

Advanced Techniques for Building Test Plans

Path Testing and Code Review

School of Computing
Clemson University
Clemson, SC 29634 USA

CPSC 2150: Wednesday, Mar. 8, 2023

Slide Sources:

- 1 Previous/Current CPSC 2150 Instructors
- 2 The lecture slides have also benefitted from instructors at Ohio State: Paolo Bucci, Wayne Heym, Paul Sivilotti, and Bruce Weide.

Last Class

- 1 Serious Testing: JUnit
- 2 Generic Example
- 3 New Vocabulary
- 4 Execution Model
- 5 JUnit Assertions
- 6 Timed Tests
- 7 Best Practices
- 8 Running JUnit on School of Computing Unix Machines

Homework Assignment

- 1 Programming assignment 3 is posted on your lab Canvas page.

Quiz

- 1 Path testing quiz is due Mar. 12 @ 11:59 pm!

Recall

- ▶ We've learned the basics of testing
- ▶ We've learned how to automate testing with JUnit
- ▶ Now we need to learn how to pick good test cases
 - ▶ Can't test all possible input
 - ▶ Some basic rules:
 - ▶ Pick routine cases
 - ▶ Pick boundary cases
 - ▶ Pick challenging cases
 - ▶ etc.
 - ▶ With more difficult test components this can be tough

Recall

Blackbox vs Whitebox Testing

- ▶ Blackbox
 - ▶ Focus on the input and output behavior
 - ▶ Ignore internal details or structure of the component
 - ▶ Use the contracts to create our test plans
 - ▶ Test cases don't change if the implementation changes
- ▶ Whitebox
 - ▶ Focuses on internal structure of the component.
 - ▶ Every state of the object and interactions are also tested
 - ▶ Look at the code while developing test cases

Blackbox Testing

Leap Year Example

Consider a method that takes in a year and returns true if that year is a leap year.

- ▶ What test cases would we create?

```
/**
 * ...
 *
 * @pre year > 0
 * @post isLeapYear iff
 *         ((year mod 4 = 0) AND
 *          ((year mod 100 != 0) OR (year mod 400 = 0)))
 */
boolean isLeapYear(int year)
```

Blackbox Testing

Days in Month Example

Consider a function that takes in a month and year and returns the number of days in the month.

- ▶ What test cases would we create?

```
/**
 * ...
 *
 * @pre 1 <= month <= 12 and year > 0
 * @post daysInMonth = 31 iff month in (1, 3, 5, 7, 8, 10, 12) and
 *       daysInMonth = 30 iff month in (4, 6, 9, 11) and
 *       daysInMonth = 28 iff (month = 2 and !isLeapYear(year)) and
 *       daysInMonth = 29 iff (month = 2 and isLeapYear(year))
 */
int daysInMonth(int month, int year);
```

Path Testing

- ▶ Whitebox testing technique
- ▶ Travel all possible paths in the code at least once
 - ▶ Have inputs to make every `if` statement true and false
- ▶ Start with a UML Activity diagram for the code
 - ▶ Must use the code written, not our own design
 - ▶ Nodes are executable blocks of code
 - ▶ Edges are flow of control
- ▶ Find inputs that lead down all possible paths
- ▶ Whitebox testing will only detect faults that come from exercising a path in the program, not any faults due to omission
 - ▶ No method is perfect

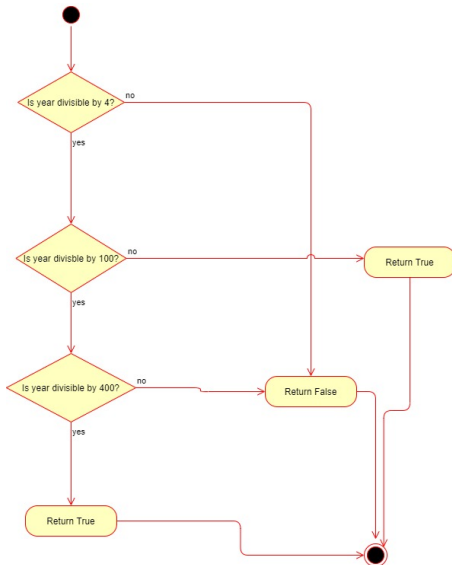
Path Testing

Practice Exercise #1

```
boolean isLeapYear(int year) {  
    if (year % 4 == 0) {  
        if (year % 100 == 0) {  
            if (year % 400 == 0) {  
                return true;  
            }  
            else {  
                return false;  
            }  
        }  
        return true;  
    }  
    else {  
        return false;  
    }  
}
```

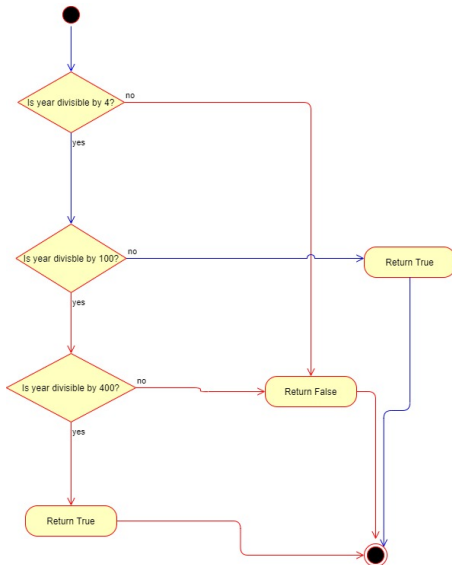
Path Testing

Practice Exercise #1 - Activity Diagram



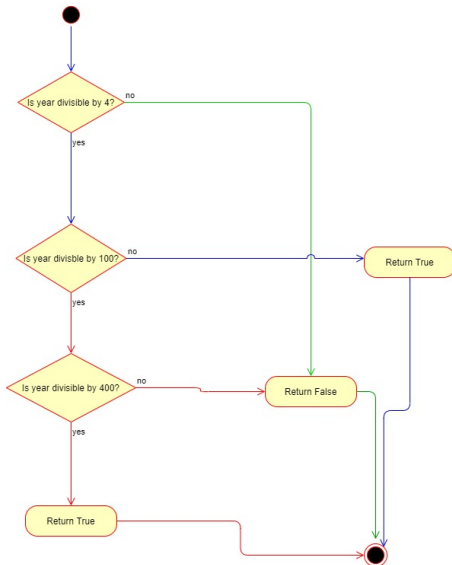
Path Testing

Practice Exercise #1 - Path 1



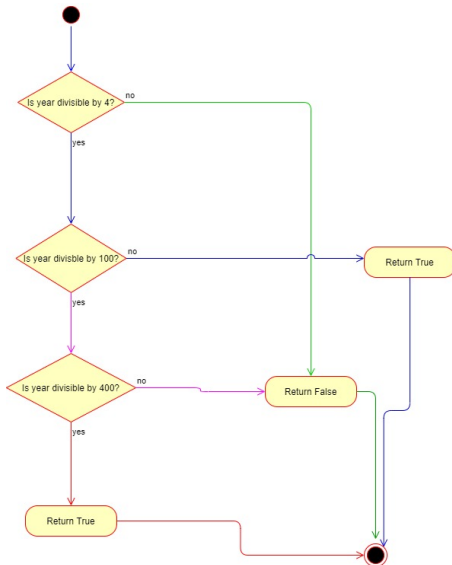
Path Testing

Practice Exercise #1 - Path 2



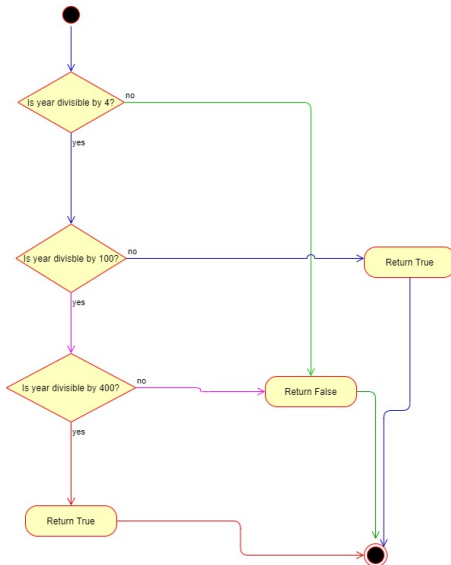
Path Testing

Practice Exercise #1 - Path 3



Path Testing

Practice Exercise #1 - Path 4



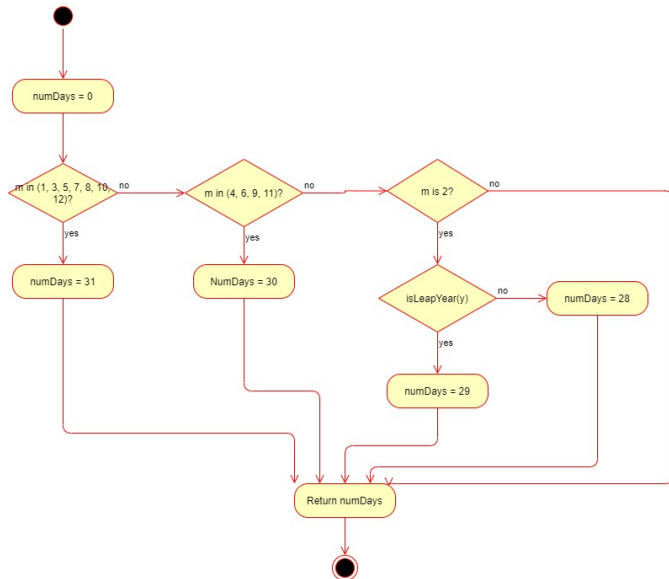
Path Testing

Practice Exercise #2

```
int daysInMonth(int m, int y) {  
    int numDays = 0;  
    if (m == 1 || m == 3 || m == 5 || m == 7 ||  
        m == 8 || m == 10 || m == 12) {  
        numDays = 31;  
    }  
    else if (m == 4 || m == 6 || m == 9 || m == 11) {  
        numDays = 30;  
    }  
    else if (m == 2) {  
        if ( isLeapYear(y) ) {  
            numDays = 29;  
        } else {  
            numDays = 28;  
        }  
    }  
  
    return numDays;  
}
```

Path Testing

Practice Exercise #2 - Activity Diagram



Blackbox testing

- ▶ Doesn't change when the code changes
- ▶ Doesn't guarantee full code coverage
 - ▶ Could miss odd exceptions
- ▶ Reliant on very clear post conditions to find expected output
- ▶ Harder to think of test cases
 - ▶ Not a set process
 - ▶ Have to guess at what would have been challenging to code

Path testing

- ▶ Covers 100% of the code
 - ▶ More likely to catch random bugs and exceptions
- ▶ Only detect faults that come from exercising a path in the program
 - ▶ Not any faults due to omission
- ▶ Requires viewing the code to create test cases
 - ▶ Test cases change when the code changes
- ▶ Will need code developer wrote the implementation to test for challenging cases
 - ▶ Assuming they wrote that code

Testing

- ▶ Use multiple methods to develop test cases
- ▶ Increases testing time
 - ▶ Automation makes this less of an issue
- ▶ The more test cases you run, the more confident you will be with your code
 - ▶ It can be easy to miss the one odd case that can cause an error
 - ▶ We can only know our code is correct if we try every possible input
- ▶ How to handle errors of omissions?
 - ▶ Test to the requirements
- ▶ How to build more confidence in the code?
 - ▶ Code Reviews (aka Component Inspections)

Code Reviews

- ▶ Reviewing the source code in a formal meeting
- ▶ Conducted by team
 - ▶ Developers
 - ▶ Author of the component
 - ▶ Moderator
 - ▶ Additional reviewers
- ▶ Time consuming process
- ▶ Has been shown to usually be more effective than regular testing
 - ▶ Is not a replacement for testing, just an additional tool
- ▶ Greatly depends on the quality of the team members
- ▶ A company standard

Code Reviews

1 Overview

- ▶ Author briefly presents purpose and scope of the component

2 Preparation

- ▶ Reviewers become familiar with the implementation

3 Inspection Meeting: Step through the code

- ▶ Team raises potential issues
 - ▶ Can be faults, best practice violations, or design concerns
 - ▶ Can be more elegant or efficient code suggestions
- ▶ Don't try to fix faults yet, just identify

4 Rework

- ▶ Author revises based on the faults found

5 Follow-up

- ▶ Moderator checks on quality of rework and decides in the component needs re-inspected

Testing and Code Review Process

- ▶ Design class, specify contracts, but don't write code yet
- ▶ Write test cases (if using TDD)
- ▶ Write your code
- ▶ Write test cases (if not using TDD)
- ▶ Run all automated tests
 - ▶ Debug and fix any faults
 - ▶ Loop until all tests pass
- ▶ Perform Code Review
- ▶ Fix faults and design issues
- ▶ Run all automated tests
 - ▶ Debug and fix any faults
 - ▶ Loop until all tests pass
- ▶ Code Review follow up if necessary...

Path Testing

Practice Exercise #3

```
/**
 * @pre 0 <= interestRate <= .20 AND
 *      MIN_YEARS <= years <= MAX_YEARS AND
 *      0 <= cs <= 900 AND
 *      .035 <= percentDown < 1
 */
private double calcRate(double interestRate, int years, int cs, double percentDown) {
    if (years >= MIN_YEARS && years < MAX_YEARS) {
        interestRate += GOODRATEADD;
    }
    else if (years == MAX_YEARS) {
        interestRate += NORMALRATEADD;
    }

    if (cs < BADCREDIT) {
        interestRate += VERYBADRATEADD;
    }
    else if (cs < FAIRCREDIT) {
        interestRate += BADRATEADD;
    }
    else if (cs < GOODCREDIT) {
        interestRate += NORMALRATEADD;
    }
    else if (cs < GREATCREDIT) {
        interestRate += GOODRATEADD;
    }

    if (percentDown < PREFERRED_PERCENT_DOWN) {
        interestRate += GOODRATEADD;
    }

    return interestRate;
}
```

```
// MIN_YEARS = 15
// MAX_YEARS = 30
// PREFERRED_PERCENT_DOWN = .2;

// BADCREDIT = 500;
// FAIRCREDIT = 600;
// GOODCREDIT = 700;
// GREATCREDIT = 750;

// double BASERATE = .025;
// double GOODRATEADD = .005;
// double NORMALRATEADD = .01;
// double BADRATEADD = .05;
// double VERYBADRATEADD = .1;
```

Homework Assignment

- 1 Programming assignment 3 is posted on your lab Canvas page.

Quiz

- 1 Path testing quiz is due Mar. 12 @ 11:59 pm!

Summary

- 1 Recall
 - ▶ Blackbox
 - ▶ Whitebox
- 2 Blackbox Testing Examples
- 3 Path Testing
 - ▶ In-Class Exercises
- 4 Comparisons
- 5 Testing and Code Review Process