



Object-Oriented Software Construction

Bertrand Meyer

Lecture 17: Testing Object-Oriented Software

Ilinca Ciupa



A (rather unorthodox) introduction (1)

3

(Geoffrey James – *The Zen of Programming*, 1988)

“Thus spoke the master: “Any program, no matter how small, contains bugs.”

The novice did not believe the master’s words. “What if the program were so small that it performed a single function?” he asked.

“Such a program would have no meaning,” said the master, “but if such a one existed, the operating system would fail eventually, producing a bug.”

But the novice was not satisfied. “What if the operating system did not fail?” he asked.



Agenda for today

2

- Introduction ←
- What is software testing?
- Why do we need software testing?
- Testing terminology
- Black box vs. white box testing
- Testing strategy
- Test automation
- Contracts and tests
- TestStudio



A (rather unorthodox) introduction (2)

4

“There is no operating system that does not fail,” said the master, “but if such a one existed, the hardware would fail eventually, producing a bug.”

The novice still was not satisfied. “What if the hardware did not fail?” he asked.

The master gave a great sigh. “There is no hardware that does not fail”, he said, “but if such a one existed, the user would want the program to do something different, and this too is a bug.”

A program without bugs would be an absurdity, a nonesuch. If there were a program without any bugs then the world would cease to exist.”

Agenda for today

5

- Introduction
- What is software testing? ←
- Why do we need software testing?
- Testing terminology
- Black box vs. white box testing
- Testing strategy
- Test automation
- Contracts and tests
- TestStudio

What does testing involve?

7

- Determine **which parts** of the system you want to test
- Find **input values** which should bring significant information
- **Run** the software on the input values
- **Compare** the produced **results** to the expected ones
- (Measure execution characteristics: time, memory used, etc)

A definition

6

"Software testing is the execution of code using combinations of input and state selected **to reveal bugs.**"

"Software testing [...] is **the design and implementation of a special kind of software system**: one that exercises another software system with **the intent of finding bugs.**"

Robert V. Binder, *Testing Object-Oriented Systems: Models, Patterns, and Tools* (1999)

Some more insight into the situation

8

"Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence."

Edsger Dijkstra, *Structured Programming* (1972)

What testing can do for you: find bugs

What testing cannot do for you: prove the absence of bugs

What testing is not

9

- Testing \neq **debugging**
 - When testing uncovers an error, debugging is the process of removing that error
- Testing \neq program **proving**
 - Formal correctness proofs are mathematical proofs of the equivalence between the specification and the program

Here's a thought...


11

- *"Imagine if every Thursday your shoes exploded if you tied them the usual way. This happens to us all the time with computers, and nobody thinks of complaining."*

Jef Raskin, Apple Computer, Inc.

Agenda for today

10

- Introduction
- What is software testing?
- Why do we need software testing? 
- Testing terminology
- Black box vs. white box testing
- Testing strategy
- Test automation
- Contracts and tests
- TestStudio

More thoughts (?!)

12

- *"I know not a single less relevant reason for an update than bug fixes. The reason for updates is to present new features."* (Bill Gates in "Focus" magazine)
- *"Microsoft programs are generally bug free ... 99.9% [of calls to the Microsoft hot line] turn out to be user mistakes."* (Bill Gates in "Focus" magazine)

To test or not to test (1)

13

- Users accept bugs as a matter of fact
- But:
 - Faulty software kills people – several examples available from the medical world
 - Faulty software produces huge costs
 - e.g. DS-1 Orion 3 Galileo Titan 4B (1999) – aggregate cost: \$1.6 billion (May 2002 NIST report)
 - Same NIST report: \$60 billion/year cost of software errors in the US only
 - Faulty software leads to loss of data
 - e.g. any of the above

...to list only a few consequences

Agenda for today

15

- Introduction
- What is software testing?
- Why do we need software testing?
- Testing terminology ←
- Black box vs. white box testing
- Testing strategy
- Test automation
- Contracts and tests
- TestStudio

To test or not to test (2)

14

- What is the first example of a software failure that you can think of?



(In)famous blue screen of death (BSOD)

Common abbreviations

16

- IUT – implementation under test
- MUT – method under test
- OUT – object under test
- CUT – class/component under test
- SUT – system under test

Bug-related terminology

17

- **Failure** – manifested inability of the IUT to perform a required function
 - Evidenced by:
 - Incorrect output
 - Abnormal termination
 - Unmet time or space constraints
- **Fault** – incorrect or missing code
 - Execution may result in a failure
- **Error** – human action that produces a software fault
- **Bug** – error or fault

Dijkstra's criticism of the word "bug"


19

We could, for instance, begin with cleaning up our language by no longer calling a bug "a bug" but by calling it **an error**. It is much more honest because it squarely **puts the blame where it belongs**, with the programmer who made the error. The animistic metaphor of **the bug that maliciously sneaked in while the programmer was not looking** is intellectually dishonest as it is a disguise that the error is the programmer's own creation. The nice thing about this simple change of vocabulary is that it has such a profound effect. While, before, a program with only one bug used to be "**almost correct**", afterwards a program with an error is just "**wrong**"...

E. W. Dijkstra, *On the cruelty of really teaching computer science*
(December 1989)

Hopper's bug

18

9/9
0800 Antan started
7000 stopped - antan ✓
1300 (030) MP-MC 1.13047645
020 PPO 2 2.13047645
0300 correct 2.13047645
Relays 602 on 033 failed speed speed test
in test 10.000 test
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Multi-Adder Test.
1545  Relay #70 Panel F
Moth in Relay.
First actual case of bug being found.
1600 Antan started.
1700 closed form.

Testing scope

20

- **Unit test** – scope: typically a relatively small executable
- **Integration test** – scope: a complete system or subsystem of software and hardware units
 - Exercises interfaces between units to demonstrate that they are collectively operable
- **System test** – scope: a complete integrated application
 - Focuses on characteristics that are present only at the level of the entire system
 - Categories:
 - Functional
 - Performance
 - Stress or load

Intent (1)

21

- **Fault-directed testing** – intent: reveal faults through failures
 - Unit and integration testing
- **Conformance-directed testing** – intent: to demonstrate conformance to required capabilities
 - System testing
- **Acceptance testing** – intent: enable a user/customer to decide whether to accept a software product

Components of a test

23

- **Test case** – specifies:
 - The state of the IUT and its environment before test execution
 - The test inputs
 - The expected result
- **Expected results** – what the IUT should produce:
 - Returned values
 - Messages
 - Exceptions
 - Resultant state of the IUT and its environment
- **Oracle** – produces the results expected for a test case
 - Can also make a pass/no pass evaluation

Intent (2)

22

- **Regression testing** - Retesting a previously tested program following modification to ensure that faults have not been introduced or uncovered as a result of the changes made
- **Mutation testing** – Purposely introducing faults in the software in order to estimate the quality of the tests

Finding test inputs

24

Partition testing

- **Partition** – divides the input space into groups which hopefully have the property that any value in the group will produce a failure if a bug exists in the code related to that partition
- **Examples of partition testing:**
 - **Equivalence class** – a set of input values so that if any value in the set is processed correctly (incorrectly) then any other value in the set will be processed correctly (incorrectly)
 - **Boundary value analysis**
 - **Special values testing**

Test execution

25

- **Test suite** – collection of test cases
- **Test driver** – class or utility program that applies test cases to an IUT
- **Stub** – partial, temporary implementation of a component
 - May serve as a placeholder for an incomplete component or implement testing support code
- **Test harness** – a system of test drivers and other tools to support test execution

Black box vs white box testing (1)

27

Black box testing	White box testing
Uses no knowledge of the internals of the SUT	Uses knowledge of the internal structure and implementation of the SUT
Also known as responsibility-based testing and functional testing	Also known as implementation-based testing or structural testing
Goal: to test how well the SUT conforms to its requirements (Cover all the requirements)	Goal: to test that all paths in the code run correctly (Cover all the code)

Agenda for today

26

- Introduction
- What is software testing?
- Why do we need software testing?
- Testing terminology
- Black box vs. white box testing ←
- Testing strategy
- Test automation
- Contracts and tests
- TestStudio

Black box vs white box testing (2)

28

Black box testing	White box testing
Uses no knowledge of the program except its specification	Relies on source code analysis to design test cases
Typically used in integration and system testing	Typically used in unit testing
Can also be done by user	Typically done by programmer

White box testing

29

- Allows you to look inside the box
- Some people prefer “glass box” or “clear box” testing

Basic measures of code coverage

31

- **Statement coverage** – reports whether each executable statement is encountered
 - Disadvantage: insensitive to some control structures
- **Decision coverage** – reports whether boolean expressions tested in control structures evaluate to both true and false
 - Also known as **branch coverage**
- **Condition coverage** – reports whether each boolean sub-expression (separated by logical-and or logical-or) evaluates to both true and false
- **Path coverage** – reports whether each of the possible paths in each function has been tested
 - Path = unique sequence of branches from the function entry to the exit point

Code coverage

30

- **Code coverage** - how much of your code is exercised by your tests
- **Code coverage analysis** = the process of:
 - Finding sections of code not exercised by test cases
 - Creating additional test cases to increase coverage
 - Computing a measure of coverage (which is a measure of test suite quality)
- A **code coverage analyzer** automates this process

Agenda for today

32

- Introduction
- What is software testing?
- Why do we need software testing?
- Testing terminology
- Black box vs. white box testing
- Testing strategy ←
- Test automation
- Contracts and tests
- TestStudio

Testing strategy

33

How do we plan and structure the testing of a large program?

- **Who is testing?**
 - Developers / special testing teams / customer
 - It is hard to test your own code
- **What test levels do we need?**
 - Unit, integration, system, acceptance, regression test
- **How do we do it in practice?**
 - Manual testing
 - Testing tools
 - Automatic testing

Testing and bug prevention


35

"Three questions about each bug you find" (Van Vleck):

- **"Is this mistake somewhere else also?"**
- **"What next bug is hidden behind this one?"**
- **"What should I do to prevent bugs like this?"**

SOFTWARE ENGINEERING COMIX ..

34



Tom Van Vleck,
ACM SIGSOFT
Software
Engineering Notes,
14/5, July 1989

Chair of Software Engineering

Test-driven development (TDD)

36

- Software development methodology
- One of the core practices of extreme programming (XP)
 - **Write test, write code, refactor**
- More explicitly:
 1. Write a small test.
 2. Write enough code to make the test succeed.
 3. Clean up the code.
 4. Repeat.
- The testing in TDD is **unit testing + acceptance testing**
- Always used together with xunit

xunit

37

- The generic name for any test automation framework for unit testing
 - **Test automation framework** – provides all the mechanisms needed to run tests so that only the test-specific logic needs to be provided by the test writer
- Implemented in all the major programming languages:
 - JUnit – for Java
 - cppunit – for C++
 - SUnit – for Smalltalk (the first one)
 - PyUnit – for Python
 - vbUnit – for Visual Basic

Why automate the testing process?


39

Facts from a survey of 240 software companies in North America and Europe:

- 8% of companies release software to beta sites **without any testing**.
- 83% of organizations' software developers **don't like** to test code.
- 53% of organizations' software developers don't like to test their own code because they find it **tedious**.
- 30% don't like to test because they find testing **tools inadequate**.

Agenda for today

38

- Introduction
- What is software testing?
- Why do we need software testing?
- Testing terminology
- Black box vs. white box testing
- Testing strategy
- Test automation 
- Contracts and tests
- TestStudio

What can you automate? (1)

40

- Test **generation**
 - Generation of test data (objects used as targets or parameters for feature calls)
 - Procedure for selecting the objects used at runtime
 - Generation of test code (code for calling the features under test)
- Test **execution**
 - Running the generated test code
 - Method for recovering from failures



What can you automate? (2)

41

- Test **result evaluation**
 - Classifying tests as pass/no pass
 - Other info provided about the test results
- Estimation of **test suite quality**
 - Report a measure of code coverage
 - Other measures of test quality
 - Feed this estimation back to the test generator
- Test **management**
 - Let the user adapt the testing process to his/her needs and preferences
 - Save tests for regression testing



Agenda for today

43

- Introduction
- What is software testing?
- Why do we need software testing?
- Testing terminology
- Black box vs. white box testing
- Testing strategy
- Test automation
- Contracts and tests ←
- TestStudio



Where is the difficulty?

42

How do you automatically evaluate test results as **pass or fail**?



You need to know the **specification** of the SUT



Contracts provide this specification



Design by Contract™ and testing

44

"Design by Contract implemented with assertions is a straightforward and effective approach to built-in-test. Not only does this strategy make testing more efficient, but it is also a powerful bug prevention technique."

Robert V. Binder, *Testing Object-Oriented Systems: Models, Patterns, and Tools* (1999)

Assertions as built-in test (BIT)

45

- Must be executable
- An executable assertion has 3 parts:
 - A **predicate expression**
 - In Eiffel: boolean expression + **old** notation
 - An **action**
 - Executed when an assertion violation occurs
 - An **enable/disable mechanism**

Quality of the contracts and the test suite (1)

47

- Use **mutation testing** to determine the quality of the specification (assertions) and the test suite
- Faulty versions of the program = **mutants**
 - A test set is relatively adequate if it distinguishes the original program from all its non-equivalent mutants
 - A **mutation score** (MS) is associated to the test set to measure its effectiveness
 - A mutant is said to be **killed** if at least one test case detects the fault injected into the mutant
 - A mutant is said to be **alive** if no test case detects the injected fault

Benefits and limitations of assertions as BIT

46

- Advantages:
 - BIT can evaluate the internal state of an object without breaking encapsulation
 - Contracts written before or together with implementation
- Limitations inherent to assertions
 - Frame problem
- **The quality of the test is only as good as the quality of the assertions**

Quality of the contracts and the test suite (2)

48

- **System test quality** (STQ)
 - S - system composed of n components denoted C_i , $i \in [1..n]$
 - d_i - number of killed mutants after applying the unit test sequence to C_i
 - m_i - total number of mutants
 - the mutation score MS for C_i being given a unit test sequence T_i :
$$MS(C_i, T_i) = d_i / m_i$$
 - $STQ(S) = \frac{\sum_{i=1..n} d_i}{\sum_{i=1..n} m_i}$
 - STQ is a combined measure of test suite quality and contract quality

References (1)

53

- **About testing:**
 - OO testing "bible":
Robert V. Binder: *Testing Object-Oriented Systems: Models, Patterns, and Tools*, 1999.
 - Writing unit tests with JUnit:
Erich Gamma and Kent Beck: *Test Infected: Programmers Love Writing Tests*
<http://junit.sourceforge.net/doc/testinfected/testing.htm>
 - Code coverage:
<http://www.bullseye.com/coverage.html>
 - Mutation testing:
Jezequel, J. M., Deveaux, D. and Le Traon, Y.
Reliable Objects: a Lightweight Approach Applied to Java. In *IEEE Software*, 18, 4, (July/August 2001) 76-83



End of lecture 17

References (2)

54

- **Testing tools:**
 - JUnit: <http://www.junit.org/index.htm>
 - Gobo Eiffel Test:
<http://www.gobosoft.com/eiffel/gobo/getest/>
 - TestStudio:
http://se.inf.ethz.ch/people/leitner/test_studio/