

Unit Testing Database Applications

Claus A. Christensen

Steen Gundersborg

Kristian de Linde

Kristian Torp

`{cac, earser, kdl, torp}@cs.aau.dk`

Department of Computer Science

Aalborg University

&

Logimatic Software A/S

Outline - Overall

- Testing in general
- Unit testing
 - The *xUnit* family of unit test frameworks
- Database applications
 - A table wrapper example
- Unit testing database applications
 - What is wrong with the current unit test frameworks
 - The *DBMSUnit* test framework
- Experiences using DBMSUnit
 - Based on experiences from Logimatic Software A/S
- Finding more information on software testing
- Conclusions
 - Future work

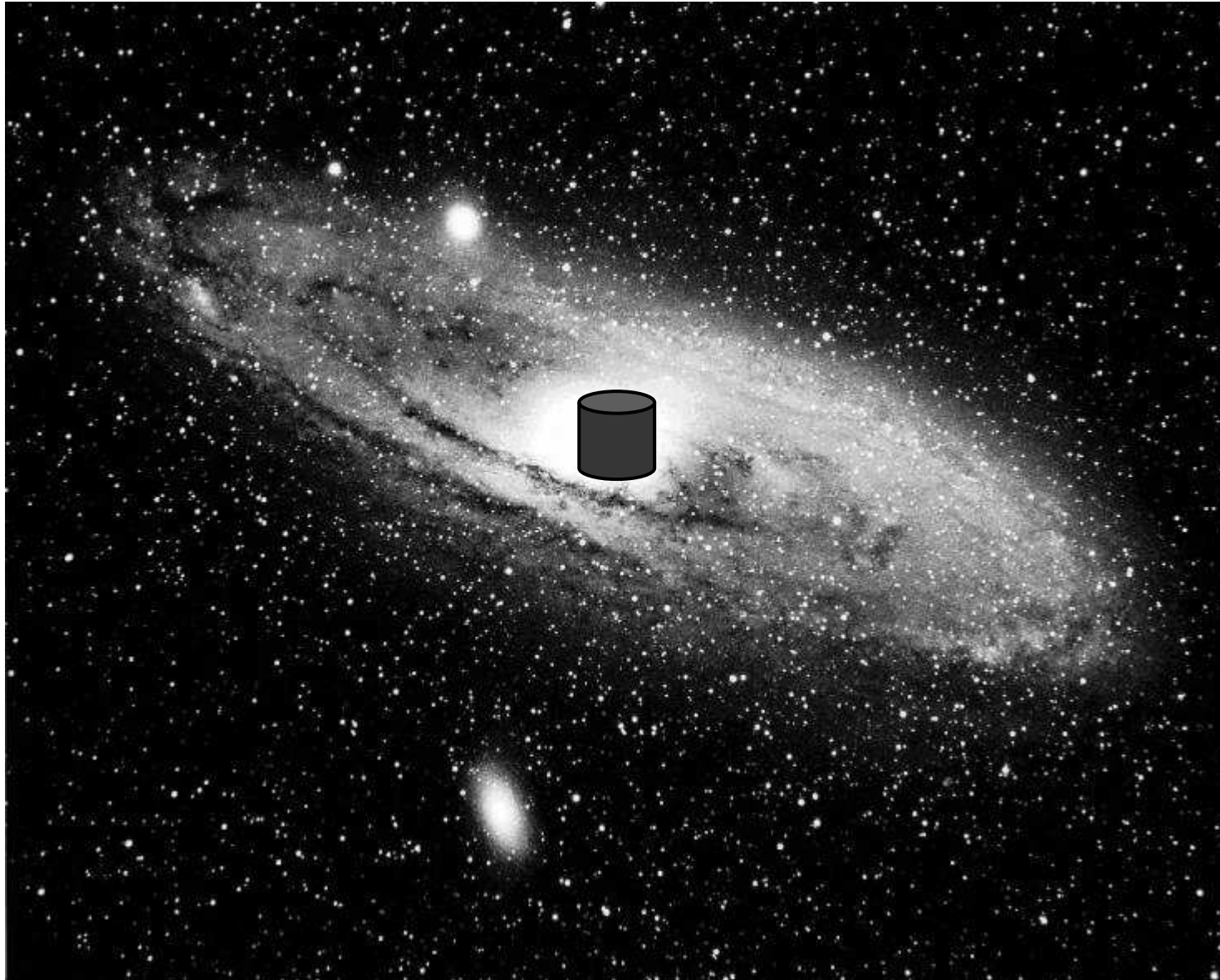
About Myself

- Education
 - Master in computer science (database systems) 1994
 - Ph.D. in computer science (database systems) 1998
- Work four years in software industry
 - Three years at Logimatic Software A/S
- Currently associated professor, Database and Programming Systems Unit, Aalborg University
- Research interests
 - Programmatic access to databases
 - XML and databases
 - Moving objects and databases

My Road to Working with Software Test

- By introducing big enough errors in released software
- Trying to increase quality of work
- Software developed may contain severe errors
 - Missing sleep and bad nerves
 - Unsatisfied customers
- Unable to understand existing code
 - Written by morons ☺
- Questions from management
 - Does it run? Is it error-free?
- Afraid to make changes to existing code
- Nice emails from the open source community

Disclaimer



Outline - Testing in General

- Motivation
- Purpose of test
- Types of tests

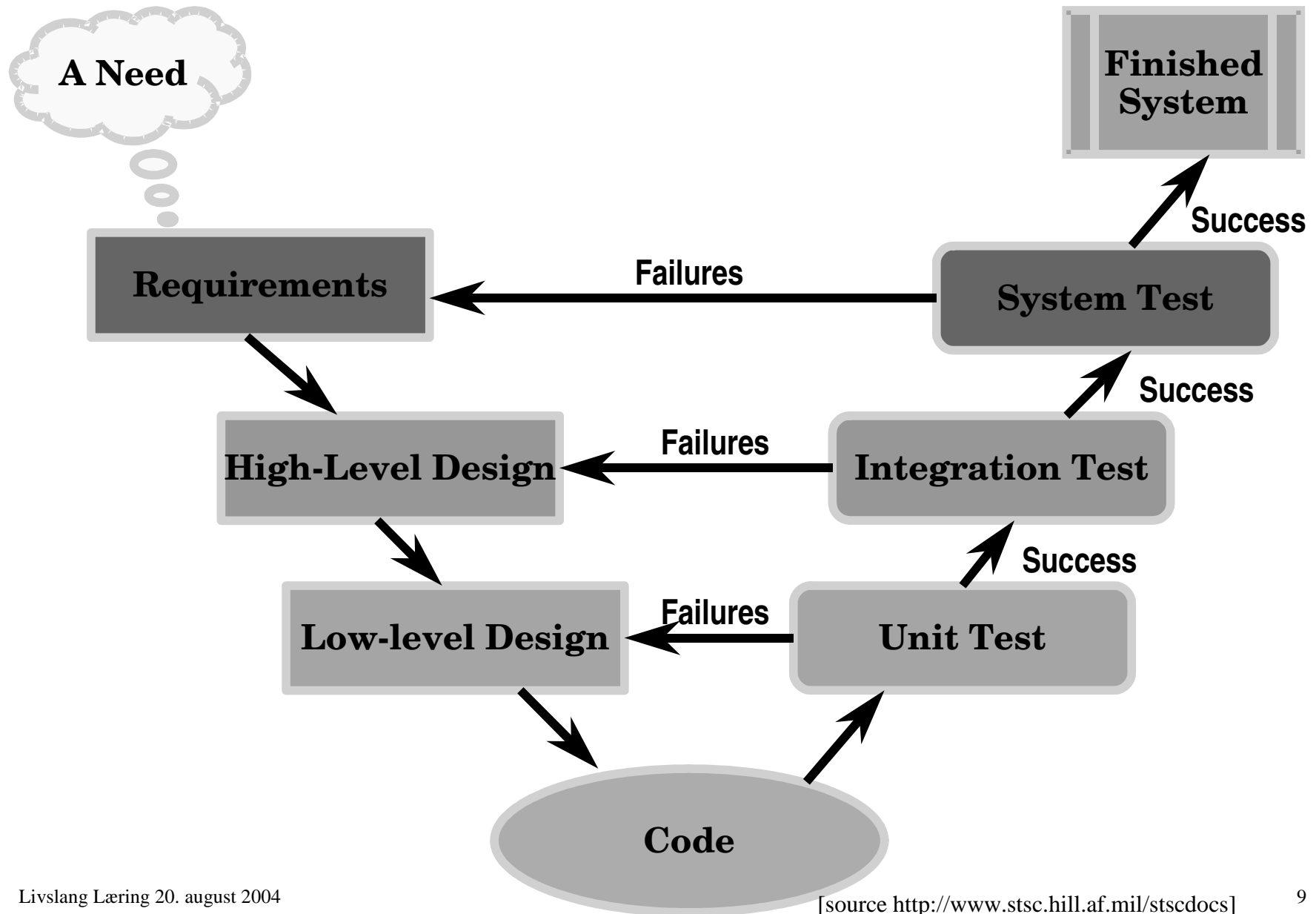
Why Focus on Test?

- *”The importance of software testing and its implications with respect to software quality cannot be overemphasized” M. Deutch*
- *“Error removal is the most time-consuming phase of a [software] life cycle” R. Glass.*
 - 20% analysis, 20% design, 20% implementation, 40% test
- Current available positions at Microsoft (2004-08-19)
 - “Software development” 556
 - “Software testing” 304

Test Purpose

- Test of performance
 - TPC*C, TPC*H, TPC*R, and TPC*W (www.tpc.org)
 - Home-grown benchmarks
 - ◆ 90% all user requests finished within a minute
- Test of design or idea
 - Prototyping
- Test of correctness
 - Formal verification
 - Code reviews and walk-throughs
 - Compile-and-run
 - Stochastic testing (n-solutions)
 - Unit testing

Unit, Integration, and System Tests



Unit, Integration, and System Tests, cont.

- System Test
 - Testing overall requirements to the system
 - “Does the system works as a whole?”
- Integration Test
 - Testing of interfaces between subsystems
 - “How well do the subsystems fit together?”
- Unit test
 - Testing a single unit (or module) of code
 - Uses stubs or drivers for interfaces to other units
 - “Is the foundation working?”

Outline - Unit Testing

- Motivation for
 - unit testing
 - JUnit
- The xUnit family
- The JUnit test framework
 - A framework for Java
- Extensions to JUnit

Motivation for Unit Testing

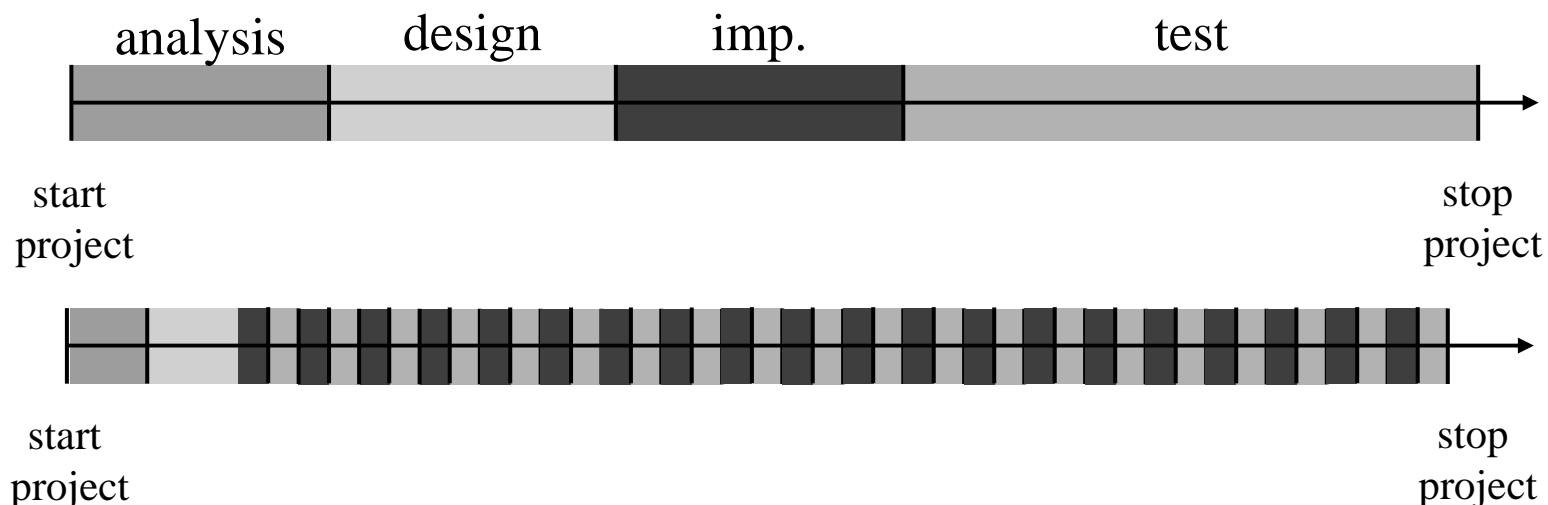
- The single most effective technique to better code!
- Very simple [and cheap] to get started
 - Return On Investment very high
- Test code very informative to read
 - Internal and informal documentation
 - End-users typically use code snips in documentation as an outset for development!
- Supports incremental development
 - System can be fully tested frequently, e.g., each night
- Better motivated for writing test cases if these are intensively used.

Motivation JUnit

- It is nearly impossible to do a good job at software error removal without tool support
- Used as model for many other unit test frameworks
- Instant "binary" feedback to developers
- Easy to use
- Widely used (even students use it, under pressure!)
- Open source (and free)
 - There are many extremely expensive commercial products related to software testing
- Integrated into many IDEs
 - Eclipse, Idea, JBuilder, JDeveloper, jEdit, etc.
- Written by very competent people
 - Kent Beck and Eric Gamma

Unit Test Characteristics

- Supports "Code a little, test a little" development



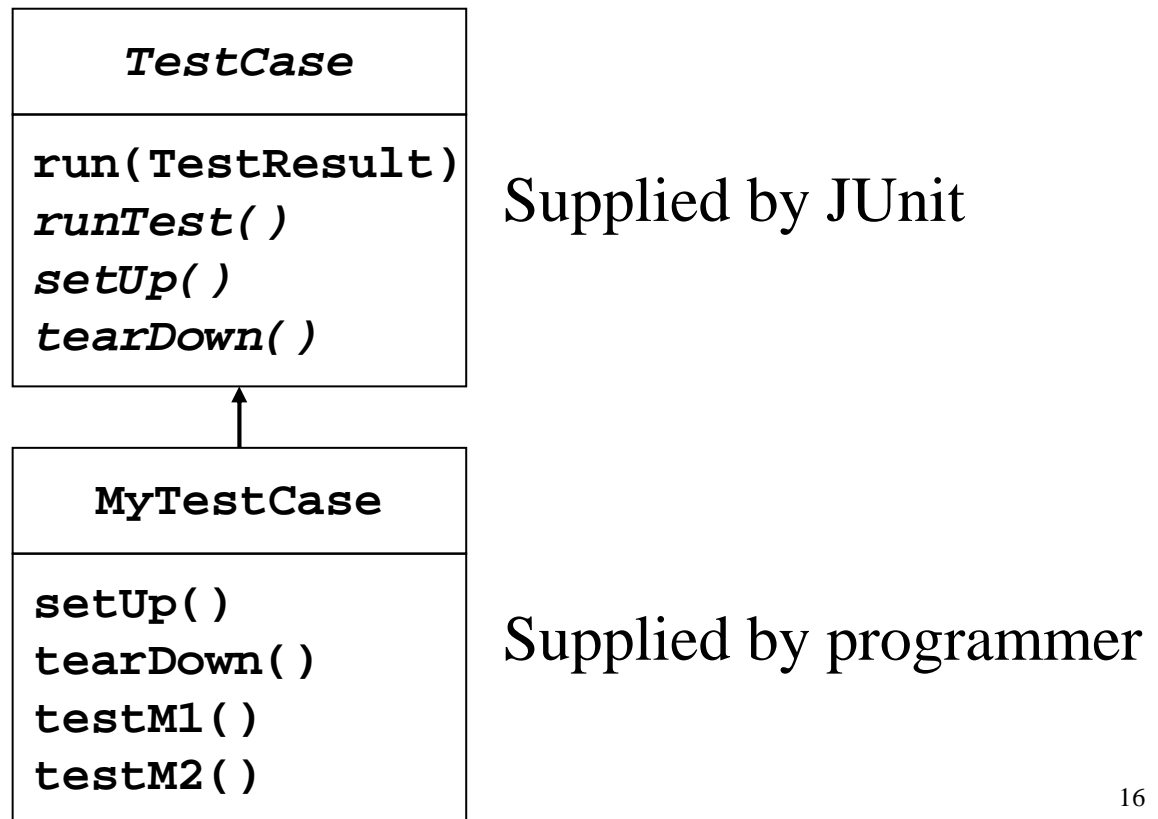
- Done by programmers, for programmers!
- Not a formal validation and verification of a program!

The xUnit Family

- JUnit (Java, master copy) **
 - <http://junit.org/>
- NUnit (all .Net languages)
 - <http://nunit.org>
- PerlUnit (Perl)
 - <http://perlunit.sourceforge.net/>
- PyUnit (Python)
 - <http://sourceforge.net/projects/pyunit/>
- RubyUnit (Ruby)
 - <http://www.ruby-lang.org/>
- utPLSQL (Oracle's PL/SQL) **
 - <http://utplsql.sourceforge.net>
- Unit++ (C++)
 - <http://unitpp.sourceforge.net/>
- VBUnit (Visual Basic, commercial product)
 - <http://www.vbunit.com>

The JUnit Test Framework

- Software framework
 - Inversion of control (“fill in the blanks programming”)
 - ”Do not call us, we will call you!”
- The basic test class is **TestCase**



The JUnit Test Framework, cont

- Three import concepts in JUnit
 - Test suite
 - Test case
 - Test method



The `setUp` and `tearDown` Methods

```
public class TestHardDrive extends TestCase{
    private HardDrive seagate1;
    private HardDrive seagate2;

    /** Sets up the test fixture. */
    protected void setUp(){
        seagate1 = new HardDrive("Seagate", "Deskstar",
                                1000.00, 40);
        seagate2 = new HardDrive("Seagate", "Deskstar",
                                1000.00, 40, Storage.GB);
    }

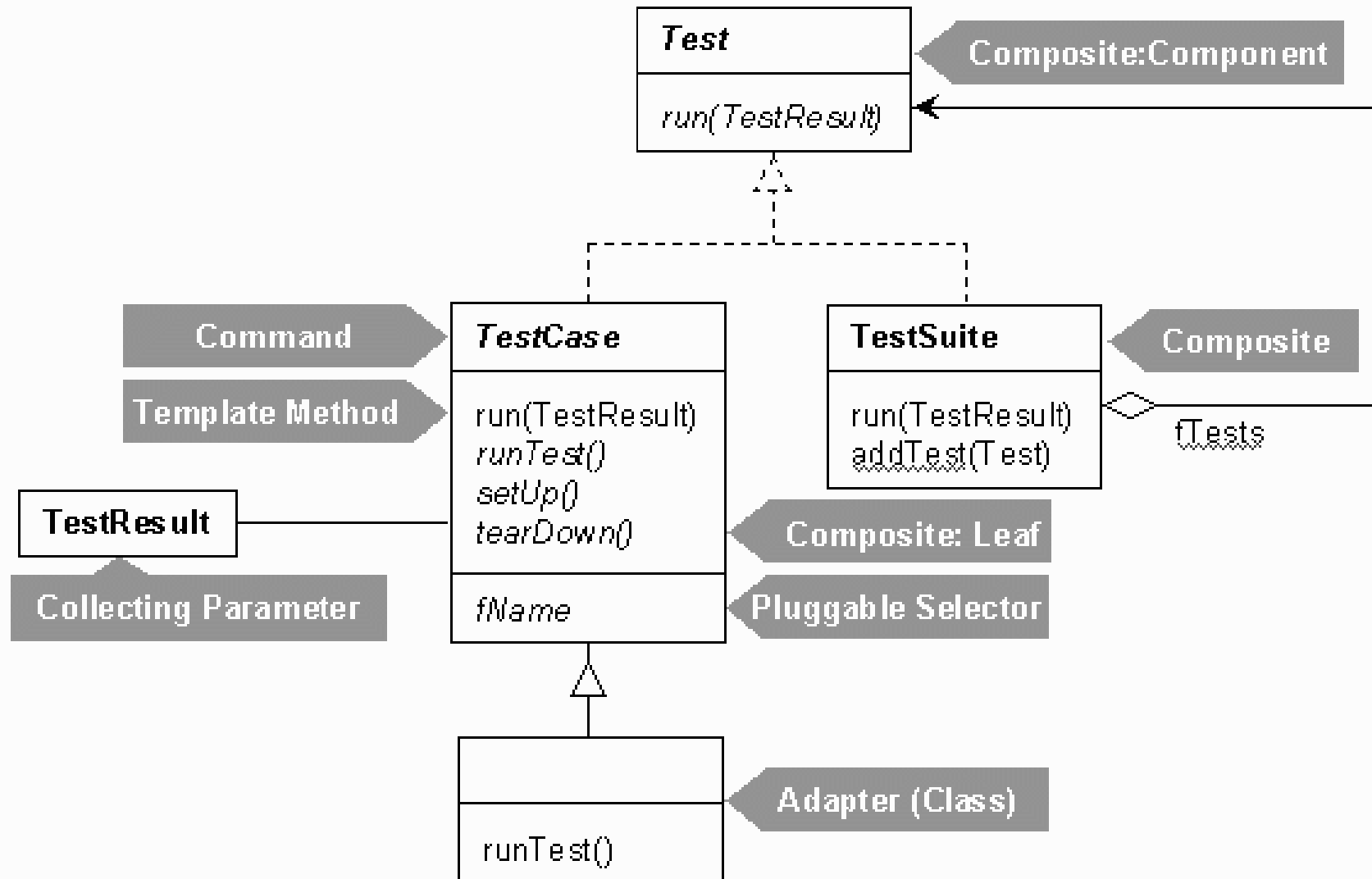
    /** Tear down the test fixture */
    protected void tearDown(){
        // empty
    }
}
```

The Test Methods

```
public class TestHardDrive extends TestCase{
    <snip>
    /** First test method */
    public void testIntel1(){
        assertEquals("Make not equal", seagate1.getMake(),
            "Seagate");
        assertTrue("Price not equal",
            seagate1.getPrice() == 1000.00);
    }

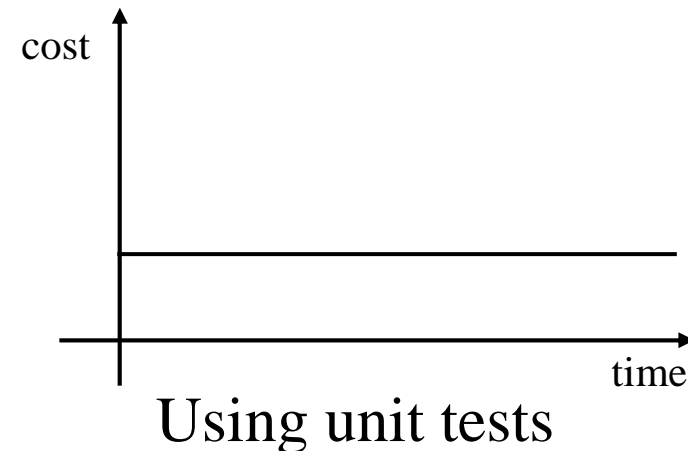
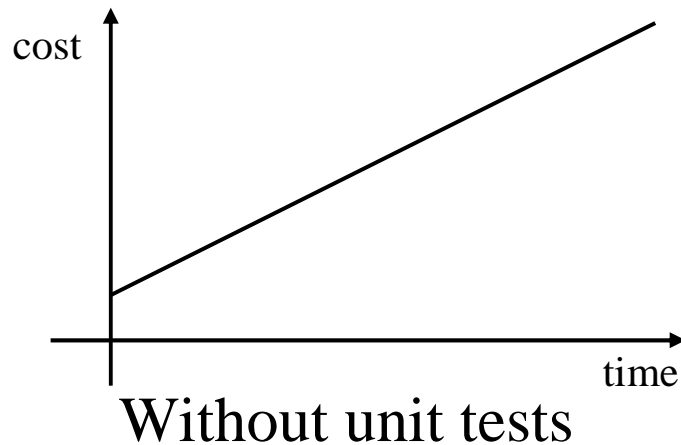
    /** Second test method */
    public void testEquals(){
        assertTrue("HardDrives are not equal",
            seagate1.equals(seagate2));
        <snip>
    }
}
```

Overview JUnit Test Framework



Consequences of using Unit Tests

- Refactoring code becomes much easier



- Have more KLOC test code than "real" code
- Makes your low-level code much more reliable!
 - Solid foundation for integration and system tests
- Decrease the use of a debugger
 - A debugger is a review tool not a test tool!

JUnit Extension

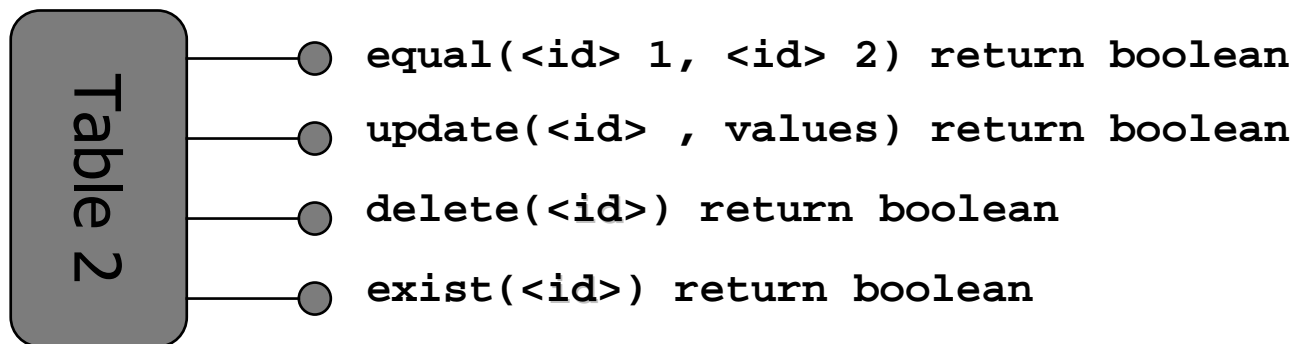
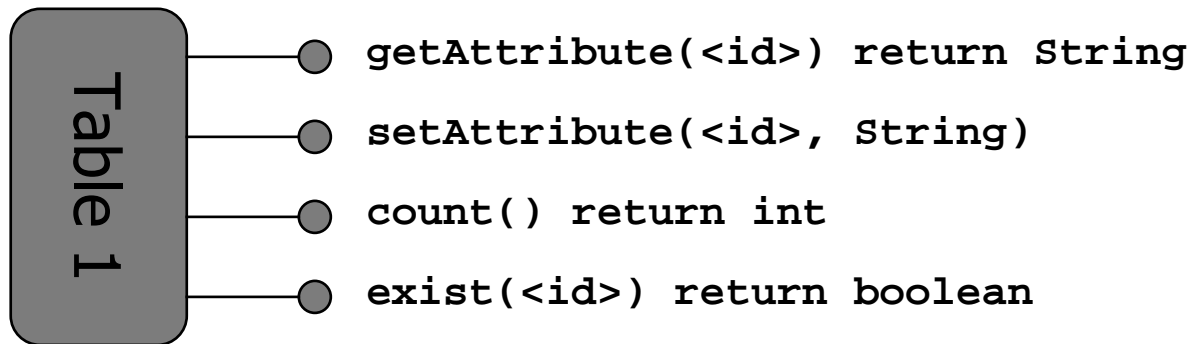
- Test coverage
 - Clover (commercial) <http://www.cenqua.com/clover/>
 - NoUnit (open source) <http://sourceforge.net/projects/nounit/>
- MockObjects
 - MockMaker <http://mockmaker.sourceforge.net>
 - EasyMock <http://www.easymock.org>
- Performance testing
 - JUnitPerf <http://www.clarkware.com/software/JUnitPerf.html>
 - Daedalos <http://www.daedalos.com/EN/djux>
- Database testing
 - DBUnit <http://dbunit.sourceforge.net/>
 - ◆ Import/export oriented. Truncates tables!
- Most extensions use the Decorator Design Pattern

Outline - Database Application

- An example
 - table wrapper API
- Overview of interfaces provided by table wrapper API
- The wrapper in an application stack

A Table Wrapper API

- Features illustrated using Java



Examples of Wrapper Interfaces

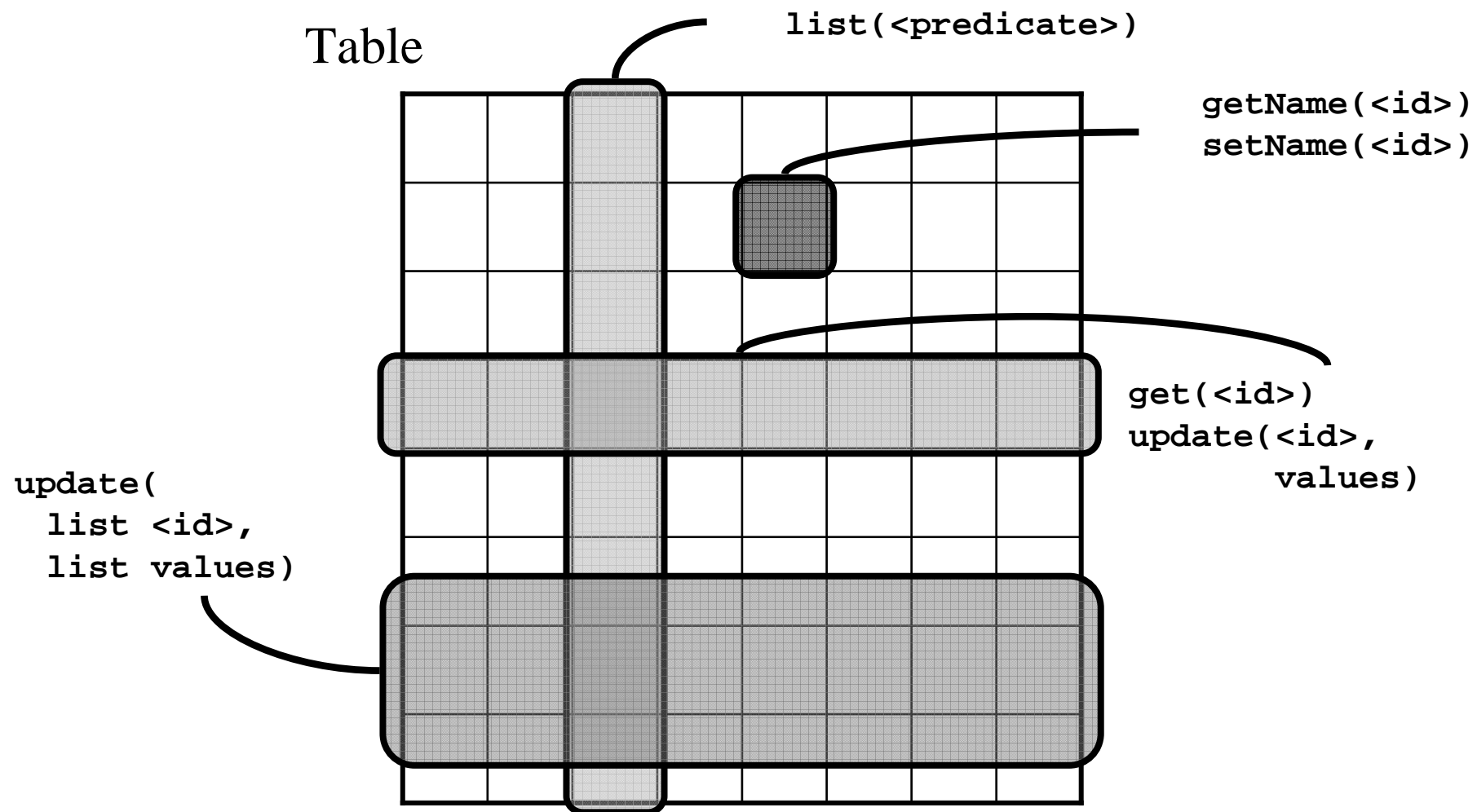
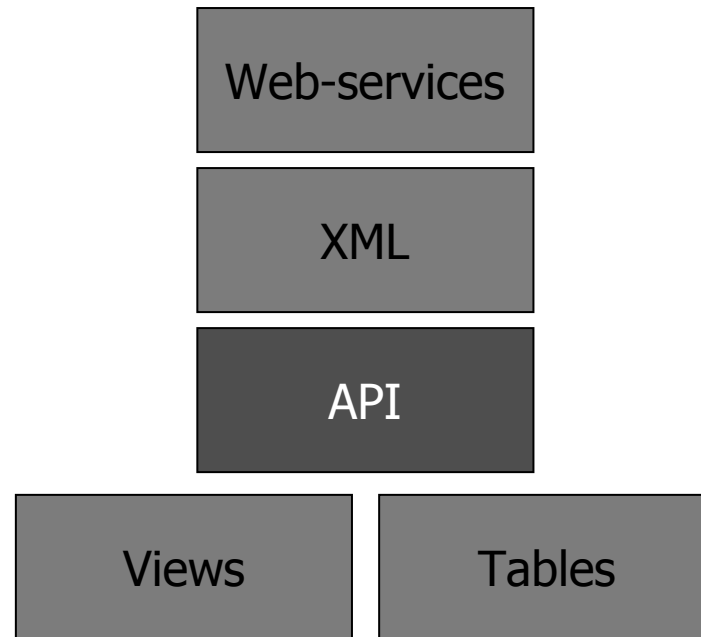


Table Wrapper in Application Stack

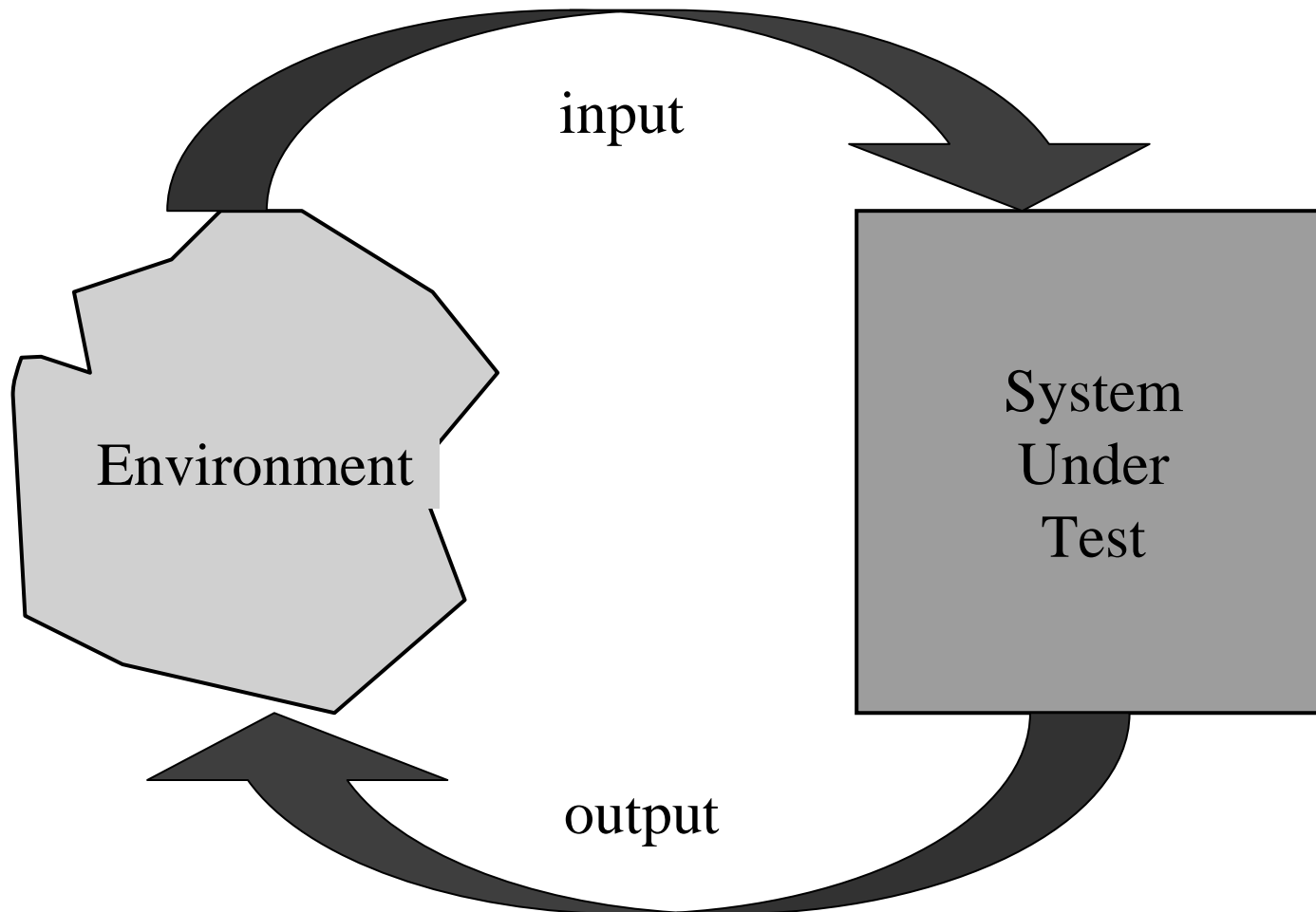


- Table wrapper API more advanced than CRUD
 - For example Hibernate (<http://www.hibernate.org/>) and Cayenne (<http://objectstyle.org/cayenne/>)
 - Customized for each single table

Outline - DBMSUnit

- Motivation
 - Problems existing frameworks
- Test framework design and implementation
 - Static structure
 - Test framework design
 - Dynamic structure via sequence diagrams
- Extensions
- Integration with utPLSQL unit test framework
- Oracle specifics

Testing seen from 30.000 Feet

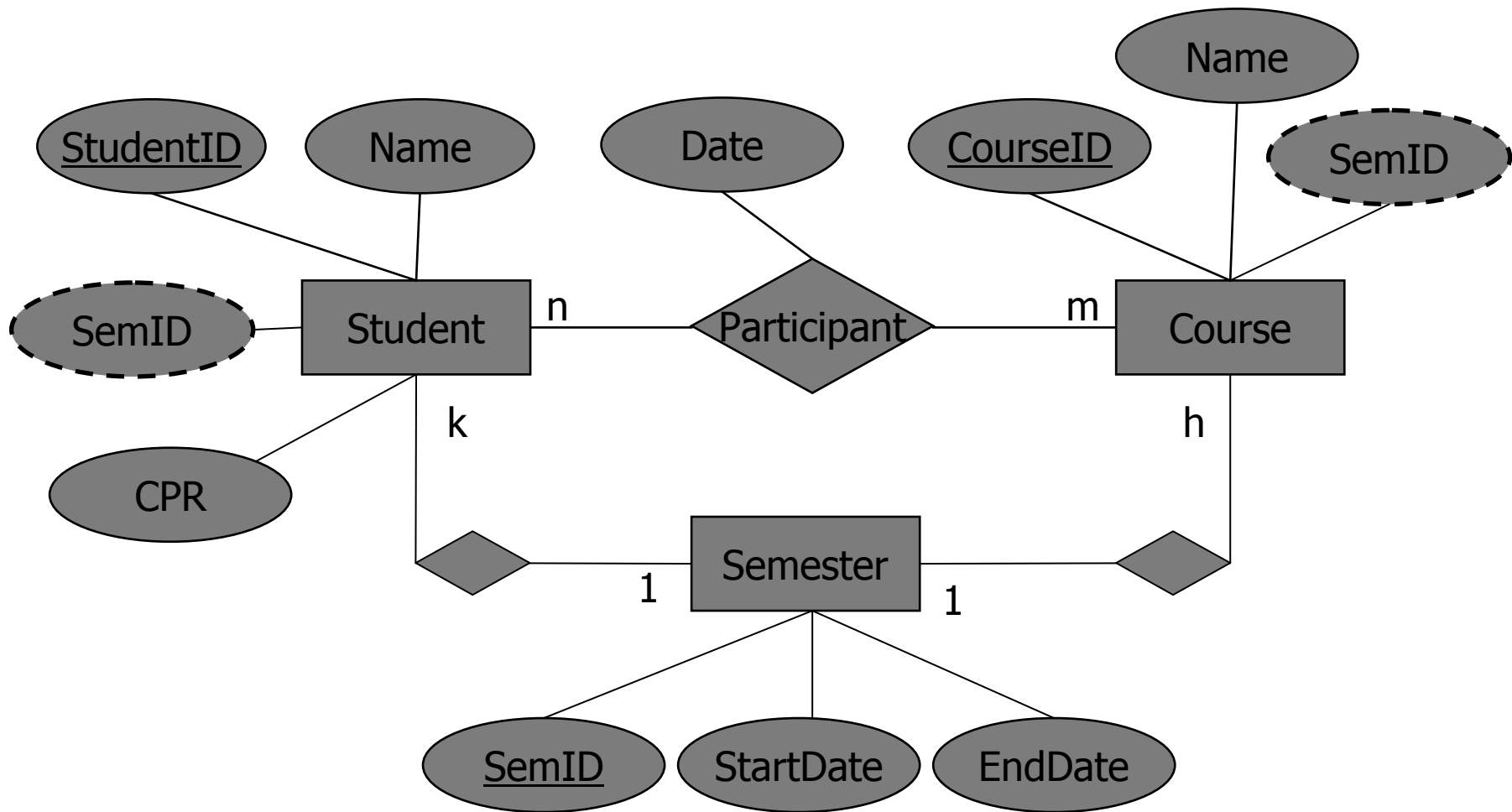


- Testing functions (input/output)
- Testing procedures (side effects)

Problems Existing Unit Test Frameworks

- Coupling between test methods
 - Test methods are independent must always build test fixture
 - Foreign keys in a database context
- Persistency
 - Focus on main memory data structures
 - Side effects (procedures not functions)
 - Manual clean up between executing test cases necessary
- Two-valued logic
 - In a database context three-valued (true, false, and unknown)

A Sample University Schema



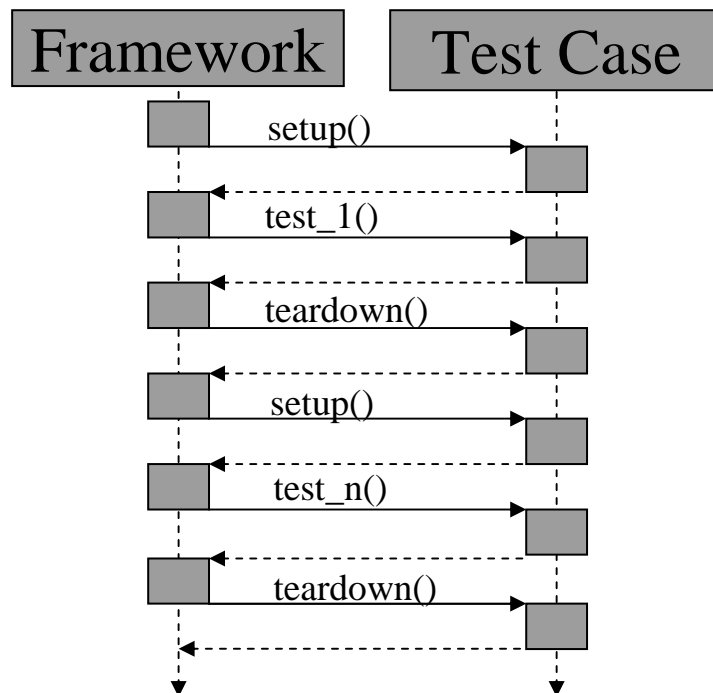
Database Application Test Framework

- Coupling between
 - Test methods
 - Test cases
- Dependent of the state of the database
- Persistency
 - Must test for side effects
 - Must clean up after use

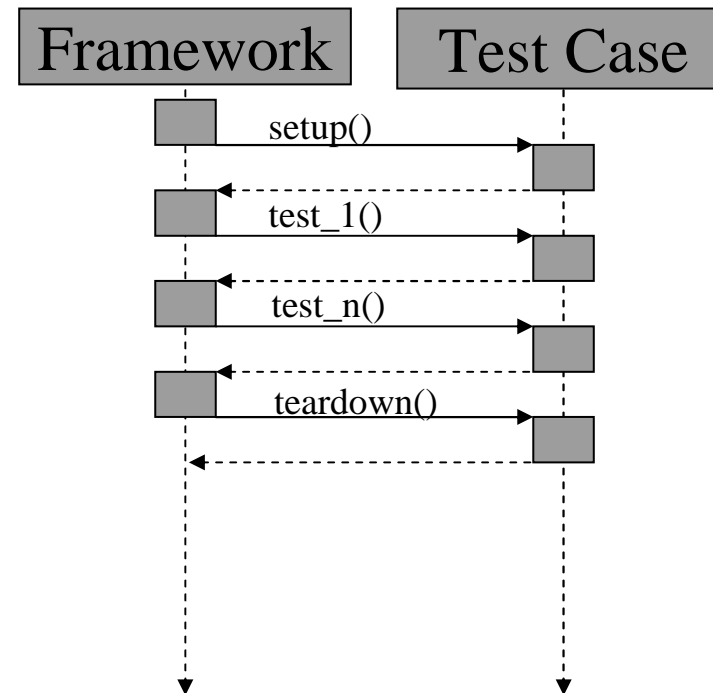
Coupling Between Test Methods

- Setup only executed once for each test case
 - Also done by some JUnit extensions
 - Reason: speed it takes milliseconds to create a database connection

JUnit



DBMSUnit



Coupling Between Test Methods, cont.


- Test methods are no longer independent
 - Must be executed in a specific order!

```
public class MyTestCase extends TestCase{
    // insert rows into underlying table(s)
    public void testInsert1(){ }
    public void testInsert2(){ }

    // query and update rows inserted
    public void testUpdate1(){ }
    public void testQuery1() { }
    public void testQuery2() { }
    public void testQuery3() { }

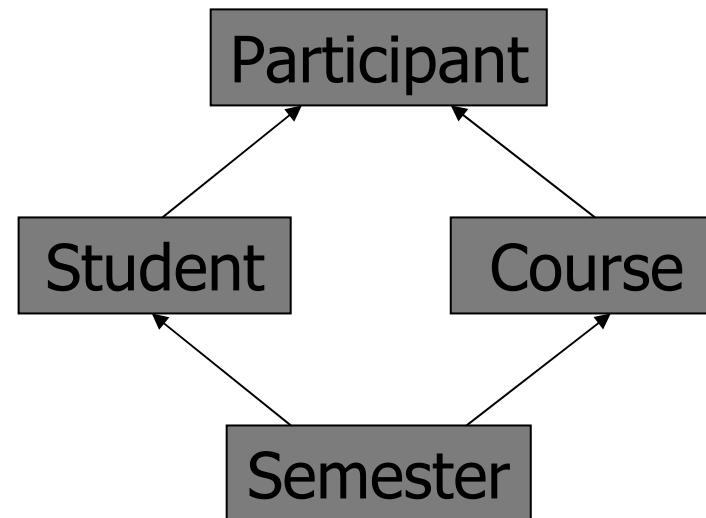
    // delete rows from underlying table(s)
    public void testDelete2(){ }
    public void testDelete1(){ }
}
```

default
order of
execution



Coupling Between Test Cases

- To test student a valid semesterID must exist
- To test course a valid semesterID must exist
- To test participant a valid courseID and studentID must exist
- Reasons
 - Controllability
 - Simpler to build text fixture



Building Test Fixtures in JUnit

Single table

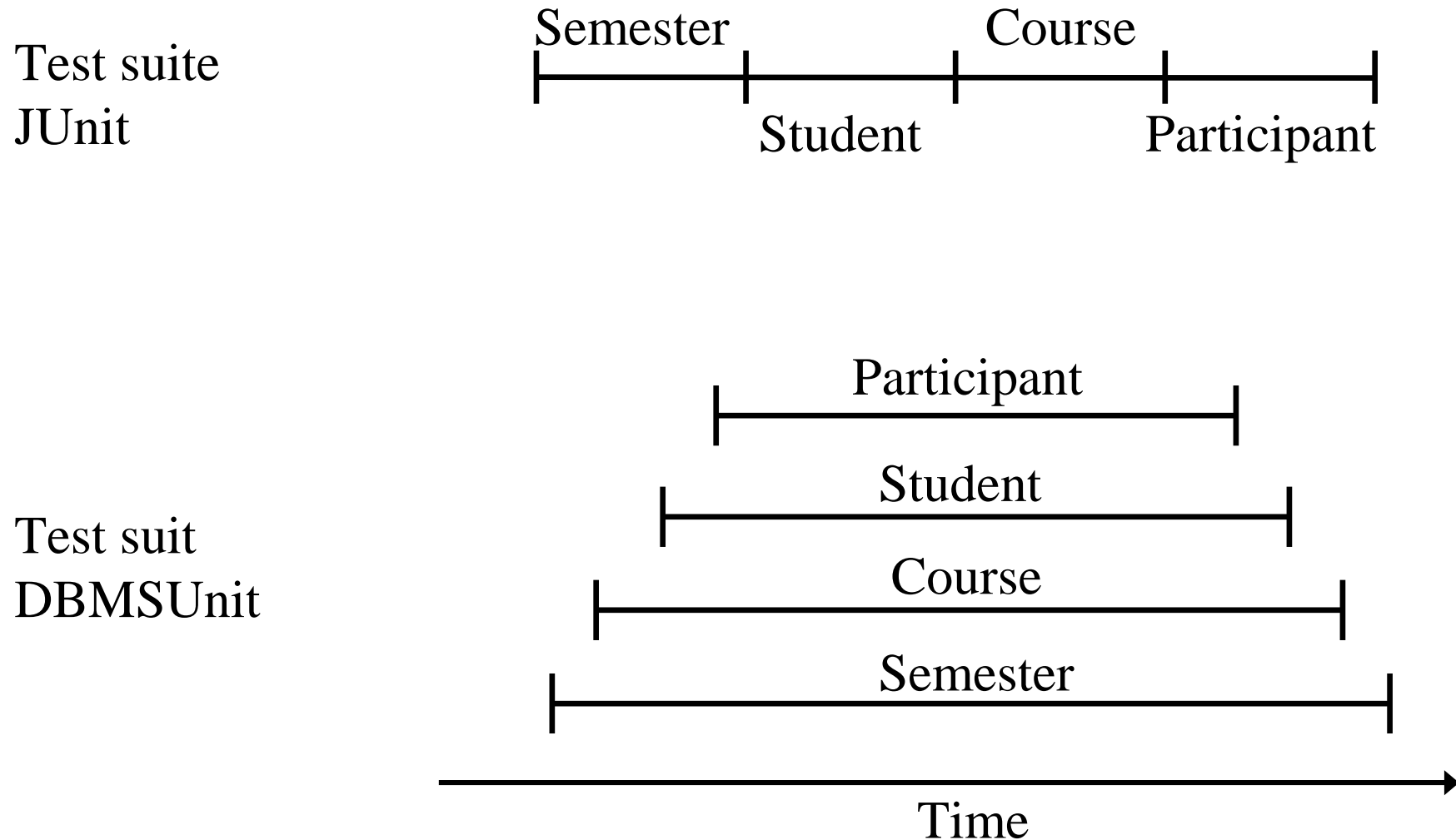
```
public void setup(){
    // semester table
    insert(row1)
    insert(row2)
}
```

Two tables

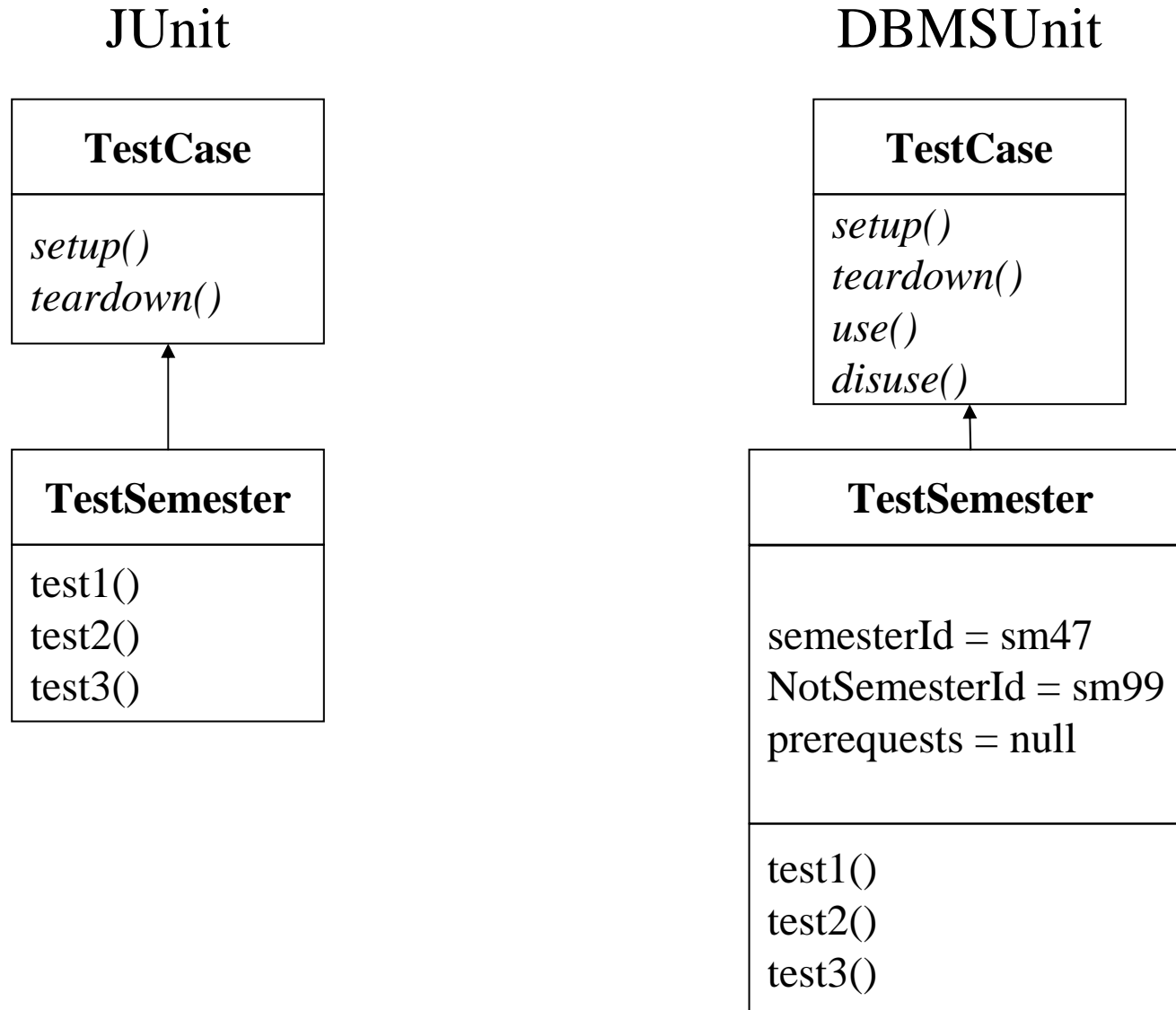
```
public void setup(){
    // semester table
    insert(row1)
    insert(row2)
    // course table
    insert(row3)
    insert(row4)
}
```

- We want to avoid the repetition of code in the setup methods

Coupling Between Test Cases, cont.



Static Structure of Test Case



Static Structure of Test Case, cont

TestSemester
semesterId = sm47 notSemesterId = sm99 prerequisites = null
test1()...

TestStudent
semesterId = TestSemester.semesterID studentId = St43 prerequisites = TestSemester
test1() ...

TestCourse
semesterId = TestSemester.semesterID courseId = Co343 prerequisites = TestSemester
test1() ...

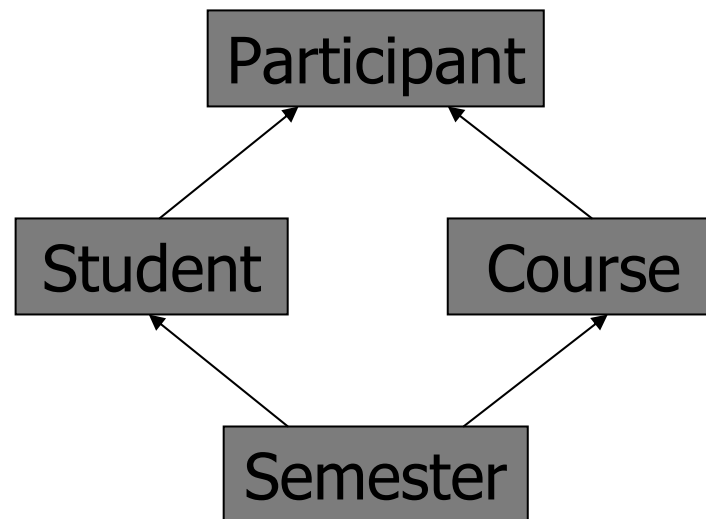
TestParticipant
semesterId = TestStudent.semesterID studentID = TestStudent.studentID courseID = TestCourse.courseID prerequisites = TestCourse, TestStudent
test1() ...

Dynamic Structure of Test Case

- Setup method
 - Responsible for setting up the test fixture
- Teardown method
 - Responsible for tearing down the test fixture
- Use method
 - Responsible for make the public variables exist in the test database.
- Disuse method
 - Responsible for make the public variables non-existing in the test database.
- Run method (called by developer)
 - Calls all test methods in specified order

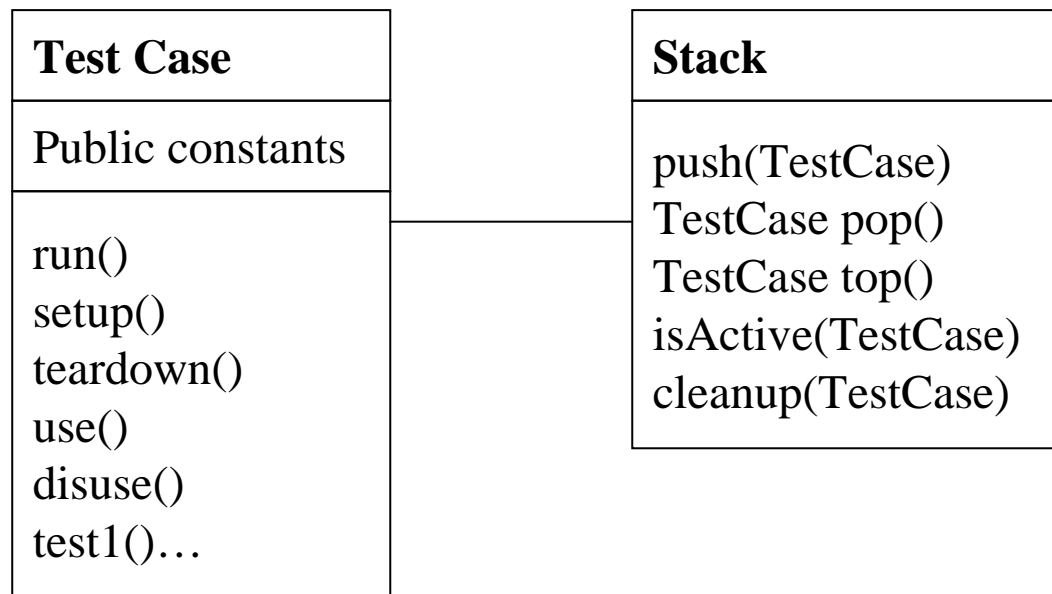
Avoiding Repeated Setups

- If we test *participant* the setup method of *semester* is called twice
 - If both calls inserts data then it leads to integrity constraints violations



The Test Stack

- Controls that setup method is only executed once for each test case in a test session
- Controls the clean-up of the database
- Singleton design pattern
 - only one Stack object (connected to database table)



Setup and Teardown Methods

```
void setup()  
    // setup what this test case depends on  
    for unit_test in prerequisites  
        unit_test.use()  
    // tell that test case is in use  
    stack.push(self)  
    // setup not part of any test method  
    <snip>  
  
void teardown()  
    // own teardown not part of any test method  
    <snip>  
    // tell that test case is longer in use  
    stack.pop()  
    // teardown what this test case depends on  
    for unit_test in reverse(prerequisites)  
        unit_test.disuse()
```

Use and Disuse Methods

```
void use()  
    // setup myself  
    self.setup()  
    // make public variables legal  
    testInsert1() // reuses insert test methods  
    ...  
    testInsertN()
```

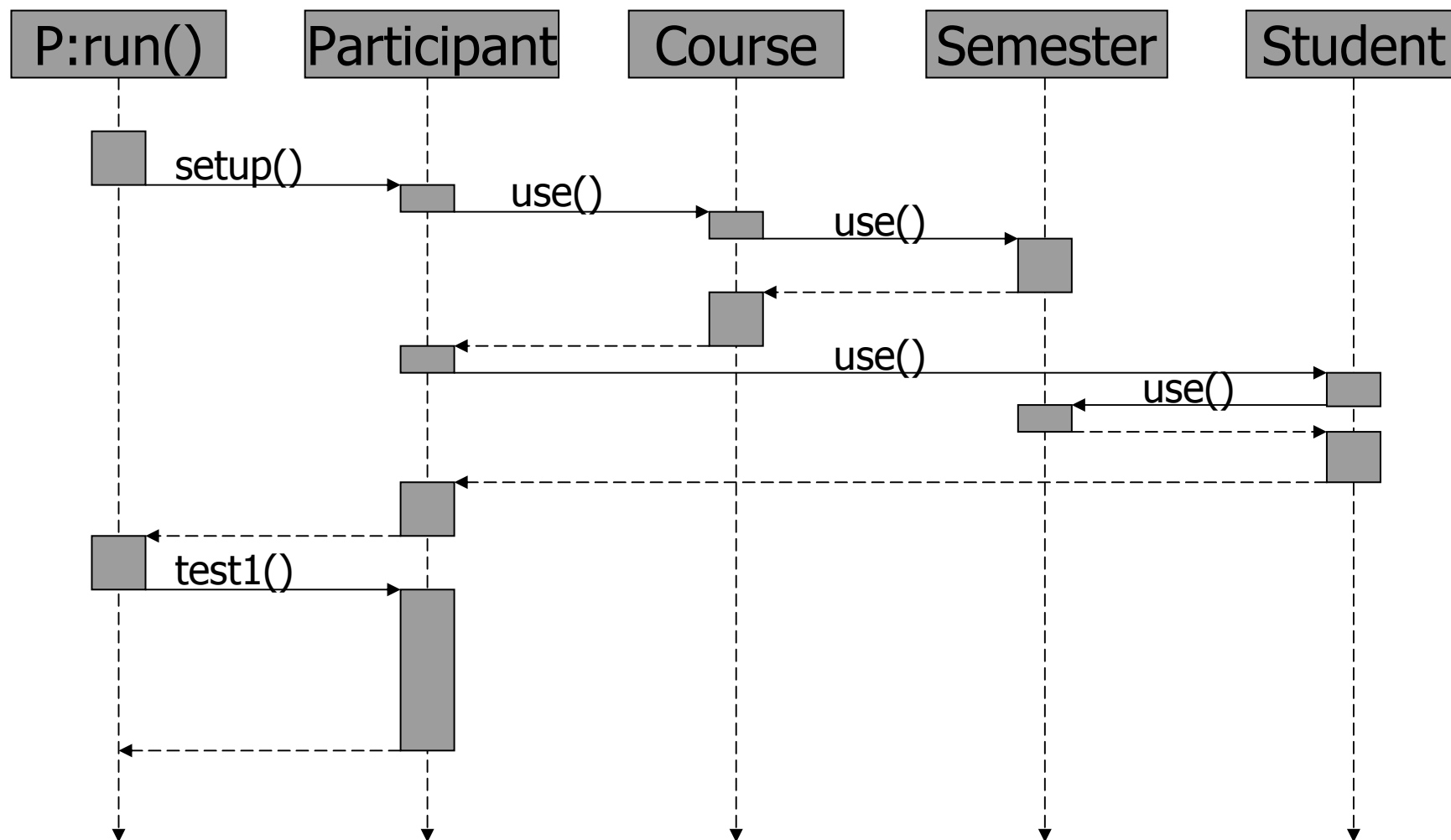
```
void disuse()  
    // make public variables illegal  
    testDeleteN()  
    ...  
    testDelete1()  
  
    // teardown myself  
    self.teardown()
```

The Goal of the Test Framework

- Easier to build test fixtures
 - Extensive reuse between test cases
- Minimal number of restrictions on application developer in test methods
 - None
 - ◆ Insert, update primary key
 - Slight modification
 - ◆ Select, update other than primary key (on specific data)
 - Trade-off
 - ◆ Insert specific primary keys, update (on specific primary keys)
- We must be able to clean up
 - Truncating tables is not an option!

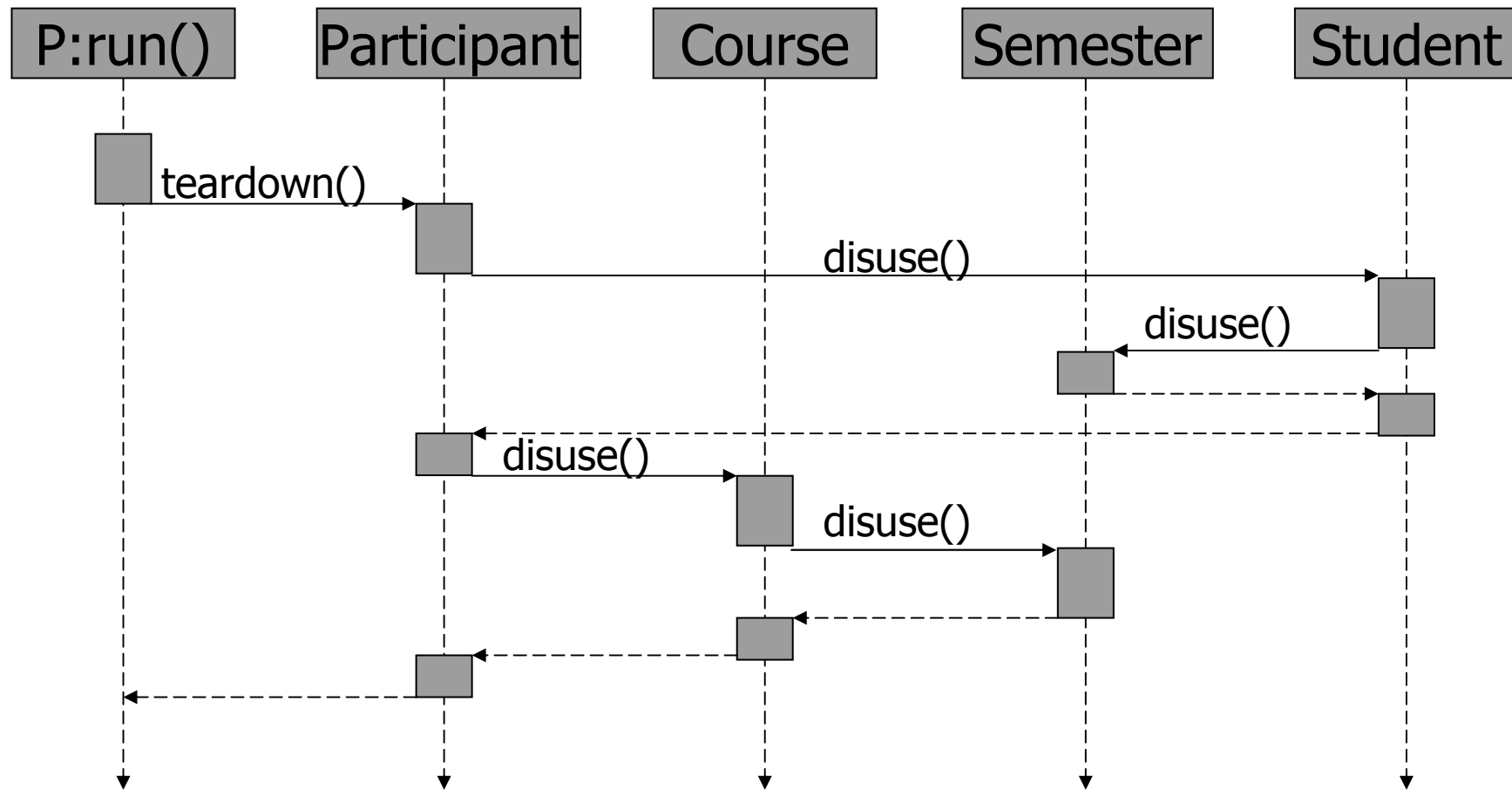
Setup Sequence Diagram

P
St
C
Se
Stack



Teardown Sequence Diagram

P
St
C
Se
Stack



Requirements to Users

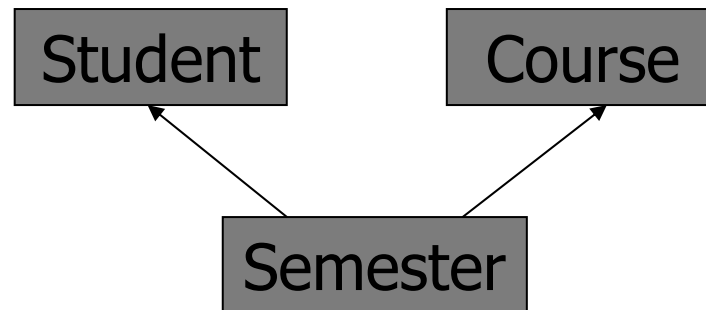
- Must publish keys
 - Inserted in underlying table(s)
 - Updated in underlying table(s)
 - Optional keys guaranteed not to be in underlying table(s)
- Must comply with rules for modification
- Specific order in which test methods are executed
 - Defaults to the textual order of the file
- Must specify test cases that are prerequested

DBMSUnit Extension

- Support multiple users running tests concurrently
 - Goal: Speed up execution of large test suites
- Avoiding repeated setups
 - Goal: Speed up execution of large test suites

Supporting Multiple Users

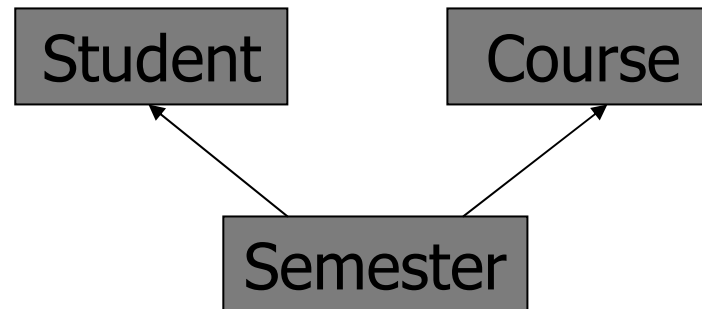
- Testing large applications
- Testing non-connected concepts



- Extensions
 - Stack – user, time to live, being tested, being setup
 - Locking

Avoiding Repeated Setups

- Minimize testing time
- Ordering of unit tests
 - Directed acyclic graph
 - Topological sort
- Teardowns
- Large schemas



Integrating with utPLSQL

- Benefits
 - Assert library
 - Large user community

- Extensions
 - Setup and teardown methods must be redefined
 - Stack must be added
 - Requirements must be followed
 - ◆ Cannot be checked by compiler!

Oracle Specifics

- Flashback queries
 - "Multi-transaction" rollback
 - Ensures that flash back is to a consistent database state
 - Removes the need for the teardown methods
 - ◆ Database "garbage collection"
- Workspace management
 - CVS-like multiple versions of rows
 - "Saves" changed rows to private copy of the database
 - ◆ Private to the user that changed the rows
 - "Checks-in" rows to shared database
 - ◆ Public to all users of the database
 - Allows multiple users to run concurrently

The Framework Offers

- A systematic way of testing a database applications
- Automatic cleanup after testing
 - System failure
 - Error in unit test
- Minimal effort to build test fixture
 - Data only inserted once!
- Simple to use
 - Looks-and-feel of existing xUnit frameworks

Outline - Experiences

- Overall experiences
 - The technical side
 - The human side
- Test patterns
 - Ideas for how to unit test database applications

Experiences

- It is non-trivial to get the test framework to work!
- The first unit test takes a long time to get the up and running due to dependencies
- Building unit tests takes quite a long time and is repetitious in nature (reads boring)
 - Auto-generation can make it more interesting
- When unit test first build, repeated execution is very simple!
- Good error messages are essential for quickly finding the test methods (and test case) that failed.
- Ripple effects could not be avoided due to coupling between test methods and test cases!
 - However, not that hard to find the source of the error
- Brute-force method for cleaning-up the database is nice to have
 - can be difficult to build, due to integrity constraints

Experiences, cont.

- It is important that the test cases can use the data that is delivered with the database application.
 - Can be modelled with public variables (and no insert methods)
- Invariants are good for detecting that an error occurred but not which error
 - As an example, a simple count on the number of rows in a table
- Encapsulate all calls to the system clock such that the time can be frozen and the test always be done at a specific time.
 - Should also be applied to random value function
 - Mock objects can be used for this
- A simple coverage analysis is better than none!
 - All method testes at least once?
 - However, remember to test for behavior not method

Experiences, cont.

- It is important to separate test code from the “real” code.
- Symmetric methods are a thrill to test (high testability)
 - e.g., object2String, string2Object
- Stochastic testing using trace files are not worth the effort.
- We made changes more aggressively and later in the project than usual.
- Do not return values from test methods
 - There is nothing to use it for
- Use dynamic SQL where possible!
 - May lead to hard-to-find errors in the code

Experiences - The Human Side

- The technique is fairly easy to learn
 - If knowledge of, e.g., JUnit before hand then even simpler!
- Test cases should not be used for building permanent test databases
- Manager : "Does the work work?"
Developer: "Yes!"

Test Patterns

- Minimal/maximal pattern
 - Minimal case only the required attributes specified
 - Maximum case all the attributes specified
- NULL pattern (the ugly face of three-valued logic)
 - Replace all the attribute in minimum case with NULL values one by one
- Repeat pattern
 - Run the insert test methods twice, should raise exception
 - Run the copy test methods twice, should raise exception
- Commit pattern
 - Execute all test method in a single transaction
 - Execute each test method in a separate transactions

Future Work

- Stepping through the execution of each test method
- Fully integration with the utPLSQL framework
- Making DBMSUnit a JUnit extension
- Performance measurements on proposed performance “improvements”

- How to use test methods for program documentation

Finding More Information

- comp.software.testing FAQ, good place to start
<http://www.faqs.org/faqs/software-eng/testing-faq/preamble.html>
- Danish site dedicated to software test
<http://www.softwaretest.dk/>
- Better Software magazine
<http://www.stickyminds.com/BetterSoftware/magazine.asp>
- *Softwaretest - kom godt i gang med testen*, booklet, Poul Staal Vinje og Klaus Olsen, Dansk IT, 2004.

Finding More Information, cont.

- Robert Glass *Facts and Fallacies of Software Engineering*, Addison-Wesley, ISBN 0-321-11742-5
 - Good introduction and arguments. No low-level details.
- David Astels, *Test-Driven Development – A Practical Guide*, Prentice-Hall, ISBN 0-13-101649
 - Overview of various unit test framework and very large example. Covers unit test of GUIs.
- Glenford Myers, *The Art of Software Testing*, John Wiley & Sons, 1979, ISBN 0-471-04328-1 (2nd Edition ISBN: 0-471-46912-2)
 - Easy to read and quite pragmatic.
- Boris Beizers, *Software Testing Techniques* Van Nostrand Reinhold, 1990 ISBN: 0-442-24592-0

Reklameblok

Modelbaseret test

Vejen til mere effektiv test

Fr. Bajers Vej 7E – Rum B2-109

Tid: Onsdag 25. august 13.00 – 16.00

<http://www.ciss.dk>

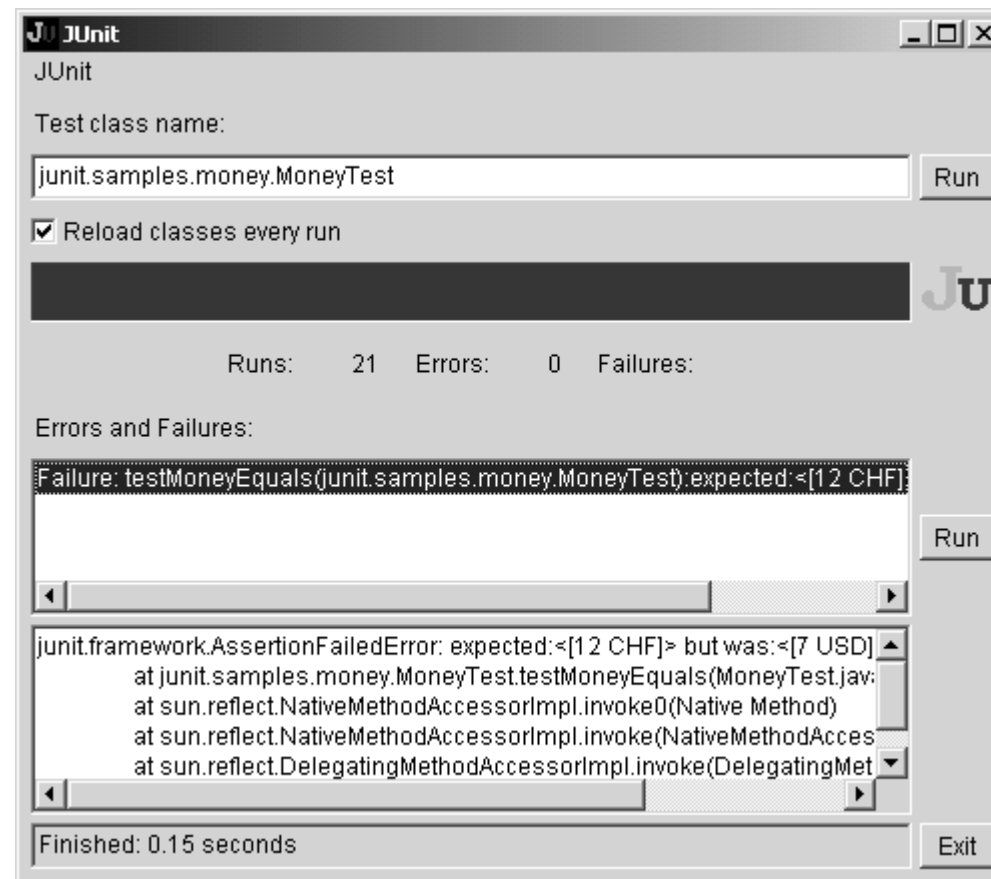
Conclusion

- Unit testing
 - is an inexpensive test approach
 - is very helpful in software maintenance phase.
 - can be the difference between success and fiasko
- ”There is no single best approach to software error removal” R .Glass
- With respect to testing tools one-size-fits all does not apply!
- DBMSUnit is better for testing database than JUnit!
 - Can you live with dependencies between test cases and test methods?

Stable requirements

Accurate estimates

JUnit Screenshot



Two Concrete Test Examples

- University example schema
- Semester test example
 - Single table
- Student test example
 - Dependent tables

Semester Test Example

Semester

SemID
1

Se
Stack

Semester

```
procedure run
begin
  setup();
  test_insert();
  test_update();
  test_exist();
  teardown();
end;
```

```
procedure teardown
begin
  delete(SemID_update);
  delete(SemID);
  commit;
  stack.pop();
end;
```

Student Test Example

Student

SID	SemID
1	1

Semester

SemID
1

Stack
St
Se

Student

```
procedure run
begin
  setup(true);
  test_insert();
  teardown();
end;
```

```
procedure teardown
begin
  delete(SID_update);
  delete(SID);
  commit;
  stack.pop();
  semester.teardown(false);
end;
```