Software Testing I

Prof. Dr. Holger Schlingloff
Humboldt-Universität zu Berlin
and
Fraunhofer Institute of Computer Architecture
and Software Technology FIRST
Outline of this Lecture Series

• 2006/11/24: Introduction, Definitions, Examples
• 2006/11/25-1: Functional testing
• 2006/11/25-2: Structural testing
• 2006/11/26-1: Model-based test generation
• 2006/11/26-2: Specification-based test generation

• Next week: Your turn!
Outline of Today’s Lecture

- Introduction
  - Motivation, definitions, …
  - Topics, classification, tools
- Quick Tour through Jorgensen-Examples
  - The triangle problem
  - The NextDate function
  - The commission problem
  - The automated teller machine
  - The currency converter
  - The windshield-wiper controller
Literature

  (3rd ed. announced for 2007)
• A.P. Mathur: Foundations of Software Testing. Purdue University, 775 pp., to appear 2007
• G.J. Myers: The Art of Software Testing, Wiley 1979
• R.V. Binder: Testing Object-oriented Systems - Models, Patterns and Tools, Addison-Wesley 1999
Why Testing?

• Perpetual “software crisis”
• Ever-increasing complexity, ubiquity
• Continuous stories about bad software
• Customer dissatisfaction, damage
• Millions and millions of lost revenues

➡ Proficient testers are well-engaged and well-paid people
What is „Software Testing“?

• Cf. the announcement: „Testing is the process of systematically experimenting with an object in order to establish its quality.“
  - **Experiment**: singular activity to find something out
  - **Probe**: experiment to find out the quality
  - **Test**: *systematic* set of probes

• systematic = in the way in which the object (system) is composed
  - needs planning
  - needs analysis of the object
  - needs measurement
What is „Software“?

...systematically experimenting with an object ...

- Object: as contrasted to the subject conducting the test („SUT“ = “system under test” or “software under test”, „IUT“ = “implementation under test”)

- Software: precise description of information processing activities to be executed by a machine
  - non-ambiguous, finite, executable, in some programming language, ...

- Information processing activity
  - static view: components, interactions, data formats, ...
  - dynamic view: actions, transitions,
What is „Software Quality“?

• „degree of accordance to the intention or specification“
  - no absolute notion of quality
  - many possible quality measures
    - functionality, usefulness
    - efficiency (time, space, money)
    - safety, reliability, robustness, fault-tolerance
    - usability, maintainability, …
    - most important: correctness, i.e. absence of errors
  - intention or specification must be written down
Specifications

• Testing is impossible without specification

• Often, specifications are implicit or imprecise
  ▪ “the SUT shall never crash”
  ▪ “no error messages”, no “doesn’t respond”
  ▪ “all buttons can be pressed”, “all methods can be called”, “all functions can be accessed”
  ▪ “as fast as possible”, “security must be maintained”, “with feasible cost/benefit ratio”

• Make sure you get the specifications right!
Testing, Validation, Verification

Trying to answer different questions

• Testing: Did we build the software right?
• Validation: Did we build the right software?
• Verification: Can we show that the software is correct?

• Dijkstra: “Testing can only show the presence of bugs, not their absence.”
• Hoare (attributed): “Beware of this program. I haven’t tried it yet, I only proved its correctness.”

Consider two airplanes: One brand new, with verified but untested software. The other one with software which is thoroughly tested but not verified. Which one would you enter?
Errors, Faults, Failures

- **Error** or **Mistake** – something a person thought or did he/she shouldn’t have (bad idea or action)
- **Fault** or **Defect** – something wrong within the design or machine (bad state or flaw), due to an error during the design or manufacturing process
- **Failure** – wrong behaviour, malfunction of an artefact due to the activation of a fault
- **Incident** or **Accident** – visible effect of a failure onto the environment of the system, esp. on people
Tests, Test Cases and Test Suites

- Test – the execution of a test case
- Test Case – an entity identifying preconditions, inputs and expected outputs or postconditions for a particular SUT behaviour
- Test Suite – set of test cases for a particular testing objective (quality measure), usually with common points of observation and control (PCOs) in the SUT
- Test Design – the construction of test suites
Implementation and Testing

• “Programmer” and “Tester” are fundamentally different roles
  ▪ programmer wants to show correctness of his creation
  ▪ tester has the task to find errors, i.e. testing is successful if it uncovers deficiencies

• “Programmer” and “Tester” are essentially similar roles
  ▪ programmer creates executable artefacts (programs) from specifications
  ▪ tester creates executable artefacts (test suites) from specifications
Implementation and Testing

- V-Model: constructive and analytical part of software life cycle
Topics in Testing

- What is the specification, what is the SUT?
- Which interfaces to the SUT are needed? (Harness)
- What is the testing objective? (Purpose, Conditions)
- How are test cases derived? (Test case generation)
- How can the verdict be assigned? (Test oracle)
- How to write down test cases? (Testing languages)
- When is a test suite sufficient for the objective? (Test strategy)
- How are test cases executed? (Testing environment)
- When to stop testing? (Test coverage)
- How to reuse test results for subsequent activities? (Regression testing)
Classification of Testing (1)

- According to system life-cycle or structure
  - analysis, design, implementation, integration, deployment
  - module/unit, component, system, user

- According to class of SUT
  - operating system, middleware, driver, library, application, GUI, web-service, embedded software, ...

- According to testing method / test selection method
  - static or dynamic, structural or functional, control- or data oriented, single or regression test, ...
Classification of Testing (2)

• According to testing purpose or objective
  - functional testing, acceptance test, usability test, load test, interoperability test, safety test, …

• According to available information and specification method
  - Black-Box, White-Box, Grey-Box
  - UML-/model based, contract/requirement based, style guide based, formal specifications, …

• According to tool use and degree of automation:
  - manual or automatic (scripted) test execution, test case generation, test evaluation, management and documentation
Levels of Testing

- **User level**: requirements test, rapid prototyping, usability test, installation and configuration test, load and stress test
- **System level**: system test, design test, module interaction test, acceptance test, back-to-back-test, GUI testing, performance and robustness test
- **Module level**: module test, integration test, communication test, data flow test, data integrity test, cause-effect test
- **Unit level**: unit test, logic test, equivalence class test, boundary value test, control flow test, loop test
Testing Tools

• Like any other software engineering activity, the effectiveness and efficiency of testing highly depends on the tools deployed

• www.testingfaqs.org lists more than 500 (!) tools in the following categories
  - Unit test tools
  - Test drivers, test suite management
  - Test implementation, static analysis
  - Test design tools, test coverage monitors
  - Load and performance testing
  - GUI test drivers
  - Defect tracking systems, bundled suites
Testing languages

Several languages are being used for writing down test cases

- .doc (esp. for manual test execution)
- MS Excel or .txt, .csv etc. (tabular notation)
- csh, .bat (command-line SUT)
- Perl, Python, AWK, Tcl, …
- C, C++
- language of the SUT
- TSL, TestML
- TTCN-3
- …
Short Break!
Outline of Today’s Lecture

- **Introduction**
  - Motivation, definitions, …
  - Topics, classification, tools

- **Quick Tour through Jorgensen-Examples**
  - The triangle problem
  - The NextDate function
  - The commission problem
  - The automated teller machine
  - The currency converter
  - The windshield-wiper controller
The Triangle Problem

• Function `triangle` takes three integers a, b, c which are length of triangle sides; calculates whether the triangle is equilateral, isosceles, or scalene.

• The task is to write down test cases for this function

• “Classical” testing task (Myers)

• Do it NOW!
Evaluation

Each “yes” gives you one point

• Do you have a test case for an equilateral triangle?
• Do you have a test case for an isosceles triangle? (must be a triangle, not, e.g. (2,2,4))
• Do you have a test case for an admissible scalene triangle (must be a real triangle, not, e.g. (1,2,3))
• Do you have at least three test cases for isosceles triangles, where all permutations of sides are considered? (e.g. (3,3,4), (3,4,3), (4,3,3))
• Did you state for each test case the expected result?
Evaluation (2)

- Do you have a test case with one side zero?
- Do you have a test case with negative values?
- Do you have a test case where the sum of two sides equals the third one? (e.g. (1,2,3))
- Do you have at least three test cases for such non-triangles, where all permutations of sides are considered? (e.g. (1,2,3), (1,3,2), (3,1,2))
- Do you have a test case where the sum of the two smaller inputs is greater than the third one?
- Do you have at least three such test cases?
Evaluation (3)

- Do you have the test case (0,0,0)?
- Do you have test cases with very large integers (maxint)?
- Do you have a test case with non-integer values? (e.g., real numbers, hex values, strings,...)
- Do you have a test case where 2 or 4 inputs are provided?

Average programmer’s score 7-8 points

Myers 1979: this example should demonstrate that testing even a trivial program is not an easy task. Consider the problem of testing an air traffic guidance system with 100,000 instructions, a compiler or just a payroll program.

Today’s programs have 1-30 MLoC
Improved Triangle Problem

• The program accepts three integers between 1 and 200 which satisfy the triangle inequalities. The output is the type of triangle determined by the three sides.
• If the input does not match the range requirements, the program issues an error message and aborts.
• If the input does not satisfy the triangle inequalities, the program output is “NotATriangle”
• Otherwise, the output is
  ▪ “Equilateral”, if all three inputs are equal
  ▪ “Isosceles”, if exactly one pair of inputs is equal
  ▪ “Scalene”, if all inputs are pairwise unequal
Second Example: NextDate

• A “date” consists of three integers: month, date, year

• NextDate takes a date and returns the date of the following day
  - Checks whether input date is valid (according to Gregorian calendar, no Feb 31st etc.)
  - Return value respects leap years etc.

• Two sources of complexity
  - input domain check
  - rules of leap years
DaysInMonth

- \( 30 + ((m \text{ mod } 2) \text{ xor (m div 8)}) - n*(n=2) \)

- if \( m=2 \) then 28
  else if \( m<7 \) and even\((m)\) or \( m>7 \) and odd\((m)\)
  then 30 else 31

- if \( m=2 \) then 28
  else if \( m \in \{4, 6, 9, 11\} \) then 30 else 31

- array
  \( \text{DiM=[31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]} \)
  return \( \text{DiM[month]} \)
Rules for Leap Years

• One year is approximately 365.2422 days

• Julius Caesar: every fourth year is a leap year
• Pope Gregory in 1582 reformed Caesarian rules
• Year is leap year iff divisible by 4 but not by 100, or if divisible by 400. Thus 1600, 2000, 2004 and 2008 are leap years, but 1700, 1900 and 2100 are not.

• Leap seconds due to slow-down of earth rotation…

Homework: implement and test the NextDate function!
Third Example: Commission Problem

- Rifle sales company produces
  - locks $45
  - stocks $30
  - barrels $25

- Salesmen send sales reports via telegraph; commission is 10% on sales up to $1000, 15% on the next $800, 20% on everything above

- Program produces monthly sales reports and commission to be paid

- typical commercial application, mix of computation and decision, input stream, output table(?), functional behaviour
Fourth example: SATM

• “Simplified Automated Teller Machine”
• One of those common money distributors
  ▪ Screen display, numerical keypad, card reader, cash dispensing unit, (receipt printer)
• Specification mostly given by screenshots (“rapid prototyping”)
  ▪ implicit information, e.g. which bills are available
• State-transition system
  ▪ not much calculation
  ▪ user interaction
  ▪ Client-server paradigm
Fifth example: Currency converter

- Example of GUI (graphical user interface) program
  - text fields, radio buttons, ...
- Web-based, web-service
Last example: Windshield wiper controller

- Embedded control system
  - Reactive system, continuous interaction
  - Real-time properties
- Input: lever and dial setting, bus signals
  Output: motor signals / voltages
- Problems: Interfacing, HiL-Testing

<table>
<thead>
<tr>
<th>Lever</th>
<th>OFF</th>
<th>INT</th>
<th>INT</th>
<th>INT</th>
<th>LOW</th>
<th>HIGH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dial</td>
<td>n/a</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Wiper</td>
<td>0</td>
<td>4</td>
<td>6</td>
<td>12</td>
<td>30</td>
<td>60</td>
</tr>
</tbody>
</table>
That’s it For Today!