Tampereen yliopisto
Tampere University

Software testing

9.9.2018

What is testing?

- Testing **produces information**:
  - Untested code is assumed to be broken
  - Testing brings information about the quality of the code to make better decisions
- Why test a program?

Why should I test?

- Testing indicates if
  - the program does what it should not do
  - the program does not do what it should do
  - the program works against the requirements (which one is wrong?)
  - the program is difficult to understand or use, is slow, or works in an unexpected way
- Everything cannot be tested

Aim: to reveal errors

- Assumption: program always has errors, the task is to find them
- Starting point: a successful test causes a failure in program execution
  - ➢ eliminating an error increases quality
  - ➢ finding out the origin of an error: root reason and technical debt

Essential terms

- **Error**: a deviation from the specification
- **Fault, defect**: caused by execution of erroneous code or an unimplemented functionality
- **Failure**: an externally observable event in the functionality, due to a fault

- **Bug**: can stand for any of the previous terms

Different ways to test

- **Dynamic testing**: executing the program with suitable input
- **Static testing**: inspecting the source code and/or documentation
- **Positive testing:** “happy case” tests, trying to ensure that the program does what it should do
- **Negative testing:** “unhappy case” tests, i.e. cases which are not described in the specification, erroneous cases etc.

How to test?

Kuva: Zhao! **(CC BY 2.0)**

A good test case

- A small test for program functionality:
  - What to test? E.g. division operator of a calculator
  - A good input? E.g. division by zero
  - Expected result? E.g. an error message, something else than program crash
- Designed either before execution or “on the fly”

The structure of a test case

1. Set-up
   - put the system in the state needed to run the test
2. Execution
   - run the system and capture all output
3. Evaluation
   - compare the results to the expected results and judge
4. Clean-up
   - restore the system to the pre-test state

Unit testing

Testing a program

- Aim: self-testing code
  - sufficient tests form a part of a working program
  - continuous reliability on that bugs will be found
  - avoiding regression

Kuva: Martin Fowler

Unit testing in practice

- What to test?
- How to test?
- Who tests?

Kuva: sleepymyf (CC BY-NC-ND 2.0)

Programmer as tester

- A good programmer can test their own code
  - programmer is responsible for testing the program units implemented by himself/herself
  - often also quality assurance tasks for other programmers' code

Kuva: sleepymyf (CC BY-NC-ND 2.0)

Unit testing

- Part of unit implementation: test the implementation as early as possible
- Use interface as a view (encapsulation)
- Test Driven Development:
  - create a test to be run automatically (and run it)
  - write code and run the test
  - fix and refactor the code

Refactoring

- Does not change the external behavior
- Improves the structure and non-functional attributes
  - decreases technical debt
  - may resolve hidden, undiscovered bugs
- Comments

Comments

```
/* if allocation flag is zero */
if ( alloc_flag == 0 )
```

```
/* if allocating new member */
if ( alloc_flag == 0 )
```

```
/* if allocating new member */
if ( alloc_flag == NEW_MEMBER )
```

(useless)

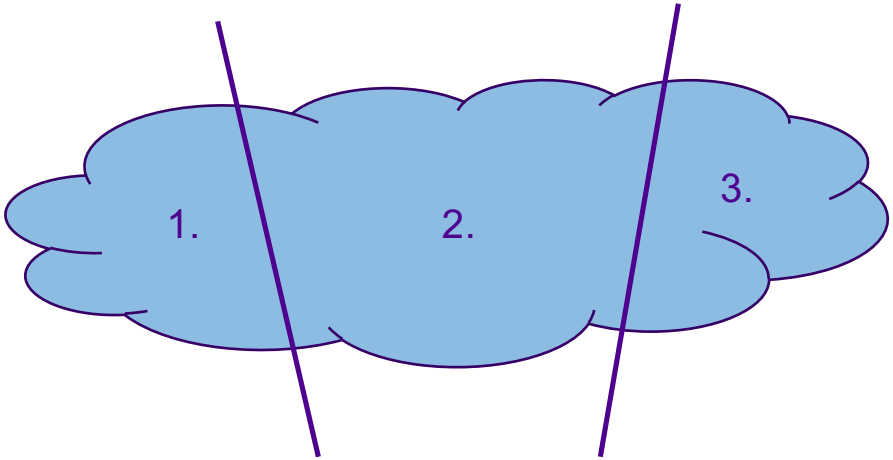
Tampereen yliopisto
Tampere University

Testing boundary values

- To be tested:
  - boundary values of parameters and return values
  - boundary values of loops
  - boundary values related to data structures

Equivalence Partitioning

- Division of input into equivalence classes
  - Select a condition from the input
  - Divide into classes
  - Basic division: allowed and not allowed



Tampereen yliopisto
Tampere University

Implementing unit testing

- Small, clear test functions and simple checks in them
  - assertions of a test framework in **test code**

```
TEST_TYPE test_square_root() {
    double result = my_sqrt(x);
    ASSERT_TRUE((result * result) == x);
    // Can you find a small bug in test code?
    // for simplicity ...
}
```

Unit testing in practice

- Test code is code that should be
  - documented
  - tested
  - maintained

Unit testing

1. What is extremely important for a method
2. Test the most usual cases
3. Be creative
4. Concentrate on the interface
5. Make tests as simple as possible
6. Use test framework

Integration testing

- After unit testing, units will be integrated into large wholes
- Version control enables each developer to see the whole program
- Test automation and continuous integration

Tampereen yliopisto
Tampere University

Continuous integration

Fig: Jenkins

Created by Swen-Peter Ekkebus
from Noun Project

New code

Version control

Autom.

CI

- Build
- Unit
- Integration

What to test?

- Equivalence partition
- Boundary value analysis
- Static analysis

BE CREATIVE!

Errors found?

- Syntax errors
- Uninitialized variables
- Return values not used
- Erroneous use of pointers
- Same code in several places, dead code
- Problems in maintenance and porting