SENG 275

SOFTWARE TESTING

DR. NAVNEET KAUR POPLI

DEPT. OF ELECTRICAL AND COMPUTER ENGINEERING

University of Victoria

TESTING TECHNIQUES- EQUIVALENCE CLASS PARTITIONING

University of Victoria

Testing Techniques

We look at different techniques to test a software system effectively, rigorously, and systematically, and how to automate as many steps as possible along the way.

1. **Specification-based testing**: Techniques to derive tests from textual functional requirements. Two types are the *category/partition method* and *equivalence partitioning* (Black box testing).
2. **Boundary testing**: Deriving tests that exercise the boundaries of our requirement (Black box testing).
3. **Structural testing**: Test cases based on the structure of the source code (White box testing).
4. **Model-based testing**: Leveraging more formal documentation such as state machines and decision tables to derive tests.
5. **Design-by-contracts**: Devising explicit contracts for methods and classes to ensure that they behave correctly when these contracts are (and are not) met.
6. **Property-based testing**: Deriving properties of the system (similar to contracts) and using them to automatically generate test cases.

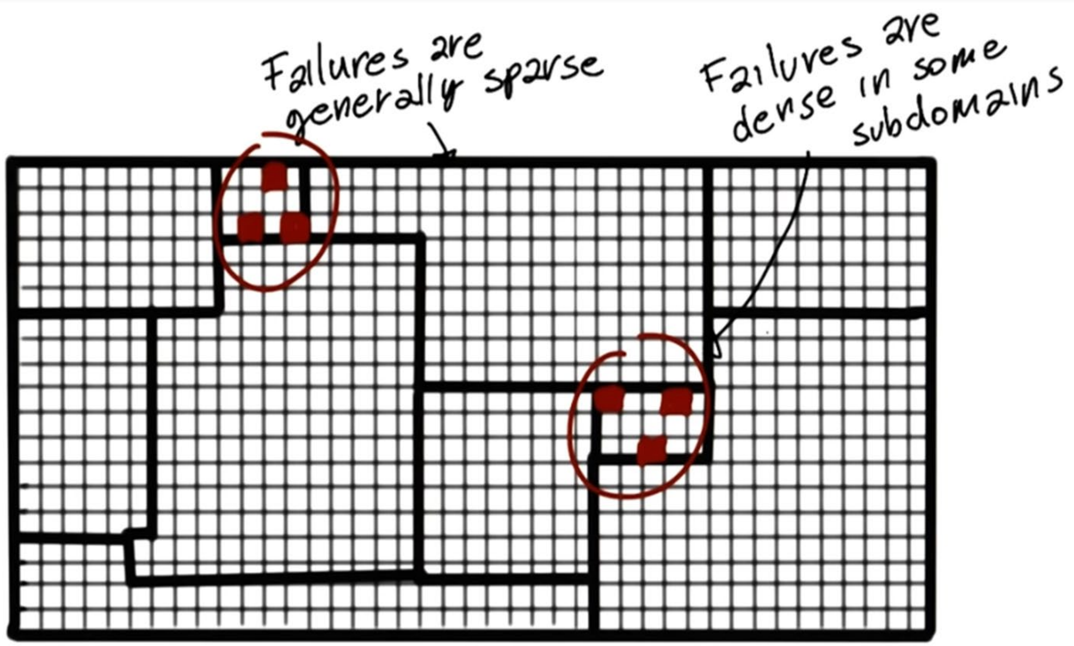
University of Victoria

Specification-based testing

- These techniques use the program requirements—such as agile user stories or UML use cases—as testing input.
- All requirements include three parts:
  - First is what the program/method must do: its business rules.
  - Input domain
  - Output domain
- Input and output domains determine your partitions.
- **Equivalence partitioning** divides **input domain** into equivalent classes from which test cases can be derived.
- BVA also works on the input domain.

Random testing

- Advantages:
  - –Pick inputs uniformly
  - –All inputs considered equal
  - –No designer bias
- Disadvantage: we may be looking at the wrong places



Partition testing

- A domain is naturally partitioned.
- So identify partitions in the domain and then
- Identify inputs from each partition.
- Partitioning can be done using:
  - Equivalence partitioning method
  - Category partitioning

EXAMPLE 1
LEAP YEAR

Specification-Based Testing-**black box testing**

- **Requirement: Leap year**
- Given a specific year as an input, the program should return true if the provided year is a leap year and false if it is not.
- A year is a leap year if:
  - The year is divisible by 4 and is not divisible by 100;
  - The year is divisible by 4 and is divisible by 100 and is divisible by 400

4 ✗			FALSE
4✓	100 ✗		TRUE
4✓	100✓	400✓	TRUE
4✓	100✓	400 ✗	FALSE

Try it…………

- Identify all equivalent partitions of the input domain.
- Devise test cases to execute one value each from each partition to check whether the program is working fine.

Partitioning the input space

- By looking at the requirements above, we can derive the following **partitions**:
  - **Year is divisible by 4, but not divisible by 100** = leap year, **TRUE (2016)**
  - **Year is divisible by 4, divisible by 100, divisible by 400** = leap year, **TRUE(2000)**
  - **Not divisible by 4** = not leap year, **FALSE(1441)**
  - **Divisible by 4, divisible by 100, but not divisible by 400** = not leap year, **FALSE(1900)**
- Note how each class above exercises the program in different ways.

University of Victoria

Equivalence partitioning

- The partitions above are not test cases that we can implement directly because each partition might be instantiated by an **infinite number of inputs**.
- For example, for the partition "year not divisible by 4", there are infinitely many numbers that are not divisible by 4 which we could use as concrete inputs to the program. So how do we know which concrete input to instantiate for each of the partitions?
- We assume that, if the program behaves correctly for one given input, it will work correctly for all other inputs from that class. This idea of **inputs being equivalent to each other** is called **equivalence partitioning**. Thus, it does not matter which precise input we select and one test case per partition will be enough.

University of Victoria

Leap year program

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

 }}
```

4 ✗			FALSE
4✓	100 ✗		TRUE
4✓	100✓	400✓	TRUE
4✓	100✓	400 ✗	FALSE

University of Victoria

```java
public class LeapYearTests {

 LeapYear leapYear = new LeapYear();

  @Test
  public void notDivisibleBy4() {
    boolean leap = leapYear.isLeapYear(1441);
    assertFalse(leap);
  }
  @Test
  public void divisibleBy4_notDivisibleBy100() {
    boolean leap = leapYear.isLeapYear(2016);
    assertTrue(leap);
  }
  @Test
  public void divisibleBy4_100_400() {
    boolean leap = leapYear.isLeapYear(2000);
    assertTrue(leap);
  }
  @Test
  public void divisibleBy4_and_100_not_400() {
    boolean leap = leapYear.isLeapYear(1900);
    assertFalse(leap);
  }
}
```

University of Victoria

EXAMPLE 2
AGE

University of Victoria

Equivalent Partitioning

INVALID CLASS 1	VALID CLASS	INVALID CLASS 2
AGE<=17	AGE: 18-60	AGE>=61
5	35	100

- **Specification**:

Age can be 18-60.

- Valid Input: 18 – 60
- Invalid Input: less than or equal to 17 (<=17), greater than or equal to 60 (>=61)

We have one valid and two invalid conditions here.

- **Valid Class**: 18 – 60 = Pick any one input test data from 18 – 60 say 35, 26, 47.
- **Invalid Class 1**: <=17 = Pick any one input test data less than or equal to 17 say 10, 5, 15 etc.(doesn’t need to be on the boundary)
- **Invalid Class 2**: >=61= Pick any one input test data greater than or equal to 61 say 84, 71, 100.

University of Victoria

EXAMPLE 3
MOBILE NUMBER

University of Victoria

Find all test cases.

MOBILE NUMBER Enter Mobile No. *Must be 10 digits

Find all test cases.

MOBILE NUMBER Enter Mobile No. *Must be 10 digits

VALID CLASS	VALID CLASS	INVALID CLASS	INVALID CLASS
NUMBER=10	NUMBER=10	NUMBER>=11	NUMBER<=9
9811546523	8877564321	8766554423109	100

EXAMPLE 4
SQUARE

Another example

- **Specification**: Suppose there is a program ‘Square’ which takes ‘x’ as an input and prints the square of ‘x’ as output.
- The range of ‘x’ is from 1 to 100.
- Find **input domain equivalence classes** for the program ‘Square’.

University of Victoria

Solution

- **Specification**: Suppose there is a program 'Square' which takes 'x' as an input and prints the square of 'x' as output.
- The range of 'x' is from 1 to 100.
- The **input domain equivalence classes** for the program 'Square' are given as:

(i) $I_1 = \{ 1 \leq x \leq 100 \}$ (Valid input range from 1 to 100)

(ii) $I_2 = \{ x < 1 \}$ (Any invalid input where x is less than 1)

(iii) $I_3 = \{ x > 100 \}$ (Any invalid input where x is greater than 100)

Test Case	Input x	Expected Output
I_1	0	Invalid Input
I_2	50	2500
I_3	101	Invalid Input

REFLECTING ON OUTPUTS

Reflecting on the outputs (Refer to textbook Pg. 31-37)

- Requirement: Develop a method that searches for substrings between two tags in a given string and returns all the matching substrings.
- Example: if str = “a**x**ca**y**ca**z**c”, open = “a”, and close = “c”, the output will be an array containing [“x”, “y”, “z”]. This is the case because the “a<something>c” substring appears three times in the original string: the first contains “x” in the middle, the second “y,” and the last “z.”
- The best way to ensure that this method works properly would be to **test all the possible combinations of inputs and outputs**.

University of Victoria

Reflecting on the outputs (Refer to textbook Pg. 31-37)

A systematic way to do such an exploration is to think of the following:

- Each input individually: “What are the possible classes of inputs I can provide?”
- Each input in combination with other inputs: “What combinations can I try between the open and close tags?”
- The different classes of output expected from this program: “Does it return arrays? Can it return an empty array? Can it return nulls?”
- The possible outputs are:
  - –Array of strings (output)
    - Null array
    - Empty array
    - Single item
    - Multiple items
  - –Each individual string (output)
    - Empty
    - Single character
    - Multiple characters
- Reflecting on the outputs may help you see an input case that you did not identify before.

University of Victoria