Introduction to Software Engineering
Assignment 9

Walter Tichy

Title page: Create a title page with ”Introduction to Software Engineering”, “Assignment 9”, your name, and date of completion.

Problem 1: (4 pt) Functional Test SS19/B6

Consider the following interface.

```
public interface Account {
  private BigDecimal balance;
  //...

  /**
   * Withdraws the amount specified, provided balance+creditLine >= amount
   *
   * @param amount
   *          The amount to be withdrawn (>= 0)
   * @param creditLine
   *          The credit line of the account (how much one can borrow) (>= 0)
   * @return true if withdrawal is possible, false otherwise
   */
  public boolean withdraw(BigDecimal amount, BigDecimal creditLine);
  //...
}
```

a) Determine the equivalence classes of method withdraw() and provide representatives. Describe each equivalence class in a short sentence.

b) Determine the boundaries of your equivalence classes. Which additional values should be tested when taking the boundaries into account?

Problem 2: (7 pt + 3 bonus pt) Control flow-oriented test SS19/B6

Consider the method `isIdentity()` .

```
01:public boolean isIdentity(double[][] mtx) {
02:  if (mtx == null || mtx.length != mtx[0].length) {
03:    return false;
04:  }
05:  int n = mtx.length;
06:
07:  for (int r = 0; r < n; r++) {
08:    for (int c = 0; c < n; c++) {
09:      if (r == c && Math.abs(mtx[r][c] - 1) >= 1E-8) {
10:        return false;
11:      } else if (r != c & Math.abs(mtx[r][c]) >= 1E-8) {
12:        return false;
13:      }
14:    }
15:  }
16:  return true;
17:}
```

a) Translate the code of `isIdentity(double[][] mtx)` into the intermediate language presented in class.

b) Draw the control flow graph for `isIdentity(double[][] mtx).` Enter the source code statements into the boxes and number the boxes. (Only entering the line numbers is not sufficient.)

c) Provide a minimal test case set and the paths traversed by them, such that statement coverage is achieved. Do the same for branch coverage. If it is not possible to achieve full coverage, explain why.

<u>Hint:</u> „Minimal" means the test case set must contain enough test cases to achieve coverage, and not more.

d) (2 bonus pt) Which test cases do you need for boundary-interior test? Explain how you would handle the nested loops in this case.

e) (1 bonus pt) Explain why path coverage for `isIdentity(double[][] mtx)` is not practicable.

Problem 3: (4 pt) Code Inspections SS19/B6

Inspect the class `Matrix` below for issues. An issue is a violation of one of the rules in the checklist following the code. Record each issue in a table of the following form:

Rule number	Line number(s)	Short description
⋮	⋮	⋮

If you discover an issue, enter the violated rule, the line number in the source text, and a short description of the issue.

BIG EXCEPTION: After you have entered the issues in your table, you have the option to meet in a team of up to 3 students total to discuss and combine your findings. Indicate on your solution the names of your team members, so we don't have to grade your solutions multiple times. To further simplify, it would be good if you chose team members from your group. Mind: you must first find issues by YOURSELF, BEFORE the group meeting. The purpose of this exercise is to practice a real inspection. You will not lose points for being in a team.

```
01: package org.iMage.HDrize.matrix;
02:
03: import org.iMage.HDrize.base.matrix.IMatrix;
04: import org.ojalgo.matrix.store.PrimitiveDenseStore;
05:
06: /**
07:  * A matrix.
08:  *
09:  */
10: public final class Matrix implements IMatrix {
11:
12:   private final int rows;
13:   private final int cols;
14:
15:   final double[][] data;

16:   /**
17:    * Create a new matrix.
18:    *
19:    * @param m the original matrix
20:    */
21:   public Matrix(IMatrix m) {
22:     this(m.copy());
23:   }
24:
25:   /**
26:    * Create a new matrix.
27:    *
28:    */
29:   private Matrix(double[][] m) {
30:     this(m.length, m.length == 0 ? 0 : m[0].length);
31:     for (int r = 0; r < this.rows; r++) {
32:       if (m[r].length != this.cols) {
33:         throw new IllegalArgumentException("Rows have not equal lengths.");
```

```java
      }
      for (int c = 0; c < this.cols; c++)
        this.set(r, c, m[r][c]);
    }

  }

  /**
   * Create a matrix (only zeros).
   *
   * @param rows the amount of rows
   */
  public Matrix(int rows, int cols) {
    if (rows < 1 || cols < 1) {
      throw new IllegalArgumentException("Rows and cols have to be >= 1");
    }
    this.rows = rows;
    this.cols = cols;
    this.data = new double[rows][cols];
  }

  @Override
  public double[][] copy() {
    double[][] copy = new double[this.rows][this.cols];
    for (int r = 0; r < this.rows; r++) {
      for (int c = 0; c < this.cols; c++) {
        copy[r][c] = this.data[r][c];
      }
    }
    return copy;
  }

  @Override
  public int rows() {
      return this.rows;
  }

  @Override
  public int cols() {
      return this.cols;
  }

  @Override
  public void set(int r, int c, double v) {
    this.data[r][c] = v;

  }

  @Override
  public double get(int r, int c) {
    return this.data[r][c];
  }

  @Override
  public String toString() {
    StringBuilder builder = new StringBuilder();
    String closeParen = "}";
    builder.append("{\n");
    for (int r = 0; r < this.rows(); r++) {
        builder.append("{");
        for (int c = 0; c < this.cols(); c++) {
          builder.append(this.get(r, c));
          if (c != this.cols() - 1) {
              builder.append(", ");
          }
        }
        builder.append("}\n");
       }
       builder.append("}");
       return builder.toString();
     }
   }
 }
```

Checklist for Java

1. **Variable and Constant Declarations**

Rule 10. Are all identifiers intelligible and in accordance with the Java naming conventions?

Rule 11. Are the types of all variables and constants correct?

Rule 12. Are all variables and constants correctly initialized?

Rule 13. Are there literals that should be declared as constants?

Rule 14. Are there local variables, instance or class variables that should be constant?

Rule 15. Are there class or instance variables that should be local?

Rule 16. Have class and instance variables the appropriate visibility? (default, `private`, `protected`, `public`)?

Rule 17. Are there instance variables that should be static or vice versa?

Rule 18. Are there libraries that are not used anywhere?

2. **Method and Constructor Declarations**

Rule 20. Are all identifiers intelligible and in accordance with the Java naming conventions?

Rule 21. Is every input parameter checked for correct value?

Rule 22. Does every `return`-statement deliver the correct value?

Rule 23. Have all methods and constructors the appropriate visibility? (default, `private`, `protected`, `public`)?

Rule 24. Are there instance methods or constructors that should be static or vice versa?

3. **Comments**

Rule 30. Have all classes and all non-private constructors, methods, constants, class and instance variables a complete JavaDoc comment?

Rule 31. Do comments describe the associated code sections correctly?

4. **Layout**

Rule 40. Is indentation correct and consistent?

Rule 41. Do all blocks have curly braces?

5. **Performance**

Rule 50. Is it possible to cache values rather than computing them repeatedly?

Rule 51. Is every computed value used?

Rule 52. Can computations be moved out of loops?