# Software Testing, Verification and Validation

December 14, 2022

Week #14— Lecture #11

U. PORTO

FEUP FACULDADE DE ENGENHARIA

UNIVERSIDADE DO PORTO

## *Final exam*

## When and where

1st

- January 19, 2023 at 2:30pm — 5:00pm
- Room: B338

2nd

- February 2, 2023 at 2:30pm — 5:00pm
- Room: B329

## Time

- You will have 120 minutes to complete the exam.
    - In the first 60 minutes, you are not allowed to leave the room.
    - Once the first 60 minutes have passed, you are allowed to hand over your exam and leave the room.

- You should answer every single question to the best of your knowledge and you must write every single answer in the provided pieces of paper. (An extra piece of paper will be provided such that you can draft some of your questions.) No other pieces of paper will be considered or evaluated.

# Structure

- Group I is worth 25% (i.e., 5 points out of 20) and it is composed by 20 multiple choice questions.
  (a) There is only one correct answer!
  (b) Each correct answer is worth +1/20 and each incorrect answer is worth (-1/20)/2. In case you get a negative overall score in this Group, your score will be truncated to 0.
- Group II is worth 25% (i.e., 5 points out of 20) and it is composed by one single open question with one or more subquestions.
- Group III is worth 25% (i.e., 5 points out of 20) and it is composed by one single open question with one or more subquestions.
- Group IV is worth 25% (i.e., 5 points out of 20) and it is composed by one single open question with one or more subquestions.

## Mock exams

- MESW, https://moodle.up.pt/mod/folder/view.php?id=93973
- MEI.C, https://moodle.up.pt/mod/folder/view.php?id=93974

## *Mock questions*

*(2018 v2)*

## Group 3

Consider the following code:

```java
/**  Check if a given string is a palindrome.
 *   A palindrome is a string that is the same
 *   when read right-to-left.
 */
public static boolean isPalindrome(String s) {
  if (s == null)
    throw new NullPointerException();
  int left = 0;
  int right = s.length() - 1;
  boolean result = true;
  while (left < right && result == true) {
    if (s.charAt(left) != s.charAt(right))
      result = false;
    left++;
    right--;
  }
  return result;
}
```

a) Derive a test suite that ensures 100% decision coverage.

b) Derive a test suite that ensures 100% condition coverage.

## *Mock questions*

*(2009 v1)*

**Grupo 3.** [3 valores] Consider a class describing an unbounded stack of generic elements E with the usual semantics, and including the following public method signatures.

```
public Stack ();
public void push (E element);
public void pop ();
public E top();
public boolean isEmpty();
```

**a)** Derive a set of equivalence classes.

**b)** Prepare a test suit. For each test specify the associated equivalence class, the driving (input) code, the expected output and the code necessary to check the result.

**Grupo 4.** [4 valores]

```
 /** *************************************************
  * Finds and prints n prime integers
  * Jeff Offutt, Spring 2003
  *****************************************************/
private static void printPrimes (int n)
{
  int curPrime; // Value currently considered for primeness
  int numPrimes; // Number of primes found so far.
  boolean isPrime; // Is curPrime prime?
  int [] primes = new int [MAXPRIMES];

  // Initialize 2 into the list of primes.
  primes [0] = 2;
  numPrimes = 1;
  curPrime = 2;
  while (numPrimes < n)
  {
    curPrime++; // next number to consider ...
    isPrime = true;
    for (int i = 0; i <= numPrimes-1; i++)
    { // for each previous prime.
      if (isDivisible (primes[i], curPrime))
      { // Found a divisor, curPrime is not prime.
        isPrime = false;
        break; // out of loop through primes.
      }
    }
    if (isPrime)
    { // save it!
      primes[numPrimes] = curPrime;
      numPrimes++;
    }
  } // End while

  // Print all the primes out.
  for (int i = 0; i <= numPrimes-1; i++)
  {
    System.out.println ("Prime: " + primes[i]);
  }
} // end printPrimes
```

a) Prepare a test suite that ensures 100% branch coverage. For each test list in tabular form the branches that it covers.

b) Classify each occurrence of each parameter and local variable as **def** or **use**. Construct tables for each variable identifying they **def-use** paths. From the tables generate tests covering as many **def-use** pairs as possible.

**Grupo 4.** [4 valores]

```
1   /** ************************************************
2    * Finds and prints n prime integers
3    * Jeff Offutt, Spring 2003
4    ***************************************************/
5  private static void printPrimes (int n)
6  {
7    int curPrime; // Value currently considered for primeness
8    int numPrimes; // Number of primes found so far.
9    boolean isPrime; // Is curPrime prime?
10   int [] primes = new int [MAXPRIMES];
11
12   // Initialize 2 into the list of primes.
13   primes [0] = 2;
14   numPrimes = 1;
15   curPrime = 2;
16   while (numPrimes < n)
17   {
18     curPrime++; // next number to consider ...
19     isPrime = true;
20     for (int i = 0; i <= numPrimes-1; i++)
21     { // for each previous prime.
22       if (isDivisible (primes[i], curPrime))
23       { // Found a divisor, curPrime is not prime.
24         isPrime = false;
25         break; // out of loop through primes.
26       }
27     }
28     if (isPrime)
29     { // save it!
30       primes[numPrimes] = curPrime;
31       numPrimes++;
32     }
33   } // End while
34
35   // Print all the primes out.
36   for (int i = 0; i <= numPrimes-1; i++)
37   {
38     System.out.println ("Prime: " + primes[i]);
39   }
40 } // end printPrimes
```

d) Consider the two mutants:

```
19 isPrime = false;
36 for (int i = 1; i <= numPrimes - 1; i++)
```

For each mutant, separately, if possible, find a test case that *does not* reach the mutant.

e) For each mutant, separately, find a test case the kills it.

## *Mock questions*

*(2016 v1)*

**Group 3.** [Graph Coverage. 3.5 points]

Consider the following method that gives the difference between the largest and the smallest integer in an array.

```
 1  public static int range (int[] v) {
 2    if (v == null)
 3      throw new NullPointerException ();
 4    if (v.length == 0)
 5      return 0;
 6    int min = v[0];
 7    int max = v[0];
 8    for (int i = 1; i < v.length; i++)
 9      if (v[i] > max)
10        max = v[i];
11      else if (v[i] < min)
12        min = v[i];
13    return max - min;
14  }
```

**a)** Draw the control flow graph for the method.

**b)** For each node and edge, identify all definitions and all uses.

**c)** Exhibit a path from a definition of variable max to a use of the same variable that is *not* def-clear with respect to max.

**d)** Describe all du-paths.

**e)** For variables max and i only, identify a set of du-paths that satisfy All-Defs Coverage (ADC) but not All-Uses Coverage (AUC). Justify your choice.

**f)** Characterise tests that cover all du-paths identified in **e)**. You do not have to code the tests.

**Group 4.** [Program Mutation Testing. 3 points]

```
 1  public static int range (int[] v) {
 2    if (v == null)
 3      throw new NullPointerException ();
 4    if (v.length == 0)
 5      return 0;
 6    int min = v[0];
 7    int max = v[0];
 8    for (int i = 1; i < v.length; i++)
 9      if (v[i] > max)
10        max = v[i];
11      else if (v[i] < min)
12        min = v[i];
13    return max - min;
14  }
```

| # | Line | Was | Becomes |
|---|---|---|---|
| 1 | 4 | v.length == 0 | v.length != 0 |
| 2 | 8 | i < v.length | i <= v.length |
| 3 | 9 | v[i] > max | v[i] >= max |
| 4 | 13 | max – min | max + min |

**a)** For each mutant induced by the mutations in the table, identify a test that kills the mutant, if possible. Justify your choices.

**b)** Are there functionally equivalent mutants? Which? Justify your answer.