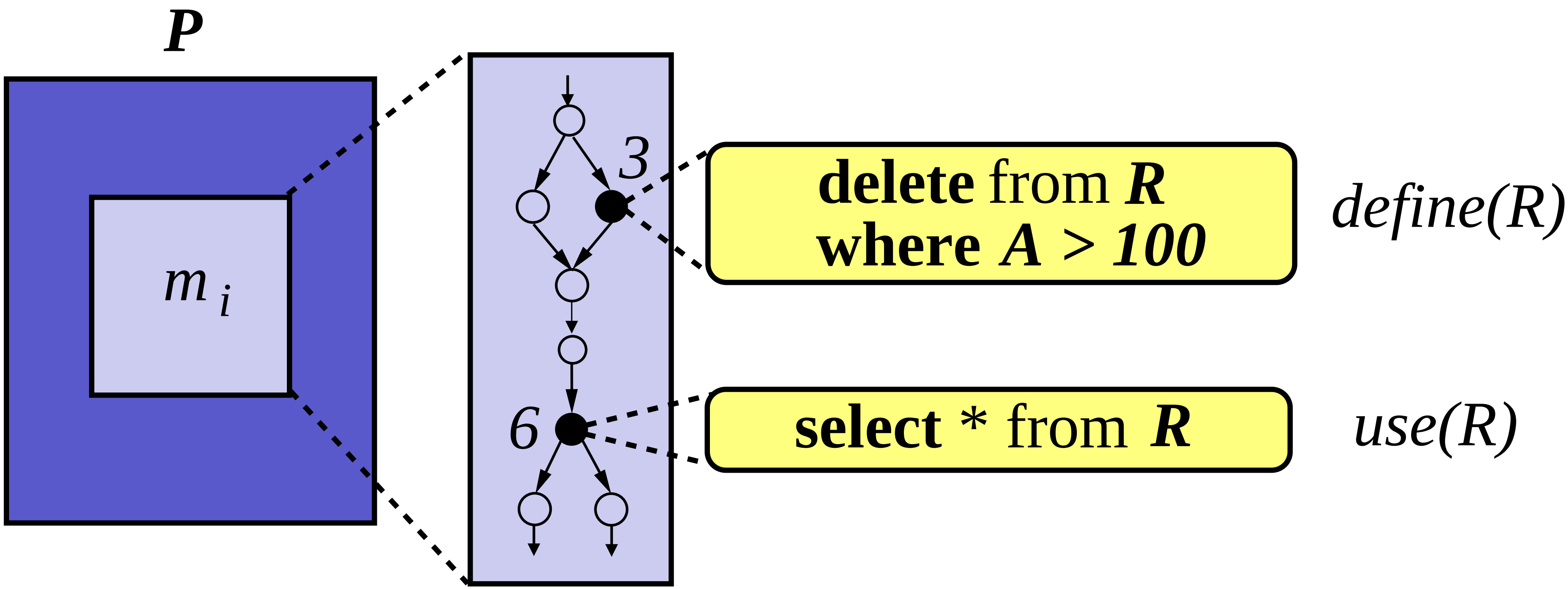
- P uses **update** or **insert** to incorrectly modify items within database
- Commission fault that violates database validity
- Database-aware adequacy criteria can support fault isolation

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

Data Flow-Based Test Adequacy

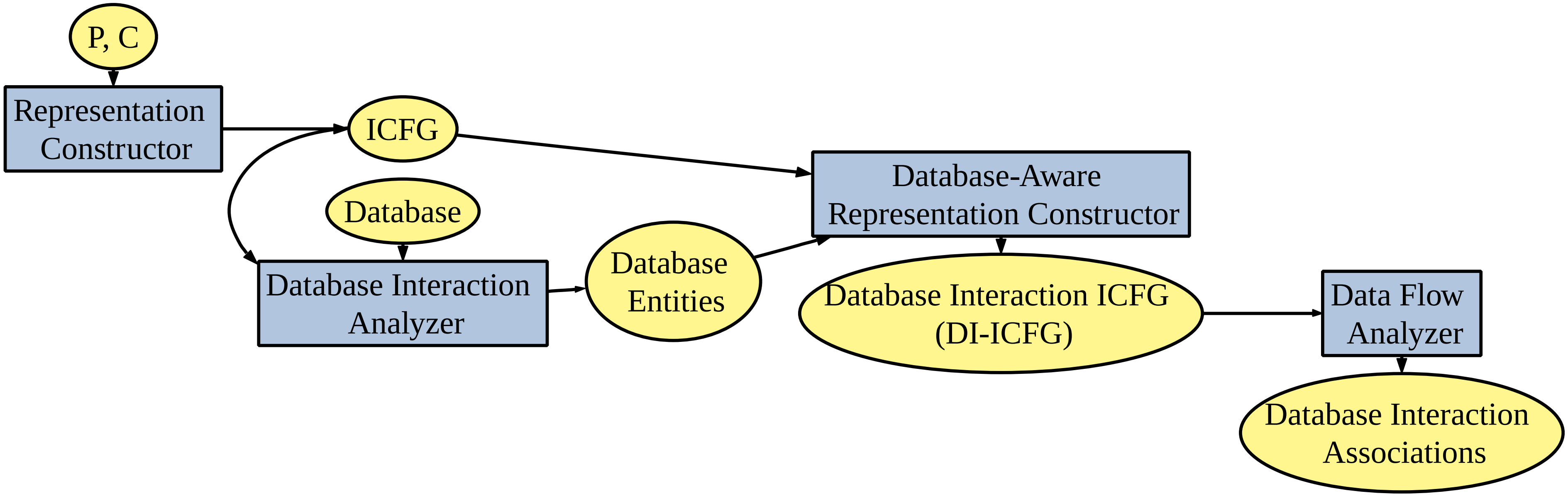


- The intraprocedural database interaction association $\langle n_3, n_6, R \rangle$ exists within method m_i

Research Contributions

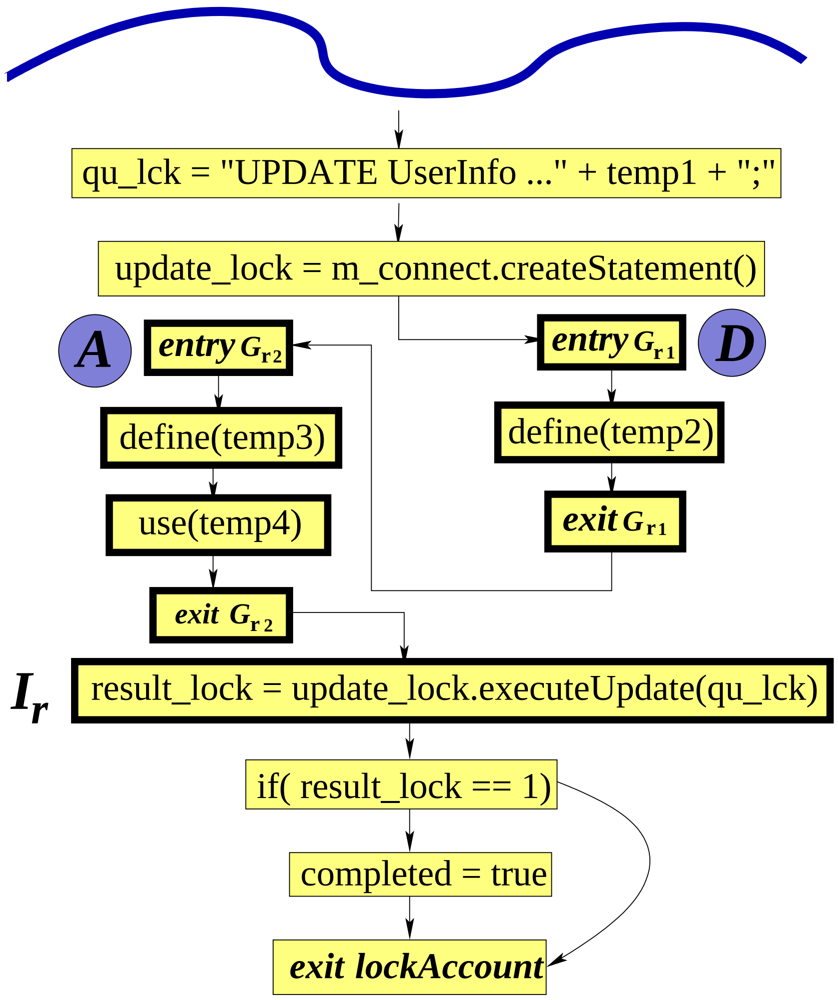
- Database Interaction Fault Model
- Test Adequacy Criteria
- Test Coverage Monitoring
- Regression Testing
  - Reduction
  - Prioritization

Test Adequacy Component



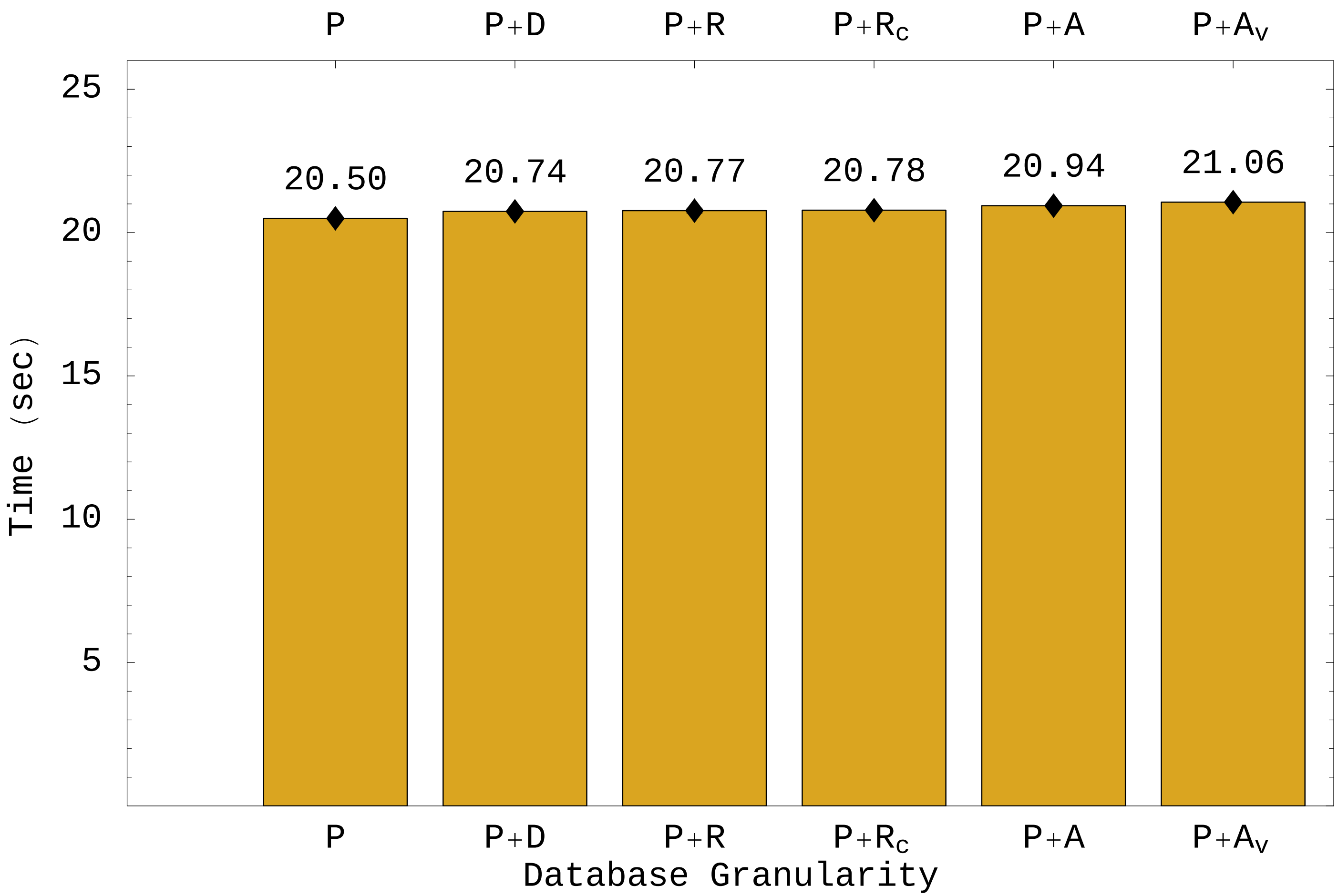
- **Process**: Create a database-aware representation and perform data flow analysis
- **Purpose**: Identify the database interaction associations (i.e., the test requirements)

Database-Aware Representation



- 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

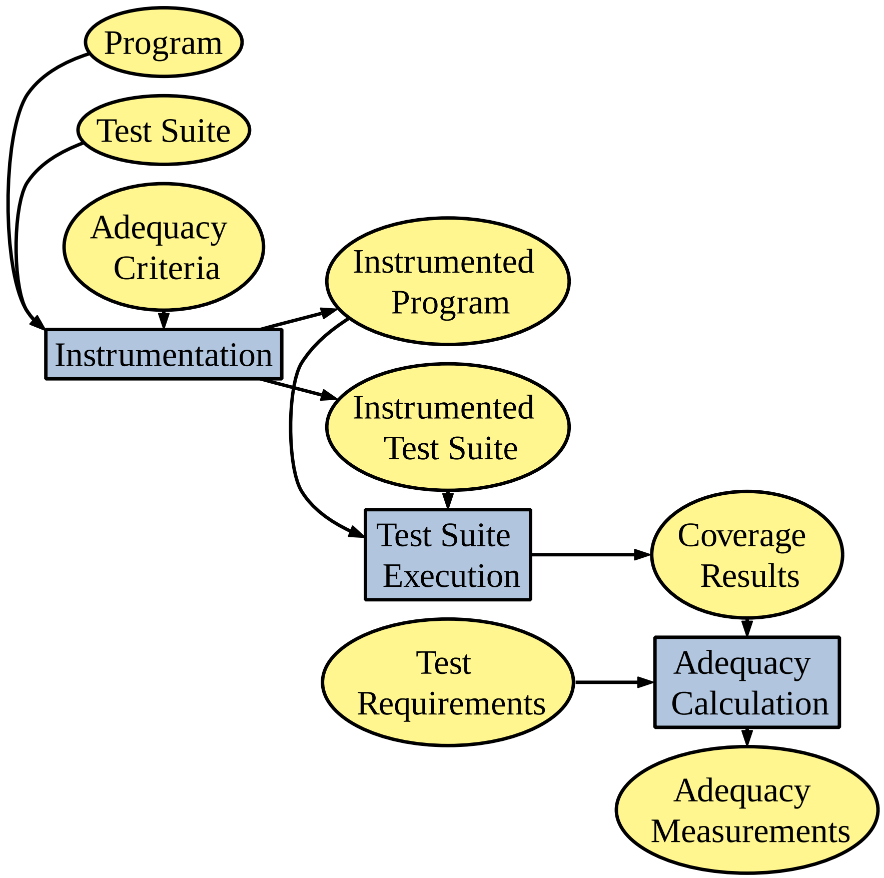


- 2.7% increase in time overhead from P to $P + A_v$ (TM)

Research Contributions

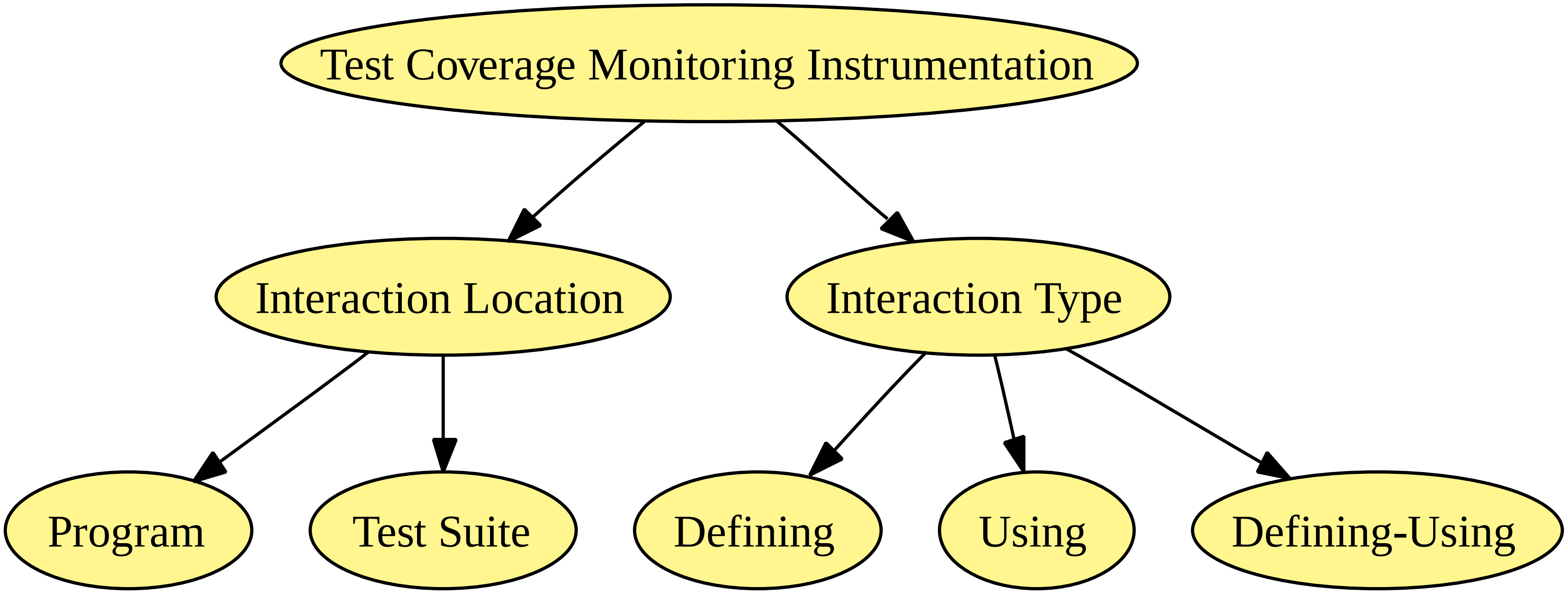
- Database Interaction Fault Model
- Test Adequacy Criteria
- Test Coverage Monitoring
- Regression Testing
  - Reduction
  - Prioritization

Database-Aware Coverage Monitoring



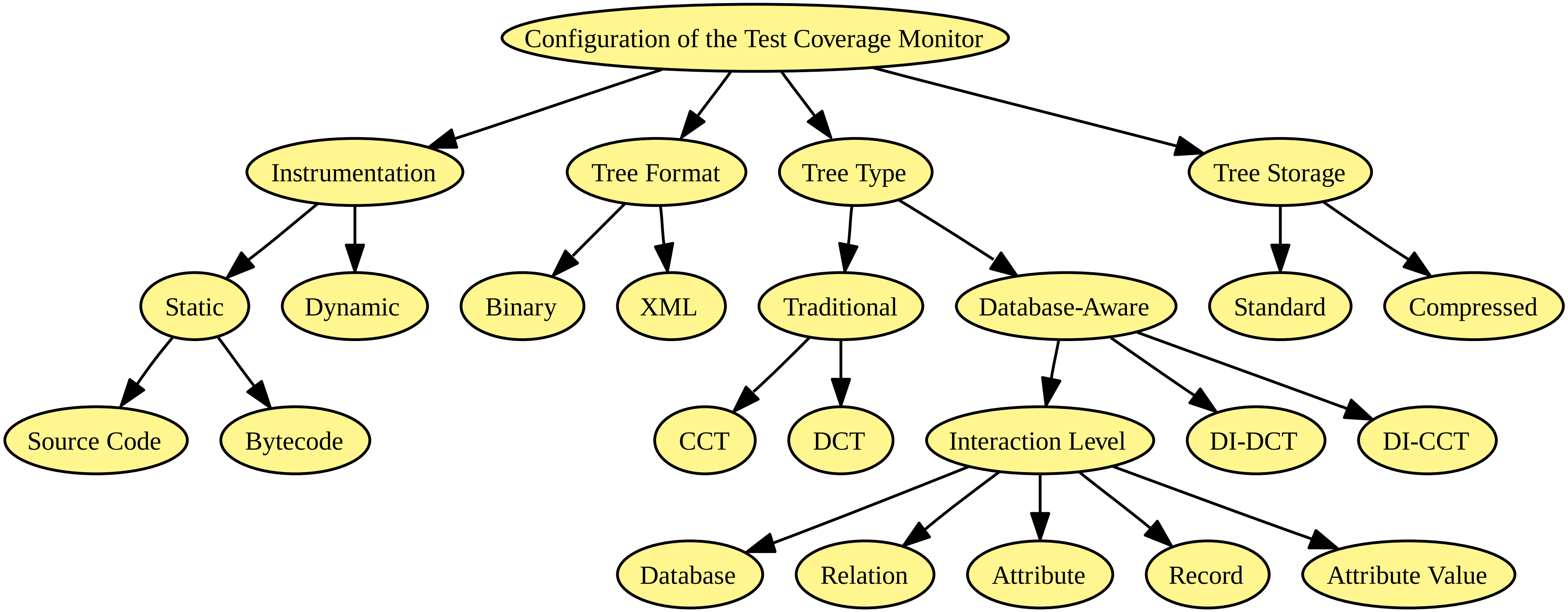
- **Purpose**: Record how the program interacts with the database during test suite execution

Database-Aware Instrumentation



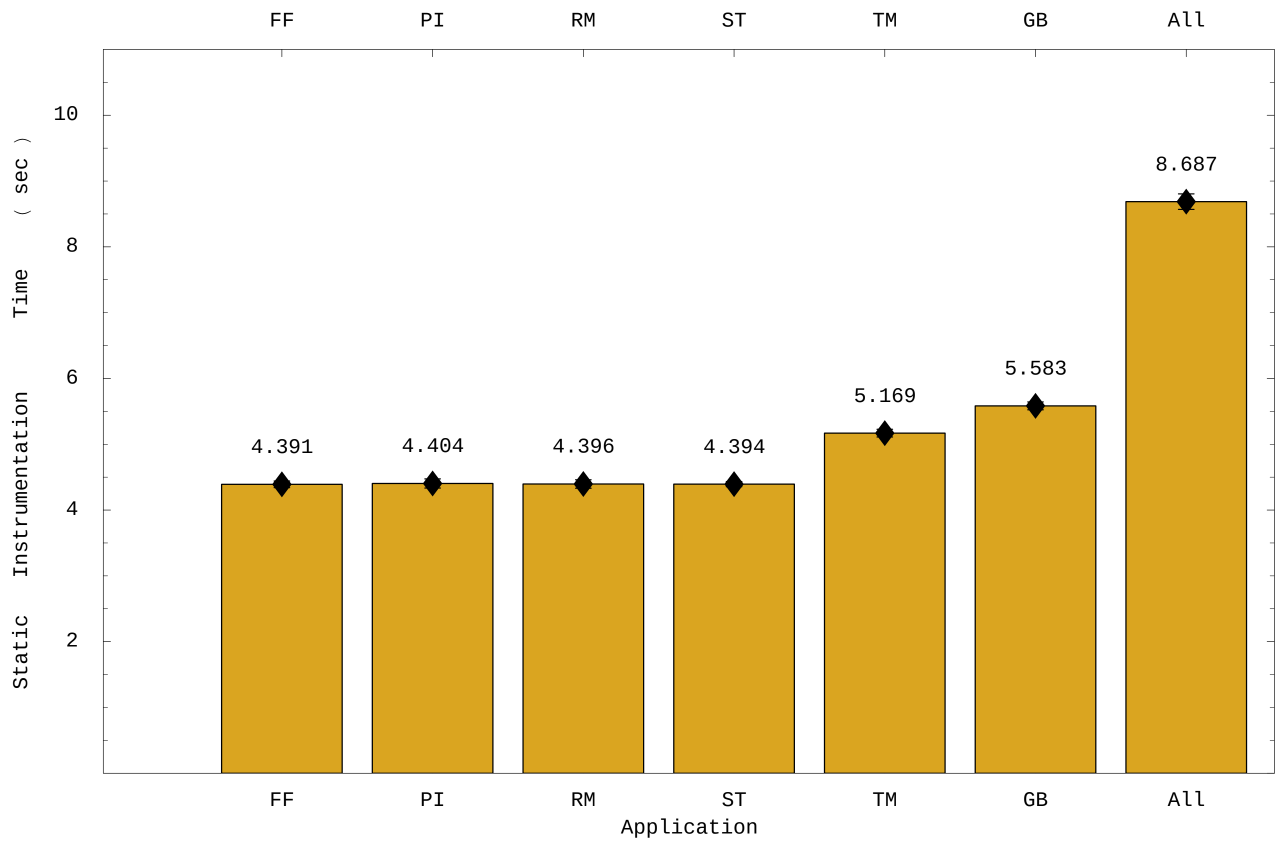
- Efficiently monitor coverage without changing the behavior of the program under test
- Record coverage information in a database interaction calling context tree (DI-CCT)

Configuring the Coverage Monitor



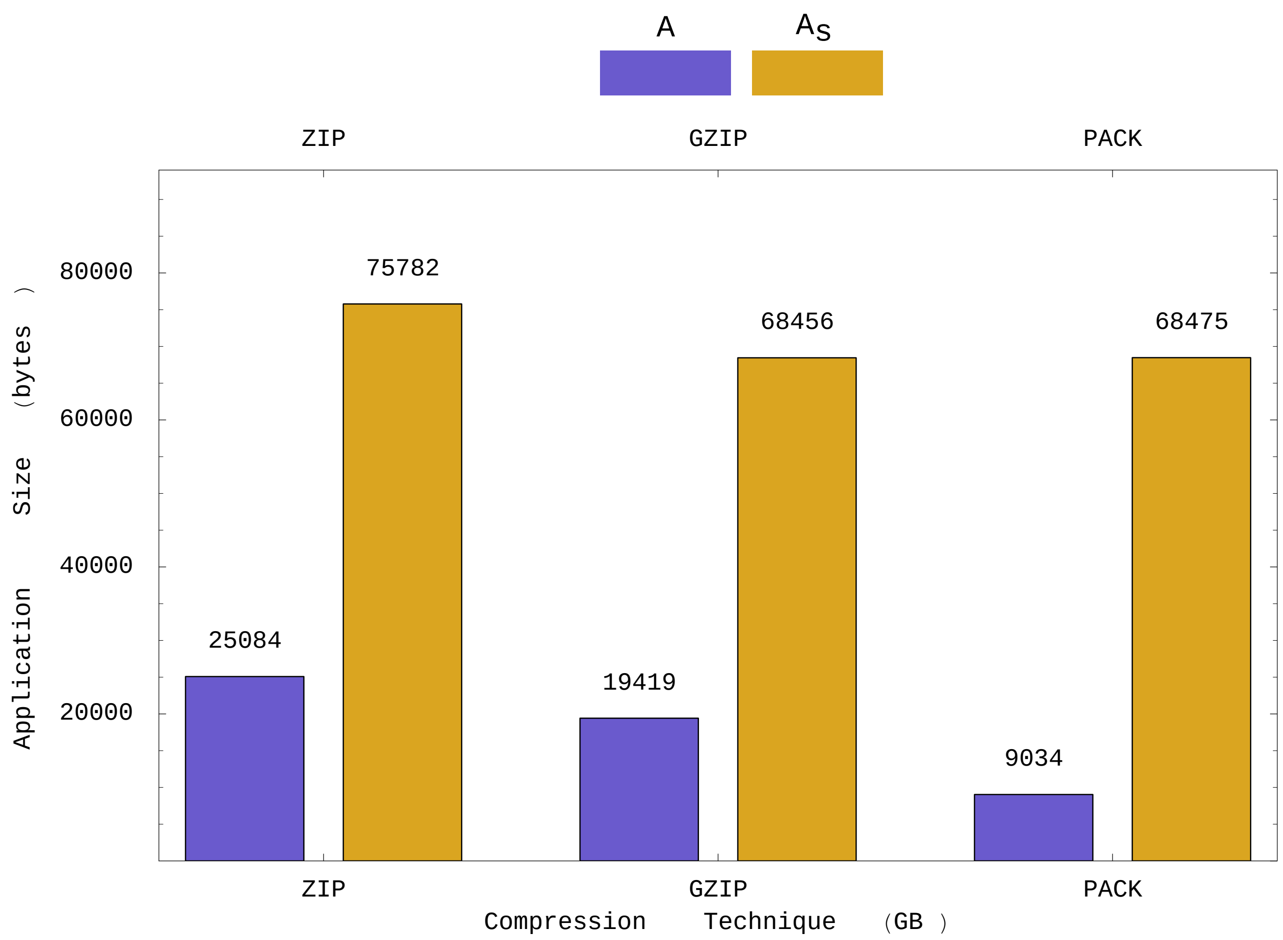
- Flexible and efficient approach that fully supports both traditional and database-centric applications

Static Instrumentation: Time



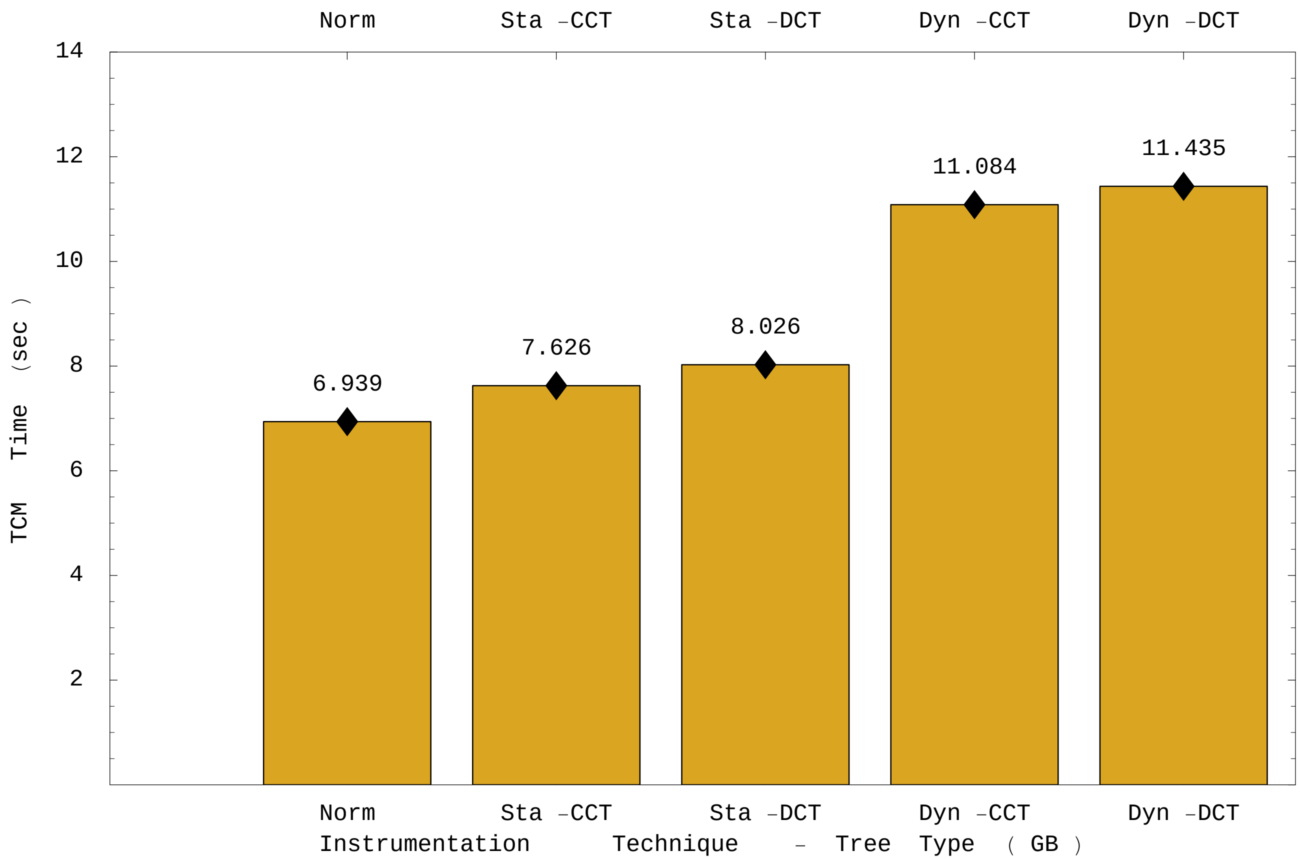
- Attach probes to all of the applications in less than nine seconds
- Static approach is less flexible than dynamic instrumentation

Static Instrumentation: Space



- Increase in bytecode size may be large (space vs. time trade-off)

Static vs. Dynamic Instrumentation



- Static is faster than dynamic / CCT is faster than DCT
- The coverage monitor is both efficient and effective

Size of the Instrumented Applications

Compr Tech	Before Instr (bytes)	After Instr (bytes)
None	29275	887609
Zip	15623	41351
Gzip	10624	35594
Pack	5699	34497

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

Database Interaction Levels

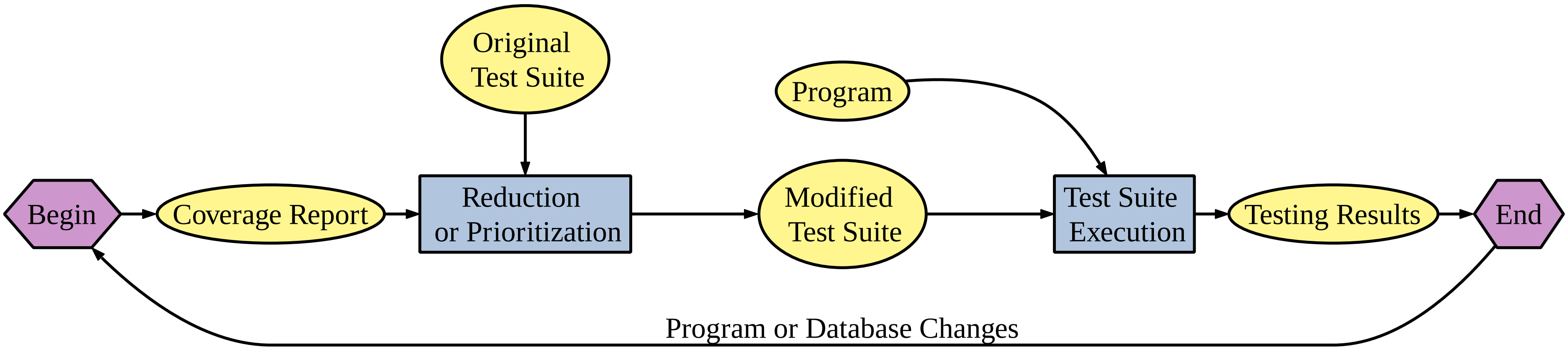
CCT Interaction Level	TCM Time (sec)	Percent Increase ($\%$)
Program	7.44	12.39
Database	7.51	13.44
Relation	7.56	14.20
Attribute	8.91	34.59
Record	8.90	34.44
Attribute Value	10.14	53.17

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

Research Contributions

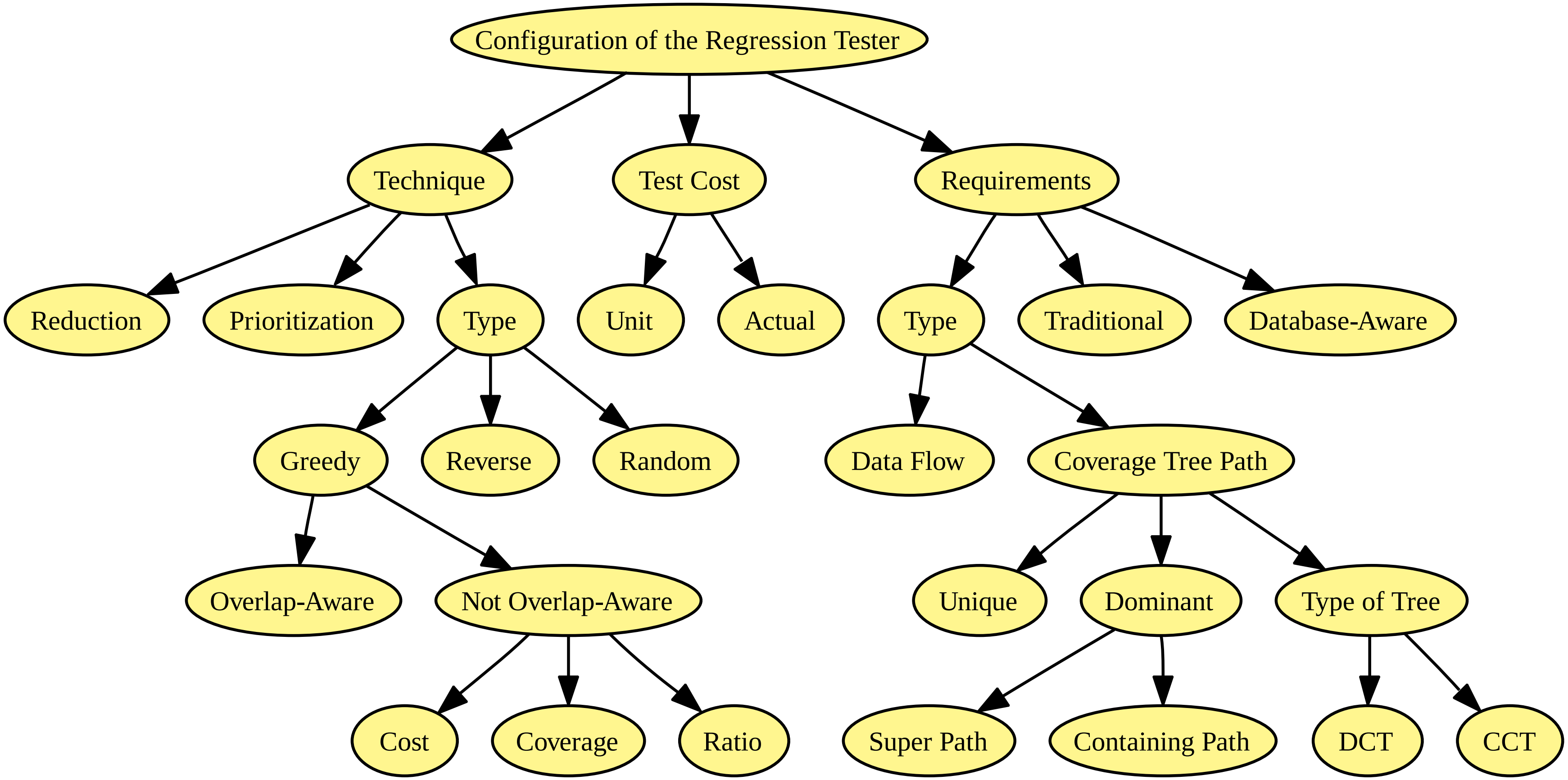
- Database Interaction Fault Model
- Test Adequacy Criteria
- Test Coverage Monitoring
- Regression Testing
  - Reduction
  - Prioritization

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

Configuring the Regression Tester

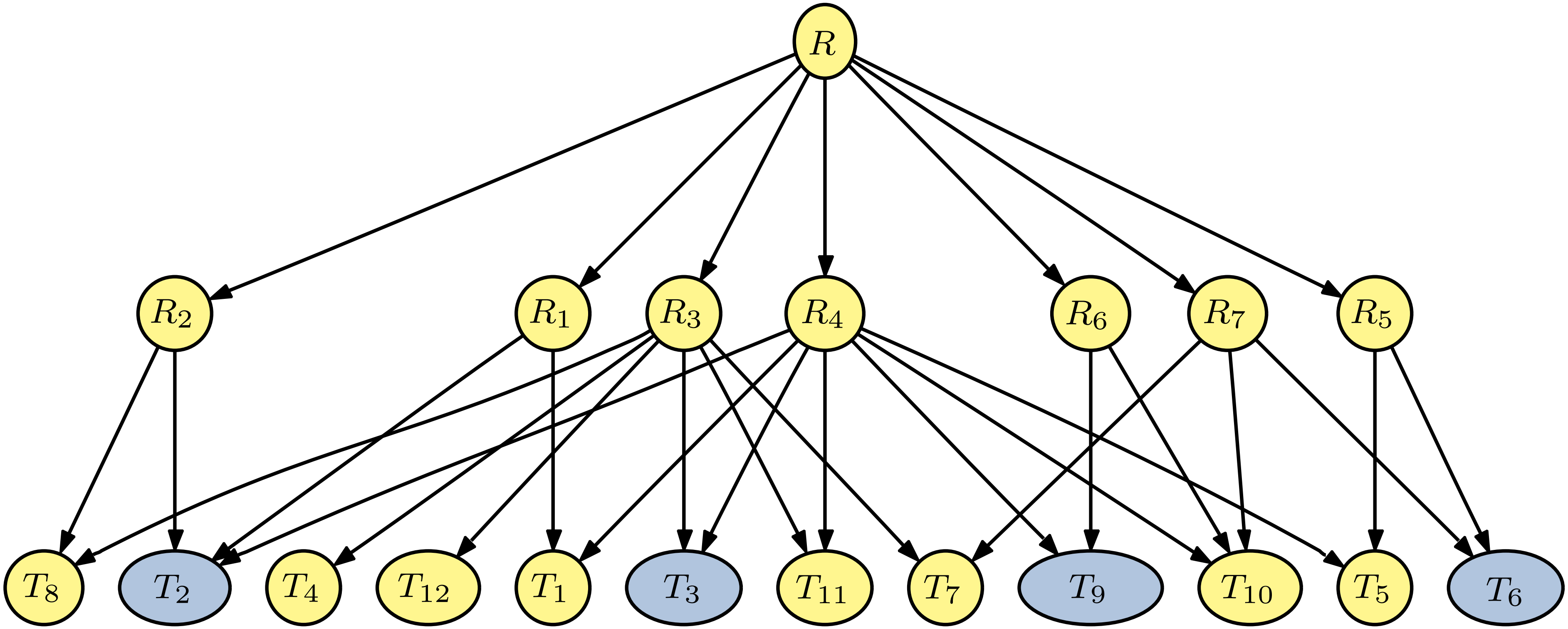


- Regression testing techniques can be used in the *version specific* model

Research Contributions

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

Finding the Overlap in Coverage



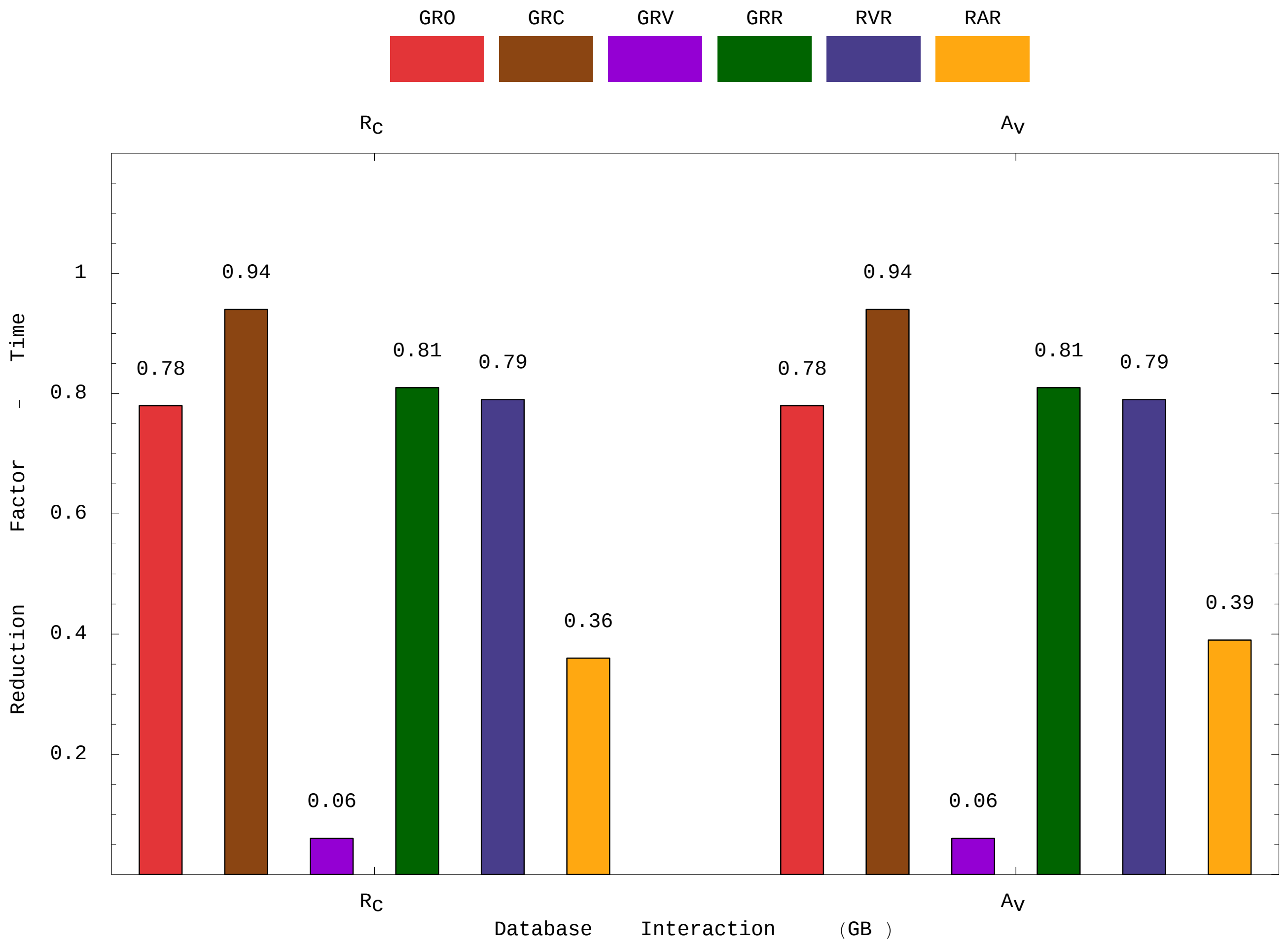
- $R_j \rightarrow T_i$ means that requirement R_j is *covered by* test T_i
- $T = \langle T_2, T_3, T_6, T_9 \rangle$ cover all of the test requirements

Reducing the Size of the Test Suite

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

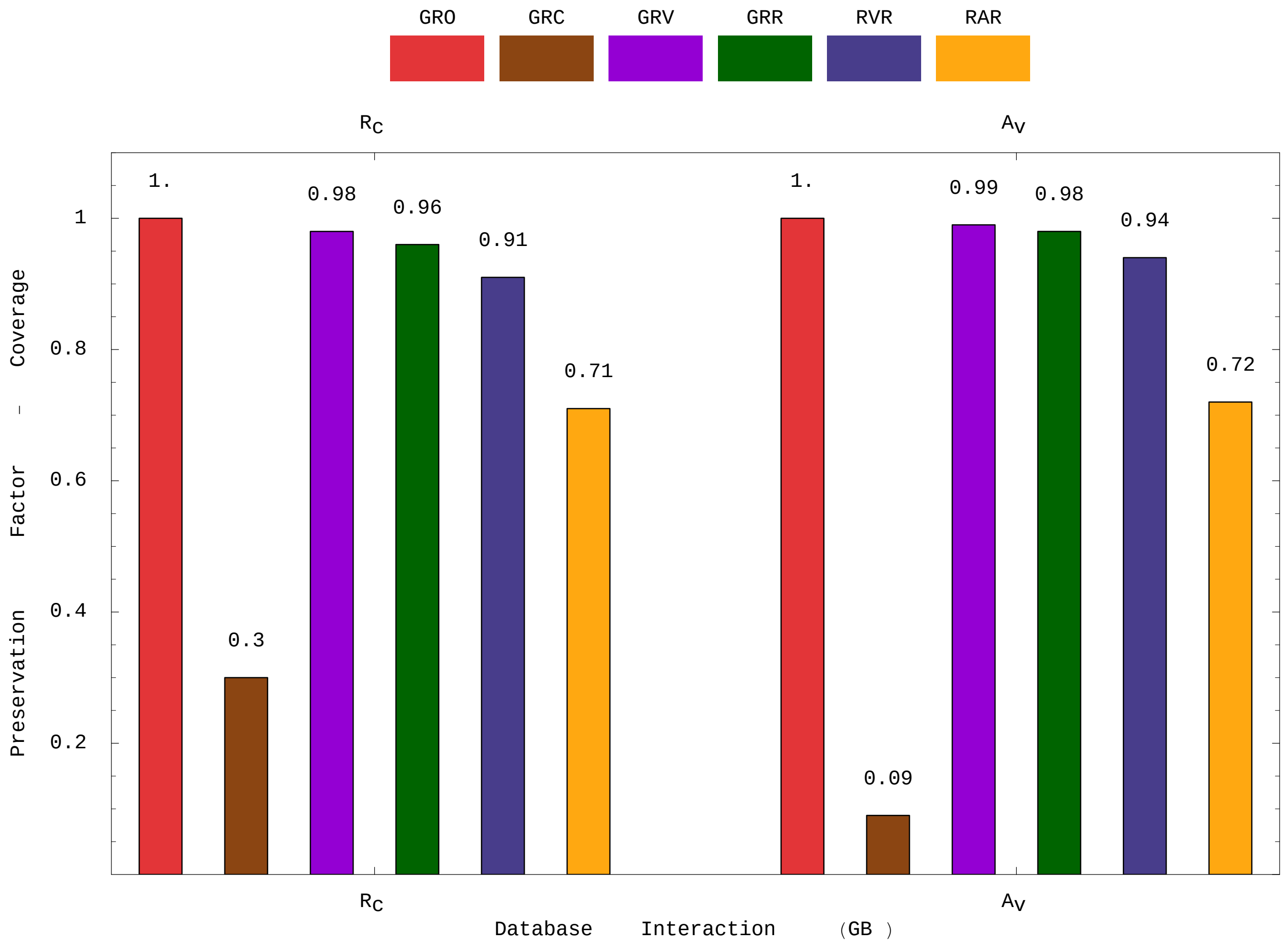
- Reduction factor for test suite size varies from $.352$ to $.8$

Reducing the Testing Time



- GRO reduces test execution time even though it removes few tests

Preserving Requirement Coverage

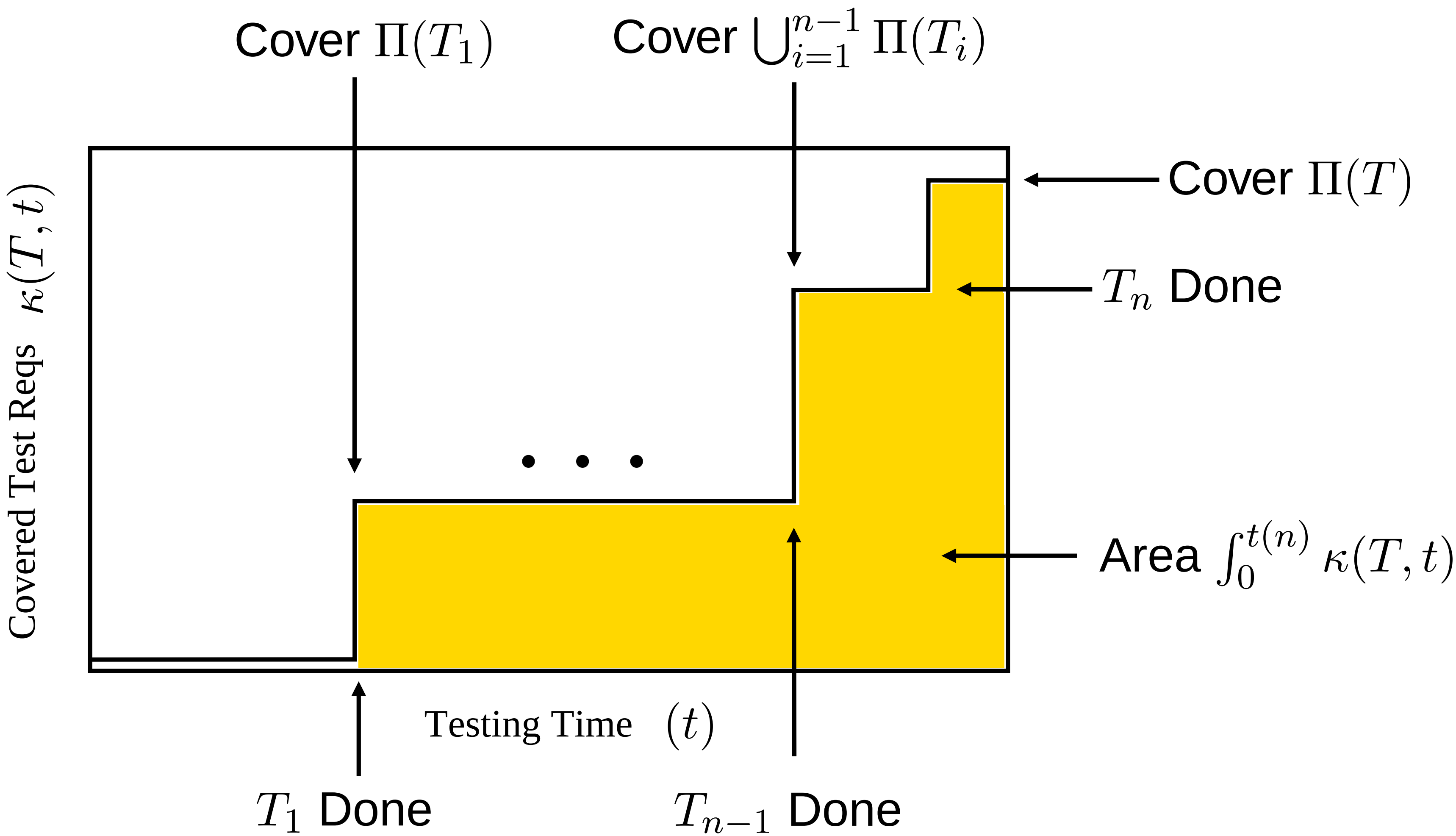


- GRO guarantees coverage preservation - others do not

Research Contributions

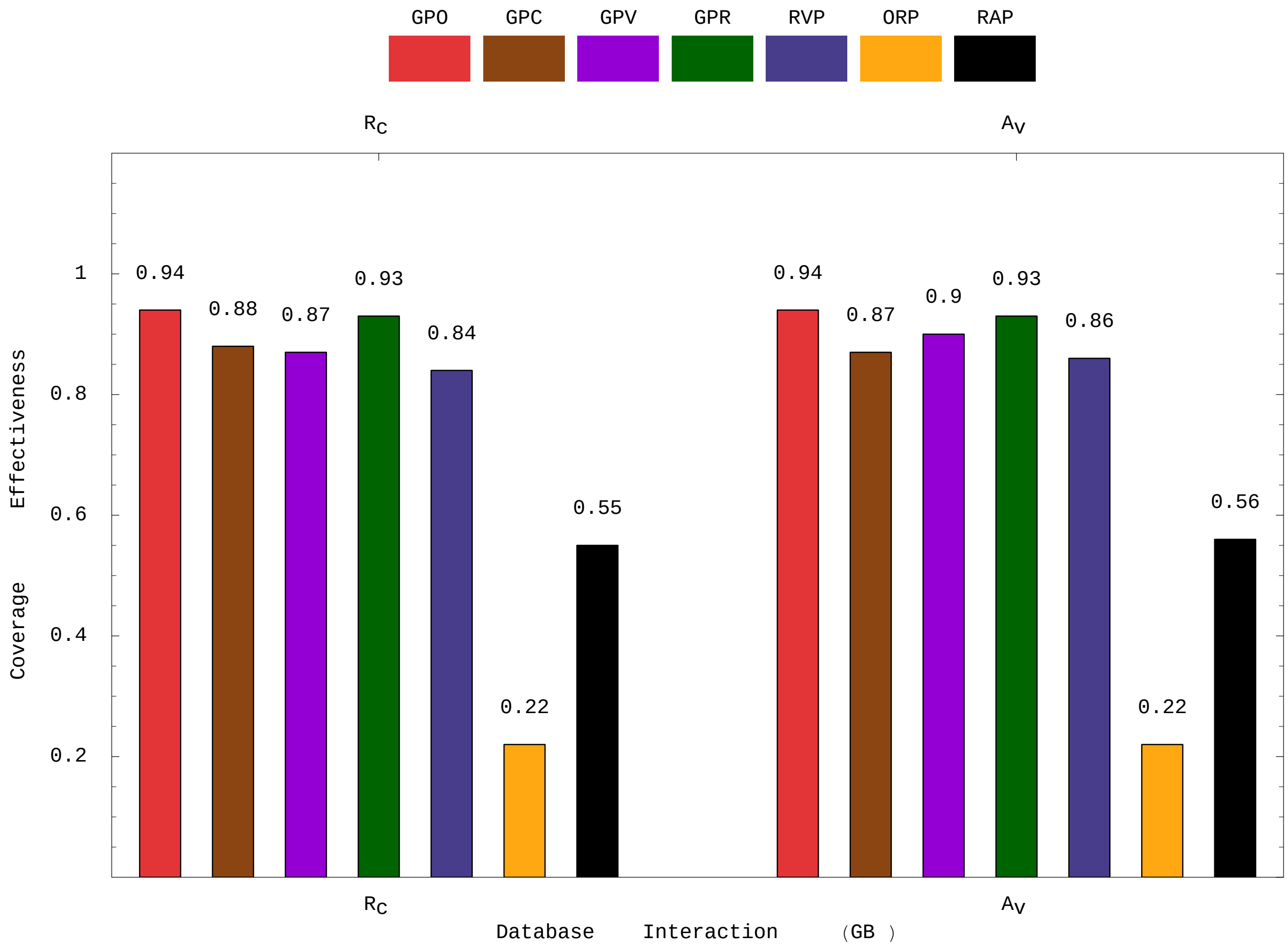
- Database Interaction Fault Model
- Test Adequacy Criteria
- Test Coverage Monitoring
- Regression Testing
  - Reduction
  - Prioritization

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

Gregory M. Kapfhammer and Mary Lou Soffa. A Family of Test Adequacy Criteria for Database-Driven Applications. In *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 2003. (Distinguished Paper Award)

Gregory M. Kapfhammer. *The Computer Science Handbook*, chapter 105: Software Testing. CRC Press, Boca Raton, FL, Second Edition, June 2004.

Gregory M. Kapfhammer, Mary Lou Soffa, and Daniel Mossé. Testing in Resource-Constrained Execution Environments. In *Proceedings of the 20th ACM/IEEE International Conference on Automated Software Engineering*, Long Beach, California, November, 2005

Kristen R. Walcott, Mary Lou Soffa, Gregory M. Kapfhammer, and Robert S. Roos. Time-Aware Test Suite Prioritization. In *Proceedings of the ACM SIGSOFT/SIGPLAN International Symposium on Software Testing and Analysis*, Portland, Maine, June, 2006.

Gregory M. Kapfhammer and Mary Lou Soffa. A Test Adequacy Framework with Database Interaction Awareness. *ACM Transactions on Software Engineering and Methodology*. Invited Paper Under Review.