UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

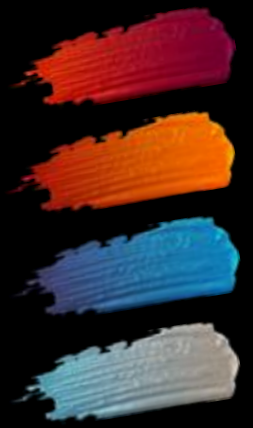
SERLAB
Software Engineering Research

Integrazione e Test di Sistemi Software

Structural testing and Code Coverage
PART 2

Azzurra Ragone

Dipartimento di Informatica - Università degli Studi di Bari
Via Orabona, 4 - 70125 - Bari
Tel: +39.080.5443270 | Fax: +39.080.5442536
serlab.di.uniba.it



Specification based + structural testing: an example

REQS:

Left-pad a string with a specified string. Pad to a size of size.

- `str`—The string to pad out; may be null.
- `size`—The size to pad to.
- `padStr`—The string to pad with. Null or empty is treated as a single space.

The method returns a left-padded string, the original string if no padding is necessary, or null if a null string is input.

EX.

- `str = 'abc'`
- `size = 5`
- `padStr = 'X'`

RESULT: 'XXabc'

Software Engineering Research LABoratory

LeftPad(): implementation

```
public static String leftPad(final String str, final int size,
  String padStr) {

  if (str == null) {
    return null;
  }

  if (padStr==null || padStr.isEmpty()) {
    padStr = SPACE;
  }

  final int padLen = padStr.length();
  final int strLen = str.length();
  final int pads = size - strLen;

  if (pads <= 0) {
    // returns original String when possible
    return str;
  }

  if (pads == padLen) {
    return padStr.concat(str);
  } else if (pads < padLen) {
    return padStr.substring(0, pads).concat(str);
```

If the string to pad is null, we return null right away.

There is no need to pad this string.

If the number of characters to pad matches the size of the pad string, we concatenate it.

If we cannot fit the entire pad string, we add only the part that fits.

Ex:

```
str = 'abc'
size = 2
padStr = 'X'
pads = 2-3<= 0
RESULT = 'abc'
```

LeftPad(): implementation

```
public static String leftPad(final String str, final int size,
  String padStr) {

  if (str == null) {
    return null;
  }

  if (padStr==null || padStr.isEmpty()) {
    padStr = SPACE;
  }

  final int padLen = padStr.length();
  final int strLen = str.length();
  final int pads = size - strLen;

  if (pads <= 0) {
    // returns original String when po
    return str;
  }

  if (pads == padLen) {
    return padStr.concat(str);
  } else if (pads < padLen) {
    return padStr.substring(0, pads).concat(str);
```

If the string to pad is null, we return null right away.

If the pad string is

Ex:

```
str = 'abc'
size = 4
padStr = 'X'
pads = 4-3 = 1
padLen = 1
RESULT = 'Xabc'
```

s to
he
pad string, we concatenate it.

If we cannot fit the entire pad string, we add only the part that fits.

LeftPad(): implementation

```
public static String leftPad(final String str, final int size,
  String padStr) {

  if (str == null) {
    return null;
  }

  if (padStr==null || padStr.isEmpty()) {
    padStr = SPACE;
  }

  final int padLen = padStr.length();
  final int strLen = str.length();
  final int pads = size - strLen;

  if (pads <= 0) {
    // returns original String when p
    return str;
  }

  if (pads == padLen) {
    return padStr.concat(str);
  } else if (pads < padLen) {
    return padStr.substring(0, pads).concat(str);
```

If the string to pad is null, we return null right away.

If the pad string is null or empty, we make it a space.

Ex:

```
str = 'abc'
size = 4
padStr = 'XXX'
pads = 4-3 = 1
padLen = 3
RESULT = 'Xabc'
```

...ers to ...he ...te it.

If we cannot fit the entire pad string, we add only the part that fits.

LeftPad(): implementation

```
  } else {
    final char[] padding = new char[pads];
    final char[] padChars = padStr.toCharArray();

    for (int i = 0; i < pads; i++) {
      padding[i] = padChars[i % padLen];
    }

    return new String(padding).concat(str);
  }
}
```

We have to add the pad string more than once. We go character by character until the string is fully padded.

Ex: pads>padLen

```
str = 'abc'
size = 7
padStr = 'XY'
pads = 7-3 = 4
padLen = 2
RESULT = 'XYXYabc'
```


Let’s start with Specification-based testing

leftPad()

Three inputs:

```
str = ‘abc’
size = 5
padStr = ‘X’
```

One output: `String`

Identify the partitions and the test suite (20 mins)

leftPad(): Identify partitions and boundaries

`str` parameter

Null
Empty string
Non-empty string

`size` parameter

Negative number
Positive number

`padStr` parameter

Null
Empty
Non-empty

`str`, `size` parameters

`size` < len(`str`)
`size` > len(`str`)

Boundaries:

- `size` = 0
- `str` having length 1
- `padStr` having length 1
- `size` being precisely the length of `str`

Software Engineering Research LABoratory

leftPad(): test suite

T1: `str` is null

T2: `str` is empty

T3: negative `size` *(ritorna la stringa in input)*

T4: `padStr` is null *(è trattato come spazio)*

T5: `padStr` is empty *(è trattato come spazio)*

T6: `padStr` has a single character

T7: `size` is equal to the length of `str` *(ritorna la stringa in input)*

T8: `size` is equal to 0 *(ritorna la stringa in input)*

T9: `size` is smaller than the length of `str` *(ritorna la stringa in input)*

Let's write the Junit tests (you can use parameterized tests) – 15 mins

Software Engineering Research LABoratory

leftPad(): specification-based test

```
public class LeftPadTest {

  @ParameterizedTest
  @MethodSource("generator")
  void test(String originalStr, int size, String padString,
   String expectedStr) {
    assertThat(leftPad(originalStr, size, padString))
        .isEqualTo(expectedStr);
  }

  static Stream<Arguments> generator() {
    return Stream.of(
      of(null, 10, "-", null),
      of("", 5, "-", "-----"),
      of("abc", -1, "-", "abc"),
      of("abc", 5, null, "  abc"),
      of("abc", 5, "", " abc"),
      of("abc", 5, "-", "--abc"),
      of("abc", 3, "-", "abc"),
      of("abc", 0, "-", "abc"),
      of("abc", 2, "-", "abc")
    );
  }
}
```

T1 T2 T3 T4 T5 T6 T7 T8 T9

The parameterized test, similar to the ones we have written before

The nine tests we created are provided by the method source.

Software Engineering Research LABoratory

leftPad(): augment the test suite

```
public static String leftPad(final String str, final int size, String padStr) {
    if (str == null) {
        return null;
    }
    if (isEmpty(padStr)) {
        padStr = SPACE;
    }
    final int padLen = padStr.length();
    final int strLen = str.length();
    final int pads = size - strLen;
    if (pads <= 0) {
        return str; // returns original String when possible
    }

    if (pads == padLen) {
        return padStr.concat(str);
    } else if (pads < padLen) {
        return padStr.substring(0, pads).concat(str);
    } else {
        final char[] padding = new char[pads];
        final char[] padChars = padStr.toCharArray();
        for (int i = 0; i < pads; i++) {
            padding[i] = padChars[i % padLen];
        }
        return new String(padding).concat(str);
    }
}
```

The red lines indicate parts of the code that are still not covered!

leftPad(): augment the test suite

We did not exercise `padStr` being smaller (3), greater (2), or equal (1) to the remaining space in `str`.

1) pads==padLen

```
str = 'abc'
size = '5'
padStr = 'XY'
Output= 'XYabc'
```

2) pads < padLen

```
str = 'abc'
size = '5'
padStr = 'XYXY'
Output= 'XYabc'
```

3) pads > padLen

```
str = 'abc'
size = '5'
padStr = 'X'
Output= 'XXabc'
```


Software Engineering Research LABoratory

leftPad(): augment the test suite

New test cases:

T10: the length of `padStr` is equal to the remaining spaces in `str`.
T11: the length of `padStr` is greater than the remaining spaces in `str`.
T12: the length of `padStr` is smaller than the remaining spaces in `str` (this test may be similar to T6).

```
of( ...arguments: "abc", 5, "--", "--abc"), // T10
of( ...arguments: "abc", 5, "---", "--abc"), // T11
of( ...arguments: "abc", 5, "-", "--abc") // T12
```

Do we miss something?

Software Engineering Research LABoratory

leftPad(): augment the test suite

```
    public static String leftPad(final String str, final int size, String padStr) {
        if (str == null) {
            return null;
        }
        if (isEmpty(padStr)) {
            padStr = SPACE;
        }
        final int padLen = padStr.length();
        final int strLen = str.length();
        final int pads = size - strLen;
        if (pads <= 0) {
            return str; // returns original String when possible
        }

        if (pads == padLen) {
            return padStr.concat(str);
        } else if (pads < padLen) {
            return padStr.substring(0, pads).concat(str);
        } else {
            final char[] padding = new char[pads];
            final char[] padChars = padStr.toCharArray();
            for (int i = 0; i < pads; i++) {
                padding[i] = padChars[i % padLen];
            }
            return new String(padding).concat(str);
        }
    }
```


leftPad(): extra test

```
public static String leftPad(final String str, final int size,
  String padStr) {

  if (str == null) {
    return null;
  }

  if (padStr==null || padStr.isEmpty()) {
    padStr = SPACE;
  }

  final int padLen = padStr.length();
  final int strLen = str.length();
  final int pads = size - strLen;

  if (pads <= 0) {
    // returns original String when possible
    return str;
  }

  if (pads == padLen) {
    return padStr.concat(str);
  } else if (pads < padLen) {
    return padStr.substring(0, pads).concat(str);
```

If the string to pad is null, we return null right away.

If the pad string is null or empty, we make it a space.

There is need to this stri

Ex:

```
str = 'sometext'
size = 5
padStr = '-'
pads = 5-8<= 0
RESULT = 'sometext'
```

If we cannot fit the entire pad string, we add only the part that fits.

Software Engineering Research LABoratory

leftPad(): extra test

New test:

```
@Test
void sameInstance() {
  String str = "sometext";
  assertThat(leftPad(str, 5, "-")).isSameAs(str);
}
```

Ex:

```
str = 'sometext'
size = 5
padStr = '-'
pads = 5-8<= 0
RESULT = 'sometext'
```

Boundary testing

Analyzing the `if` statements in the `leftPad` program (on & off points):

- `if(pads<=0)`
  - The on point is 0 (it evaluates the expression to true). The off point is the nearest point to the on point that makes the expression evaluate to false. In this case, given that `pads` is an integer, is 1.
- `if (pads == padLen)`
  - The on point is `padLen`. Given the equality and that `padLen` is an integer, we have two off points: one that happens when `pads == padLen - 1` and another that happens when `pads=padLen+1`.
- `if (pads < padLen)`
  - The on point is again `padLen`. The on point evaluates the expression to false. The off point is, therefore, `pads == padLen - 1`.

Software Engineering Research LABoratory

Structural testing + Code coverage: are they enough?

Structural testing + Code coverage: are they enough?

Exercise A (20 mins): This program counts the number of "clumps" in an array. A clump is a sequence of the same element with a length of at least 2. (ex. [2,3,3,3,4]). Write the minimum number of tests to achieve 100% code coverage.

`nums`—The array must be non-null and length > 0; the program returns 0 if any pre-condition is violated.

```
public static int countClumps(int[] nums) {
  if (nums == null || nums.length == 0) {
    return 0;
  }
  int count = 0;
  int prev = nums[0];
  boolean inClump = false;
  for (int i = 1; i < nums.length; i++) {
    if (nums[i] == prev && !inClump) {
      inClump = true;
      count += 1;
    }
    if (nums[i] != prev) {
      prev = nums[i];
      inClump = false;
    }
  }
  return count;
}
```

If null or empty (pre-condition), return 0 right away.

If the current number is the same as the previous number, we have identified a clump.

If the current number differs from the previous one, we are not in a clump.

Software Engineering Research LABoratory

Solution to Exercise A

Exercise A (20 mins): given this program that found “clumps” in an array (ex. [2,3,3,3,4]), write the minimum number of tests to achieve 100% code coverage

- T1: an empty array
- T2: a null array
- T3: an array with a single clump of three elements in the middle (for example,
- [2,3,3,3,4])
- T4: an array with a single element

Solution to Exercise A

Exercise A (20 mins): given this program that found “clumps” in an array (ex. [2,3,3,3,4]), write the minimum number of tests to achieve 100% code coverage

```
@ParameterizedTest
@MethodSource("generator")
void testClumps(int[] nums, int expectedNoOfClumps) {
  assertThat(Clumps.countClumps(nums))
        .isEqualTo(expectedNoOfClumps);
}

static Stream<Arguments> generator() {
  return Stream.of(
    of(new int[]{}, 0), // empty
    of(null, 0), // null
    of(new int[]{1,2,2,2,1}, 1), // one clump
    of(new int[]{1}, 0) // one element
  );
}
```

The four test cases we defined

Solution to Exercise A

Exercise A (20 mins): given this program that found "cl

[2,3,3,3,4]), write the minimum number of tests to achie

We miss several cases:
1- multiple clumps in the array
2- clump as last item
3- clump as first item

```
@ParameterizedTest
@MethodSource("generator")
void testClumps(int[] nums, int expectedNoOfClumps) {
  assertThat(Clumps.countClumps(nums))
        .isEqualTo(expectedNoOfClumps);
}

static Stream<Arguments> generator() {
  return Stream.of(
    of(new int[]{}, 0), // empty
    of(null, 0), // null
    of(new int[]{1,2,2,2,1}, 1), // one clump
    of(new int[]{1}, 0) // one element
  );
}
```

The four test cases we defined

Structural testing + Code coverage: excercise

Exercise B: This program counts the number of "clumps" in an array. A clump is a sequence of the same element with a length of at least 2. (ex. [2,3,3,3,4]). Write the minimum number of tests to achieve 100% code coverage.

`nums`—The array must be non-null and length > 0; the program returns 0 if any pre-condition is violated.

Do effective software testing!

```
public static int countClumps(int[] nums) {
  if (nums == null || nums.length == 0) {
    return 0;
  }
  int count = 0;
  int prev = nums[0];
  boolean inClump = false;
  for (int i = 1; i < nums.length; i++) {
    if (nums[i] == prev && !inClump) {
      inClump = true;
      count += 1;
    }
    if (nums[i] != prev) {
      prev = nums[i];
      inClump = false;
    }
  }
  return count;
}
```

If null or empty (pre-condition), return 0 right away.

If the current number is the same as the previous number, we have identified a clump.

If the current number differs from the previous one, we are not in a clump.

Software Engineering Research LABoratory

Structural testing recap

- They are used to augment specification-based testing
- Code coverage helps in identify part of the code not exercised be the test suite
- Identify partitions you may miss
- You may purposefully decide not to cover some lines
- Don’t game the metric
- **100% coverage** does not necessarily mean the system is properly tested
- Having very **low coverage** *does* mean your system is *not* properly tested
- Coverage criteria trade-off (money – accuracy)

Mutation testing

- In MT we purposefully insert a bug in a code to see if the test suite breaks:
  - If it does -> Ok, the test suite *kills the mutant*
  - If it does not -> we should improve our test suite (the *mutant survives*)
- ***Coupling effect***: a complex bug is caused by a combination of many small bugs, if your test suite can catch simple bugs, it will also catch the more complex ones.

Software Engineering Research LABoratory

Example of mutators

- *Conditionals boundary*—Relational operators such as < and <= are replaced by other relational operators.
- *Increment* —It replaces i++ with i-- and vice versa.
- *Invert negatives* —It negates variables: for example, i becomes -i.
- *Math operators*—It replaces mathematical operators: for example, a plus becomes a minus.
- *True returns* —It replaces entire boolean variables with true.
- *Remove conditionals* —It replaces entire if statements with a simple if(true) {...}.

Software Engineering Research LABoratory

Surviving mutants

- Evaluate each surviving mutants, as some may have no sense
- Mutation testing tools do not know your code—they simply mutate it, sometimes they create mutants that are not useful.
- Ex: `pads = size - strLen`, size is mutated in `size++` (ignore this mutant)

Mutation testing: recap

- MT like CC is useful to augment your test suite
- MT could be quite expensive
- Run it for a small set of classes to have valuable insight about what to test further

Pitest coverage Report

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
1	95%	19/20	89%	17/19	89%	17/19

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
ch3	1	95%	19/20	89%	17/19	89%	17/19

Software Engineering Research LABoratory

Pitest coverage Report

Pit Test Coverage Report

Package Summary

ch3

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
1	95%	19/20	89%	17/19	89%	17/19

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
LeftPadUtils.java	95%	19/20	89%	17/19	89%	17/19

Pitest coverage Report

Mutations

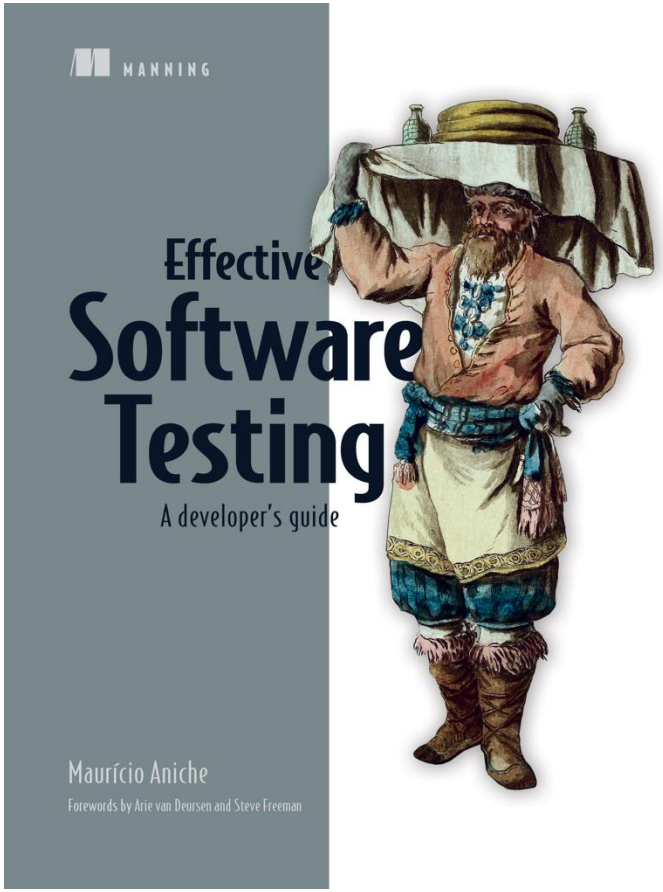
8	1. negated conditional → KILLED 2. negated conditional → KILLED 3. replaced boolean return with true for ch3/LeftPadUtils::isEmpty → KILLED
23	1. negated conditional → KILLED
24	1. replaced return value with "" for ch3/LeftPadUtils::leftPad → KILLED
26	1. negated conditional → KILLED
31	1. Replaced integer subtraction with addition → KILLED
32	1. changed conditional boundary → SURVIVED 2. negated conditional → KILLED
33	1. replaced return value with "" for ch3/LeftPadUtils::leftPad → KILLED
36	1. negated conditional → KILLED
37	1. replaced return value with "" for ch3/LeftPadUtils::leftPad → KILLED
38	1. changed conditional boundary → SURVIVED 2. negated conditional → KILLED
39	1. replaced return value with "" for ch3/LeftPadUtils::leftPad → KILLED
43	1. changed conditional boundary → KILLED 2. negated conditional → KILLED
44	1. Replaced integer modulus with multiplication → KILLED
46	1. replaced return value with "" for ch3/LeftPadUtils::leftPad → KILLED

Homework n.2

Understand how Pitest works.
https://pitest.org/

Reference book:

Effective Software Testing. A developer's guide. Mauricio Aniche. Ed. Manning. (**Chapter 3**)



References

- Pitest (PIT) https://pitest.org/

Azzurra Ragone

Department of Computer Science- Floor VI – Room 616
Email: azzurra.ragone@uniba.it