The Non-Virtual Reality of Testing or What's Feasible in Real World Testing

Contents
1. Introduction
2. Seven myths about testing and their demystification
3. A kind of conclusion

Karol Frühauf, INFOGEM AG, CH-5400 Baden, Karol.Fruehauf@infogem.ch
Seven myths about testing

I. Testing is a hobby of quality people
II. The quickest way to deployment is ping-pong testing
III. Test automation is cheap
IV. You don't need to see what you test
V. Integration testing is interface testing
VI. Test coverage is a glass box test concept
VII. Test planning is an easy task
I Testing is a hobby of quality people (1)

- project goals
  - dates
  - cost
  - requirements
  - actual versus planned
  - review or test
    - dates
    - cost
    - work results
  - project state
  - without review and test no real progress control
I Testing is a hobby of quality people (2)

- project management
- product development
- product testing

⇒ don't throw defects over the wall to the developer
The quickest way to release is ping-pong testing

.. as soon as the tester detects a defect

he returns the software to the developer

⇒ we have one defect to fix ...
⇒ expensive regression tests
⇒ if special condition then rucksack;
⇒ execute all specified test cases, then switch to repair mode
III  Test automation is cheap (1)

system level
- capture / replay tools
  + capture is cheap
  - replay is expensive
- test case managers
  requires strong update discipline

unit level
+ JUnit etc.
+ test coverage profiler
+ etc.
III Test automation is cheap (2)

use home made dedicated tools ... keep them up-to-date
IV You don't need to see what you test (1)
IV You don't need to see what you test (2)
Integration testing is interface testing (1)

Integration testing: Testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them.
[IEEE 610.12]

Integration testing: Testing performed to expose faults in the interfaces and in the interaction between integrated components.
Interface testing: Integration testing where the interfaces between system components are tested
[BS7925-1]

Integration testing is the process of verifying the interaction between system components (possibly and hopefully tested already in isolation).
[SWEBOK 1.0]
V Integration testing is interface testing (2)

implementation testing
→ testing in which aggregates are tested with the aim to detect defects caused by errors made during implementation
→ concern is the functionality of the aggregate (unit testing) or the interaction of its parts (interface testing)

integration testing
→ testing in which aggregates are tested with the aim to detect defects caused by errors made during integration, e.g.
  o building
  o writing scripts (function test of scripts)
  o integration of components to tiers and these to system
  o integration of components to subsystems and these to system
  o configuration of the system
  o installation of the system in the target environment
V Integration testing is interface testing (3)

type of errors integration testing is looking for
- wrong address
- wrong name used
- queue is not set-up
- queue is too small
- file is missing or is in wrong location
- processes are started in a wrong sequence
- a process is not started at all
- wrong setting of configuration parameters or no setting at all
- etc.
VI  Test coverage is a glass box test concept (1)

a quite usual conversation ...
A quite usual conversation ...

A: How do you test your programs?
E: In the usual way, like anybody else.
A: I mean, how do you select the test cases that you intend to execute in order to torture your program?
E: Simple, that’s easy.
A: Good. Which method do you apply?
E: Method? I know what I need to test.
A: Of course you know it. I am interested to learn, when do you stop test case selection and specification?
E: When I have enough test cases.
A: Exactly, that is what I want to know, when do you have enough?
E: As soon as I don’t need any more.
A: Yes, of course, but how do you decide that you don’t need any more, that your set of test cases is complete?
E: Man, everybody knows that there is nothing like complete testing.
A: I am convinced there is.
E: Even if it were it’s too expensive, nobody can afford it ... and it doesn't work anyway.
A: Would you agree, then, that you test intuitively?
E: Yes, I do, and I am proud of it.
Example: Black-box test of the Windows clock
## Example: A complete set of test cases (1)

<table>
<thead>
<tr>
<th>output</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>analogue time display</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>digital time display</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>font (28 types)</td>
<td>Arial</td>
<td></td>
<td></td>
</tr>
<tr>
<td>display of the Greenwich time</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>display of the system time</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>display of the title bar</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>no display of the title bar</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>display of seconds</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>no display of seconds</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>display of the date</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>no display of the date</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>display of information</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
**Example: A complete set of test cases (2)**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>time display</td>
<td>gch</td>
<td>gch</td>
<td>gch</td>
<td>gch</td>
<td>sys</td>
<td>sys</td>
<td>sys</td>
<td>sys</td>
</tr>
<tr>
<td>title bar display</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>seconds display</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>date display</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

**analogue display of time:** 8 test cases

**digital display of time:** 448 test cases

**date display is possible:** doubles the analogue test cases = 16

**28 font types available:** $16 \times 28 = 448$

**total:** analogue display + digital display + info = $8 + 448 + 1 = 457$ test cases
VI  Test coverage is a glass box test concept (2)

first criterion (3 test cases)
\[\Rightarrow \text{ for all possible types of display at least one of the possible outputs is produced by at least one test case}\]

second criterion (457 test cases)
\[\Rightarrow \text{ all possible combinations of outputs are produced by at least one test case}\]

a possible criterion in between (30 test cases)
\[\Rightarrow \text{ all possible outputs are produced by at least one test case}\]
VI Test coverage is a glass box test concept (3)

- testing is a sampling procedure
  - the sample content depends on risks
    - the sample size is defined by the envisaged "confidence level" of the test

⇒ coverage defines the sample
⇒ coverage is a target for the test designer
⇒ coverage makes systematic test case selection possible
⇒ coverage determines the extent, thus also the cost of testing
⇒ coverage enables the project leader / software manager to (better) assess the state of affairs
VII Test planning is an easy task

– we do unit testing, integration testing, system testing

Test planning involves
  o identify system boundaries and system structure
  o define strategy for reviewing, integration, and testing
  o analyse risks
  o define test objects
  o for all test objects define the characteristics of the test
  o design the test infrastructure and specify the test harness
  o identify all testing activities and estimate the effort
  o trade cost and benefit of the tests
  o schedule test activities and assign resources

Can't be all done at the beginning and not all of what can be done, can be defined with the same level of detail
## Characteristics of a single test

<table>
<thead>
<tr>
<th><strong>test object</strong></th>
<th>one executable unit (or many)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>test level</strong></td>
<td>unit or component or system or an aggregate in between</td>
</tr>
<tr>
<td><strong>test environment</strong></td>
<td>development or integration or test or production</td>
</tr>
<tr>
<td><strong>error types to look for</strong></td>
<td>logic, data entry, navigation, fault tolerance, connection, communication, response time, size, etc.</td>
</tr>
<tr>
<td><strong>basis for test case specification</strong></td>
<td>artefact used to gather information about possible test inputs and expected output</td>
</tr>
<tr>
<td><strong>basis for test case selection</strong></td>
<td>artefact used to define test coverage criteria used to assess the completeness of the selected test case set</td>
</tr>
<tr>
<td><strong>test dimensions</strong></td>
<td>configuration parameters of the run-time environment</td>
</tr>
<tr>
<td><strong>test goal</strong></td>
<td>extent of error type and test dimensions coverage</td>
</tr>
<tr>
<td><strong>test execution</strong></td>
<td>manually using a checklist, using test procedures, with automatic test logging, completely automated, etc.</td>
</tr>
<tr>
<td><strong>tester</strong></td>
<td>user, test engineer, ignorant, expert, etc.</td>
</tr>
<tr>
<td><strong>test evaluation</strong></td>
<td>compare with specification (basis), compare with assured results, etc.</td>
</tr>
<tr>
<td><strong>test record</strong></td>
<td>completed checklist, manual, test log, automatic test log, etc.</td>
</tr>
</tbody>
</table>
Example: System test planning with variations

<table>
<thead>
<tr>
<th>WEB</th>
<th>dimension</th>
<th>possible values</th>
<th>cardinality</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td></td>
<td>NT 2000 XP</td>
<td>3</td>
</tr>
<tr>
<td>browser</td>
<td>Netscape</td>
<td>IE Firefox</td>
<td>3</td>
</tr>
<tr>
<td>registered user</td>
<td>no</td>
<td>yes</td>
<td>2</td>
</tr>
<tr>
<td>locked user</td>
<td>no</td>
<td>yes</td>
<td>2</td>
</tr>
<tr>
<td>user language</td>
<td>German</td>
<td>French Italian English</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WAP</th>
<th>dimension</th>
<th>possible values</th>
<th>cardinality</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator</td>
<td>we</td>
<td>foreign</td>
<td>2</td>
</tr>
<tr>
<td>device brands</td>
<td>5 new</td>
<td>15 legacy</td>
<td>20</td>
</tr>
<tr>
<td>registered user</td>
<td>no</td>
<td>yes</td>
<td>2</td>
</tr>
<tr>
<td>locked user</td>
<td>no</td>
<td>yes</td>
<td>2</td>
</tr>
<tr>
<td>user language</td>
<td>German</td>
<td>French Italian English</td>
<td>4</td>
</tr>
</tbody>
</table>

minimal number of variations

WEB: 4
WAP: 20

theoretically maximal number of variations

WEB: 144
WAP: 640
tesztelést fejtegettem
kevertem tekertem
remélem
elég értelmesen értékeltem
eme nemes cselekedést

esetleg nem érzékeltem
kedves teremben elhelyezettek
értékes nézetét
e tett szenvedésért
elnézést kérek

kellemes rendhelytelenség
kegyetlen keresését
tisztelt tesztelők

The program did not crash,
Mister Byte!