



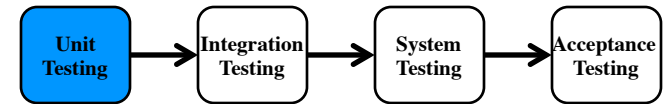
Software Testing: Unit Testing

When should you write a test?

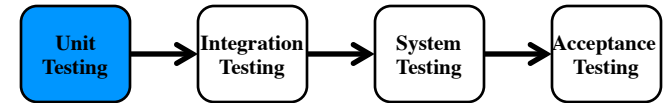
- Traditionally after the source code is written
- In XP before the source code written
 - Test-Driven Development Cycle
 - Add a test
 - Run the automated tests
 - => see the new one fail
 - Write some code
 - Run the automated tests
 - => see them succeed
 - Refactor code



Unit Testing

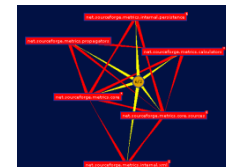


- Static Testing (at compile time)
 - Static Analysis
 - Review
 - Walk-through (informal)
 - Code inspection (formal)
 - Automated tools
 - check for syntax and semantic errors
 - departure from coding standards
- Dynamic Testing (at run time)
 - Black-box testing
 - White-box testing



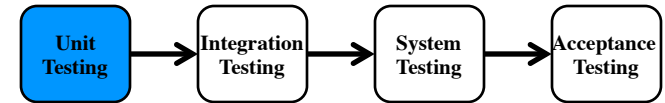
Static Analysis with Eclipse

- Compiler Warnings and Errors
 - *Possibly uninitialized Variable*
 - *Undocumented empty block*
 - *Assignment has no effect*
- Checkstyle
 - Check for code guideline violations
 - <http://checkstyle.sourceforge.net>
- FindBugs
 - Check for code anomalies
 - <http://findbugs.sourceforge.net>
- Metrics
 - Check for structural anomalies
 - <http://metrics.sourceforge.net>



Black-box Testing

- Black-box testing methodology looks at
 - what are the available inputs for an application
 - what the expected outputs are that should result from each input
- Black-box testing is NOT concerned with
 - the inner workings of the application
 - the process that the application undertakes to achieve a particular output
 - any other internal aspect of the application that may be involved in the transformation of an input into an output



Black-box testing

- Focus: I/O behavior
 - If for any given input, we can predict the output, then the component passes the test
 - Requires test oracle (strategy to determine if a test passes or fails)
- Goal: Reduce number of test cases by equivalence partitioning:
 - Divide input conditions into equivalence classes
 - Choose test cases for each equivalence class.

Black-box testing: Test case selection

- a) Input is valid across range of values
 - Developer selects test cases from 3 equivalence classes:
 - Below the range
 - Within the range
 - Above the range
- b) Input is only valid, if it is a member of a discrete set
 - Developer selects test cases from 2 equivalence classes:
 - Valid discrete values
 - Invalid discrete values

Black box testing: An example

```
public class MyCalendar {  
  
    public int getNumDaysInMonth(int month, int year)  
        throws InvalidMonthException  
    { ... }  
}
```

Representation for **month**:

1: January, 2: February,, 12: December

Representation for **year**:

1904, ... 1999, 2000,, 2006, ...

How many test cases do we need for the black box testing of `getNumDaysInMonth()`?

Black box testing: An example contd.

How many test cases do we need for the black box testing of `getNumDaysInMonth()`?

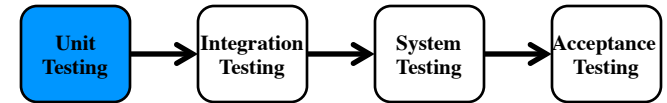
Depends on calendar: gregorian calendar

- Month parameter equivalence classes
 - 30 days
 - 31 days
 - February
 - No month 0, 13, -1
- Year: parameter equivalence classes
 - Normal year
 - Leap year:
 - /4
 - /100
 - /400
 - Before christ/ After Christ year -200
 - Before at and after 2000

=> 12 test cases

White-box Testing

- White-box testing methodology looks under the covers and into the subsystem of an application
- It enables you to see what is happening inside the application
- It provides more than just having access to the user interface
- It allows tester to refer to and interact with the objects that comprise an application



White-box testing overview

- Code coverage (Tool: EclEmma)
 - Statement Coverage
 - Branch coverage
 - Condition coverage
 - Path coverage

White-box testing

```
public int returnInput (int input, boolean condition1, boolean condition2,  
                        boolean condition3)  
{  
    int x = input;  
    int y = 0;  
    if (condition1)  
        x++;  
    if (condition2)  
        x--;  
    if (condition3)  
        y=x;  
    return y;  
}
```

Statement Coverage

TestCase:

shouldReturnInput(x, true, true, true)

100% statements covered

Branch Coverage

TestCase:

shouldReturnInput(x, true, true, true)

100% statements covered

Only half of branches are covered

- Every "if-statement" as two branches (true-branch and false-branch).
- Above test case follows only "true-branches" of every "if-statement". Only 50% of branches are covered.

How can you achieve 100% branch coverage?

- In order to cover 100% of branches we would need to add following testcase: *shouldReturnInput(x, false, false, false)*

Bernd Bruegge & Allen H. Dutoit

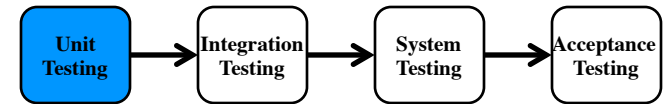


There are 8 separate paths:

- t -t -t - covered with test case 1
- t -t -f
- t -f -t
- t -f -f
- f -t -t
- f -t -f
- f -f -t
- f -f -f - covered with test case 2

Unit Testing Heuristics

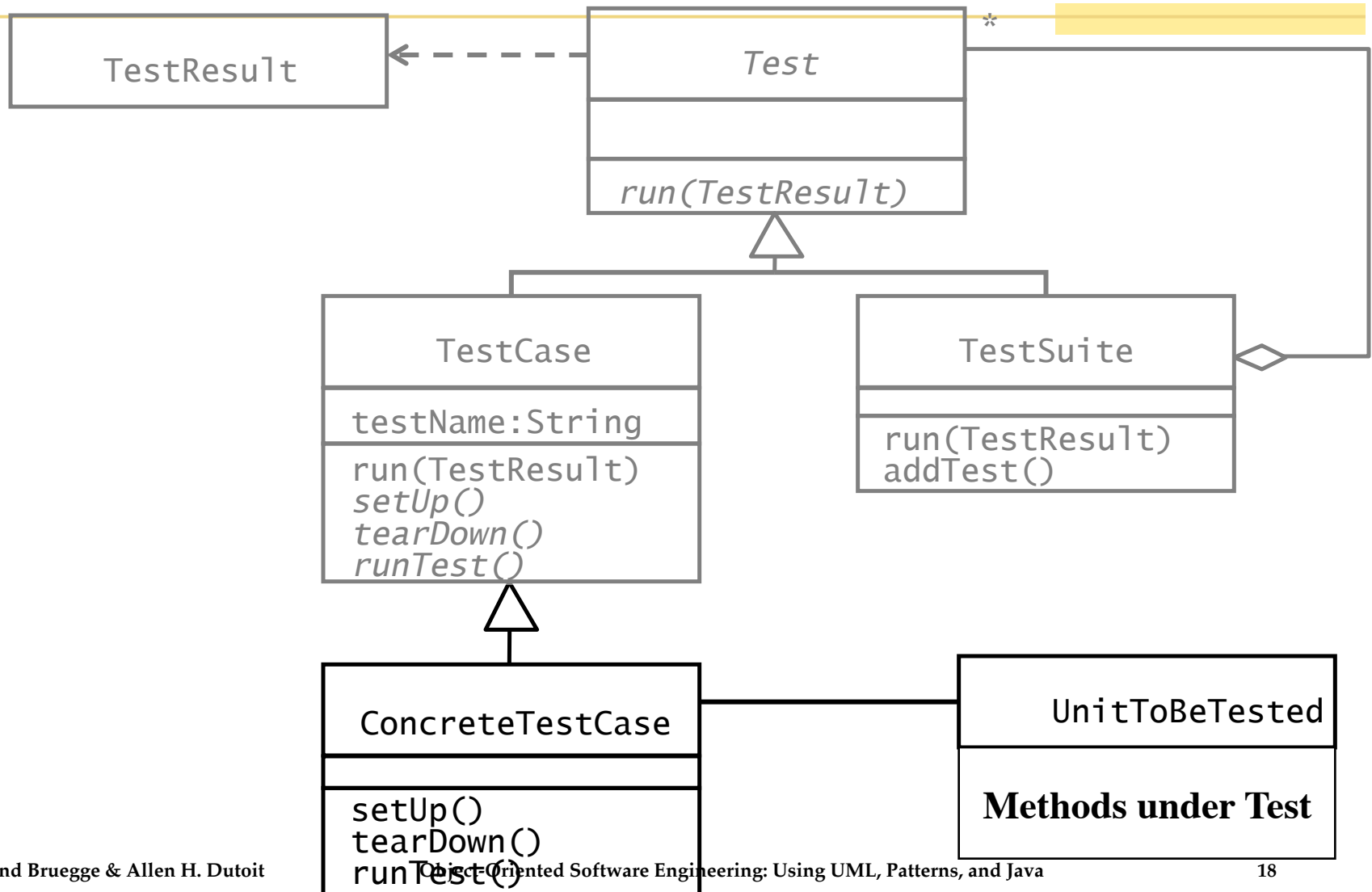
1. Create unit tests when object design is completed
 - Black-box test: Test the functional model
 - White-box test: Test the dynamic model
2. Develop the test cases
 - Goal: Find effective number of test cases
3. Cross-check the test cases to eliminate duplicates
 - Don't waste your time!
4. Desk check your source code
 - Sometimes reduces testing time
5. Create a test harness
 - Test drivers and test stubs are needed for integration testing
6. Describe the test oracle
 - Often the result of the first successfully executed test
7. Execute the test cases
 - Re-execute test whenever a change is made (“regression testing”)
8. Compare the results of the test with the test oracle
 - Automate this if possible.



JUnit: Overview

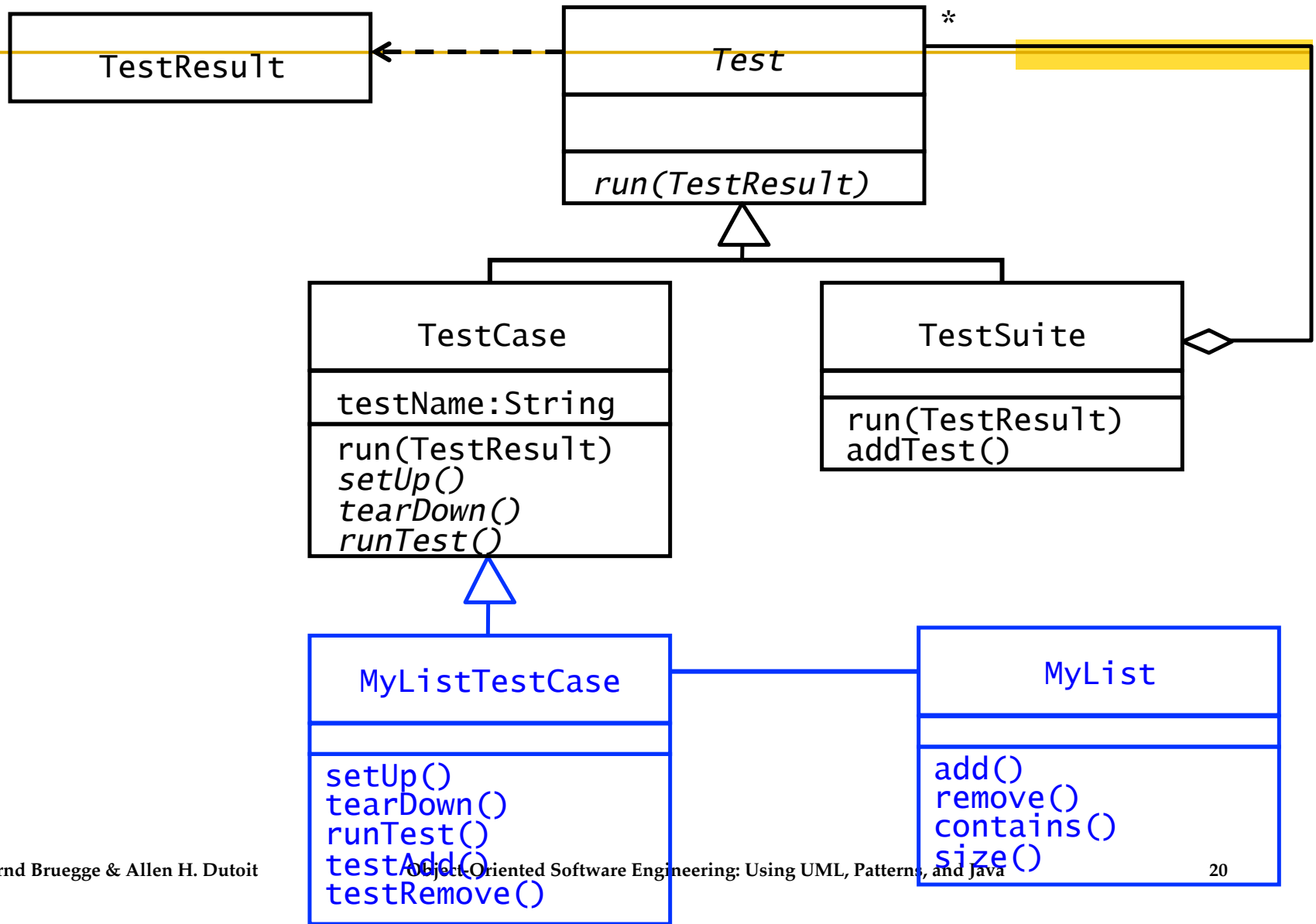
- A Java framework for writing and running unit tests
 - Test cases and fixtures
 - Test suites
 - Test runner
- Written by Kent Beck and Erich Gamma
- Written with “test first” and pattern-based development in mind
 - Tests written before code
 - Allows for regression testing
 - Facilitates refactoring
- JUnit is Open Source
 - www.junit.org

JUnit Classes



An example: Testing MyList

- Unit to be tested
 - MyList
- Methods under test
 - add()
 - remove()
 - contains()
 - size()
- Concrete Test case
 - MyListTestCase



Other JUnit features

- Textual and GUI interface
 - Displays status of tests
 - Displays stack trace when tests fail
- Integrated with Maven and Continuous Integration
 - <http://maven.apache.org>
 - Build and Release Management Tool
 - <http://Maven.apache.org/continuum>
 - Continuous integration server for Java programs
 - All tests are run before release (regression tests)
 - Test results are advertised as a project report
- Many specialized variants
 - Unit testing of web applications