Software Testing

Testing

- Testing = Executing software with test input and checking whether it does what we want
- Example:
  - We have the program “division.exe”
  - What we want: the program should print the quotient of two numbers
- Let’s test it:

```
> division.exe  6  3
2                                  good✓
> division.exe  12  3
4                                  good✓
> division.exe  4  0
Exception in line 5: Division by zero   bad?
```

Testing (2)

- Obviously, we can only write and test a program if we know what it should do
- Different possibilities to specify what a program should do
  1. Formal specification:

$$division(a,b) = \begin{cases} \frac{a}{b}, & if \;\; b \neq 0 \\ error, & otherwise \end{cases}$$

  2. Specification document:

"The program should print the quotient of …"

  3. User requirement:

"The user wants a calculator"

Functional vs non-functional tests

- Tests can be done to check whether a program satisfies functional requirements or non-functional requirements
- Examples for **functional requirements**:
  - "The program should calculate a/b"
  - "The program should sort a list"
  - "The program should print all prime numbers"
  - …
- Examples for **non-functional requirements**:
  - "The program should have complexity $O(n)$“
  - “The program should be written in Java”
  - "The program should be easy to use"
  - "The program should not contain a virus"
  - …

Finding input values for tests

- To test whether a program fulfills the requirements, we have to test it with input values from its *input domain*
  - In our "division.exe" example, the input domain is $\mathbb{Z} \times \mathbb{Z}$
- Do we have to test all possible input values?
  - Hopefully not! We expect that if division.exe works for $a = 5, \mathrm{b} = 7$ it will also work for $a = 12,\ b = 25$
- So, our approach to find useful input values for our tests is:
  1. Look at the input domain of the program

     $$\mathbb{Z} \times \mathbb{Z}$$

  2. Split the input domain into interesting sub-domains

     Two sub-domains: $a \in \mathbb{Z}, b \in \mathbb{Z} \setminus \{0\}$ and $a \in \mathbb{Z}, b = 0$

  3. Choose test input values from each sub-domain:

     $a \in \mathbb{Z}, b \in \mathbb{Z} \setminus \{0\} \rightarrow a = 5, b = 7$
     $a \in \mathbb{Z}, b = 0 \rightarrow a = 3, b = 0$

Quiz (Answer on the next slide)

- Let‘s say you want to test the following method:

```
int[] sortArray(int[] array)
```

a) What is the input domain?

b) What are possible sub-domains of the input domain?

Answer

- Let‘s say you want to test the following method:

```
int[] sortArray(int[] array)
```

a) What is the input domain?

$$\mathbb{Z}^n \quad where\ n \in \mathbb{N}\ is\ the\ length\ of\ the\ array$$

b) What are possible sub-domains of the input domain?

1. Empty array ($n = 0$)
2. Array with one element ($n = 1$)
3. Unsorted array with $n > 1$
4. Array already sorted in ascending order with $n > 1$
5. Array already sorted in descending order with $n > 1$

It’s always good to have disjoint sub-domains that cover the entire input domain!

We can test a program at different levels

- **Unit testing** = testing a single method

  "Does the method give the correct result?"

- **Module testing** = testing a module (in Java: module ≈ class)

  "Does the class work correctly? Does it have the required methods?"

- **Integration testing** = testing several modules together

  "Do the modules work together correctly? Do all modules have the right methods? Do the modules use the methods of the other modules correctly?"

- **System testing** = testing the entire system or program

  "Does the system follow the specification?"

- **Acceptance testing** = testing at the customer

  "Does software do what the user wants?"

Who does the tests?

- Unit and module tests

The author of the unit or module

- Integration tests
  - Done by the developer team
- System test
  - Done by the test team
- Acceptance test
  - Done by the customer or by people who know what the customer needs (“domain knowledge”)

Test levels

- Unit testing
- Module testing
- Integration testing
- System testing
- Acceptance testing

Test difficulty

Tests can be done very early (as soon as you have written a method) and frequently

Tests are difficult: The software has to be installed, users have to “play“ with it,…

If you find a bug…

Very easy to fix

Very expensive to fix