T-76.5613 Software Testing and Quality Assurance

Lecture 2, 13.9.2006

Basic Testing Concepts and Terminology

Juha Itkonen
SoberIT
Contents

- Realities and principles of testing
- Testing terminology and basic concepts
- V-model, test levels and types
- The concept of software quality
Realities and principles of testing
The problem of complete testing

- Why not just test everything?
- Complete testing would require testing:
  - every possible valid and invalid input of the system
  - and every possible valid and invalid output
  - and every possible combination of inputs
    - in every possible sequence of inputs and outputs
  - and user interface errors
  - and configuration and compatibility issues
  - and environment related failures
  - ...
- There is definitely too much cases to test completely any realistic software module, not to mention a whole system.
Testing is always prioritizing

- Time is always limited
- Use company’s significant **risks** to focus testing effort
  - what to test first
  - what to test most
  - how thoroughly to test each feature
  - what not to test
- Most important tests first

- Possible ranking criteria
  1. test where a failure would be most severe
  2. test where failures would be most visible
  3. test where failures are most likely
  4. ask the customer to prioritize the requirements
  5. what is most critical to the customer’s business
  6. areas changed most often
  7. areas with most problems in the past
  8. most complex areas, or technically critical
Realities of testing (1/2)

- It is impossible to test a program completely
- Testing can show the presence of defects but cannot show the absence of defects
  - You can report found defects and what you have tested and how
- All bugs cannot be found
- All found bugs will not be fixed
  - It’s really not a bug
  - It’s too risky to fix, there is not enough time
  - There is more important tasks to do
- Testing does not create quality software or remove defects
- The critique is focused on the product, not the developer
Realities of testing (2/2)

- Product specifications are never final
  - You cannot wait until specifications are final
  - There will always be changes to specifications
  - The specifications are never complete

- The more bugs you find, the more bugs there are
  - Programmers have bad days
  - Programmers often make same mistake
  - Some bugs are really just the tip of the iceberg

- The Pesticide Paradox
  - Programs become immune to unchanging tests

- Software testing should be based on risks
Testing principles

- When the test objective is to detect defects, the a good test case is one that has a high probability of revealing a yet undetected defect(s).
- Test results should be inspected meticulously.
- A test case must contain the expected results.
- Test cases should be developed for both valid and invalid input conditions.
- The probability of the existence of additional defects in software component is proportional to the number of defects already detected in that component.
- Testing should be carried out by a group that is independent of the development team.
- Tests must be repeatable and reusable.
- Testing should be planned.
- Testing activities should be integrated into the software life cycle.
- Testing is a creative and challenging task.

Burnstein 2003
Testing terminology and basic concepts
Validation and verification (V&V)

- Validation – are we building the right product?
  - Implementation meets customer requirements, needs and expectations

- Verification – are we building the product right?
  - Program conforms to its specification

- **Note!** In the chapter 1 of the course book (Burnstein, 2003, p. 6) there is different definition for V&V. The above definition, however, is more common and makes more clear distinction between the terms.
Black-box and white-Box testing

- **Black-box (functional, behavioral, data-driven)**
  - The software under test considered as a black-box and no knowledge of the internal structure or how the software actually works is utilized in testing
  - Testing based on inputs and respective outputs
  - The size of the black box can vary from one class or component to a whole system

- **White-box (structural, logic-driven)**
  - Testing based on knowing the inner structure of the system and the program logic
Functional testing

- Testing conducted to evaluate the compliance of a system with specified functional requirements.
  
  *IEEE Standard Glossary of Software Engineering Terminology*

- Compare to non-functional testing
  - E.g. performance, reliability, usability, security testing
Dynamic and static quality practices

- Dynamic methods – testing – executes code
  - Software testing in the traditional sense
  - Dynamic analysis methods

- Static methods do not execute code
  - Reviews, inspections, static analysis
  - Some authors name these static testing
Scripted and non-scripted testing

- In scripted (test case based) testing test cases are pre-documented in detailed, step-by-step descriptions
  - Different levels of scripting possible
  - Scripts can be manual or automated
- Non-scripted (exploratory) testing is usually manual testing without detailed test case descriptions
  - Can be disciplined, planned, and well documented exploratory testing
  - ... or ad-hoc testing
What is a "bug"?

- **Error** - a human action that produces an incorrect result
- **Fault** - a manifestation of an error in software
  - also known as a defect, bug, issue, problem, anomaly, incident, variance, inconsistency, feature, ...
  - if executed, a fault may cause a failure
- **Failure** - deviation of the software from its expected delivery or service

Failure is an event; fault is a state of the software, caused by an error.
Error → Fault → Failure

A programmer makes an error...

...that creates a fault in the software...

100101
1110101
00100101
1110100

...that can cause a failure in operation
What is a fault?

- A software error is present when the program does not do what its end user reasonably expects it to do (Myers, 1979)
- There can never be an absolute definition for bugs, nor an absolute determination of their existence. The extent to which a program has bugs is measured by the extent to which it fails to be useful. This is a fundamentally human measure. (Beizer, 1984)
Rules for identifying a defect

- The software does not do something that the specification says it should do
- The software does something that the specification says it should not do
- The software does something that the specification does not mention
- The software does not do something that the specification does not mention but it should
- The software is difficult to understand, hard to use, slow, or will be viewed by the end user as just plain not right

Source: Software Testing, Ron Patton
Test Oracles – How do we know it’s broken?

- An oracle is the principle or mechanism by which you recognize a problem
- Test oracle provides the expected result for a test, for example
  - Specification document
  - Formula
  - Computer program
  - Person
- In many cases it is very hard to find an oracle
  - Even the customer and end user might not be able to tell which is the correct behaviour

Oracle problem is one of the fundamental issues in test automation: How do we teach an automated test to recognize a defect or failure when it happens? – This is not a trivial problem.
Oracle – Is this correct?

- Font size test Arial 22pt
  - Font size test Arial 20pt
    - Font size test Arial 18pt
      - Font size test Arial 16pt
      - Font size test Arial 14pt

- Font size test Times 22pt
  - Font size test Times 20pt
    - Font size test Times 18pt
      - Font size test Times 16pt
      - Font size test Times 14pt
Severity and priority

- **Severity** of a software fault refers to the severity of the consequences of a failure caused by that fault
  - A tester or customer is probably the best person to set the severity

- **Priority** is the fixing order of the found faults and a result of separate prioritisation activity
  - A tester is probably not the best person to set the priority
  - Prioritisation is typically a managerial decision
    - business priorities
    - customer priorities
    - other faults and features
    - quality policy
    - release strategy
  - Prioritisation is usually based on severity

The priority of a fault depends on other issues
Re-testing vs. regression testing

Re-tests
- New version of software with “fixed fault”
- Re-run the same test that revealed the defect
  - must be exactly repeatable
  - same environment, versions (except for the software which has been intentionally changed!)
  - same inputs and preconditions
- If test now passes, fault has been fixed correctly – or has it?

Regression tests
- Running a set of tests that have been run before
- After software or environment changes
  - For confidence that everything still works
  - To reveal any unanticipated side effects
- An asset (regression test suite/pack)
  - Standard set of tests - needs designing & maintaining
  - Well worth automating
Regression testing

Regression tests look for unexpected side-effects (but may not find all of them)

Fault fix introduces or uncovers new faults

Test finds fault

Re-test to check fix
V-model, test levels and types
V-model of testing

Requirements → Functional specification → Architecture design → Module design → Coding → Unit testing → Integration testing → System testing → Acceptance testing → Requirements

Build → Test
V-model is easy to understand

- Simple generic model
- Easy to understand and explain
- Describes the test levels and phases
- Shows how testing activities are related to other development activities
- Matches to familiar waterfall model
V-model says...

- It is good to have requirements laid out first.
- It is good to have plans and specifications done before implementation.
- It is good to complete one task before proceeding to the next that depends on the output of the first task.
- It is good to test each intermediate product.
- It is good to use each development specification as a basis for test process.
- It is much easier to find faults in small units than in large entities. Therefore, test small units first alone before putting them together.
- Testing of large units can be carried out more easily when the smaller parts are already tested.
- **Test level** – a test level is a group of test activities that focus into certain level of the test target
  - Test levels can be seen as levels of detail and abstraction
  - How big part of the system are we testing?
  - E.g. unit test, integration test, system test, acceptance test
Unit testing

- A unit = smallest testable software component
  - Objects and methods
  - Procedures / functions
  - Reusable components

- Focus on programming errors
  - Verifying that the code unit works as the specification says and the developer designed it
  - Most thorough look at the details

- Testing units in isolation
- Usually done by a programmer (in practice)
  - A pair tester can help
- Also known as component, module, or program testing
Integration Testing

- More than one (tested) unit work together
- Focus on interfaces
  - Communication between units
- Helps assembling incrementally a whole system
- Integration strategy: big-bang vs. incremental (top-down, bottom-up, functional, ...)
- Done by developers/designers or independent testers
  - Preferably developers and testers in collaboration
- Modern development methods emphasize continuous integration, no separate integration phase
- Integration testing can be done on different levels
  - Class level
  - Component level
  - Subsystem level
  - Systems integration level
System testing

- Testing the system as a whole
- Focus
  - Verifying that specifications are met
  - Validating that the system can be used for the intended purpose
    - By the development organization
- Functional as well as non-functional testing
  - Non-functional requirements often poorly specified
- Independent testing group
  - Collaborating developers and testers
Acceptance testing

- Final stage of validation
  - Customer (user) should perform or be closely involved
  - Customer should be able to perform any test they wish, usually based on their business processes
  - Final customer sign-off

- Approach
  - Should be performed in real operating environment
  - Environment related problems
  - Can the system be used for the intended purpose in the real operating environment
  - Problems that the real users have with the system

<table>
<thead>
<tr>
<th><strong>Project business</strong></th>
<th><strong>Product business</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Contract acceptance</td>
<td>Final checks on releases, (alpha, beta)</td>
</tr>
<tr>
<td>Customer viewpoint</td>
<td>User viewpoint throughout the development; validation</td>
</tr>
<tr>
<td>Validates that the right system was build</td>
<td></td>
</tr>
</tbody>
</table>
Levels, types, and phases – not a same thing

- **Test level** – a test level is a group of test activities that focus into certain level of the test target.
  - E.g. unit test, integration test, system test, acceptance test
  - Test levels can be seen as levels of detail and abstraction

- **Test type** – A test type is a group of test activities aimed at evaluating a system for a number of associated quality characteristics.
  - E.g. functional test, performance test, stress test, usability test

- **Test phase** – Test phases are used to describe temporal parts in testing process that follow sequentially each other, with or without overlapping
Common test types

Functional testing
- Function testing
- Installation testing
- Smoke testing
- Configuration testing
- Compatibility testing
- Exception testing
- Interface testing
- Conversion testing
- ...

Non-Functional testing
- Performance testing
- Load testing
- Stress testing
- Volume testing
- Reliability testing
- Security testing
- Usability testing
- Recoverability testing
- Maintainability testing
- Localization testing
- Documentation testing
- Data quality testing
- Platform testing
- Concurrency testing
- ...

See course book chapter 6.13 System Test: The Different Types
Software Quality
Which one describes high a quality software system?

- Defect free
- Popular in industry
- Fast to learn
- Nice graphics
- Easy to install and upgrade
- Robust
- Cheap
- Simple
- Small and compact
- Customizable
- Open-source
- Fit-for-purpose
- Maintainable
- Appeals majority
- Feature rich
- Award winning product
- Fast to use
- Low system requirements
- Reliable customer support
- Cool
- Expensive
- Big and complicated
- Large and scalable
- Standardized
- Reliable vendor
- Multipurpose
- Fast time-to-market
- Perfect for target segment
- ...
Definition of software quality

- **Quality**: (1) The degree to which a system, component or process meets specified requirements. (2) The degree to which a system, component, or process meets customer or user needs or expectations.


- Quality is value to some person(s)

Users’ viewpoint differs from yours

- All stakeholders have different viewpoints to quality
  - Customer
  - End user
  - Programmer
  - Project manager
  - Tester
- Whose opinion counts?
  - A bug free product can be unacceptable to the users
    - The goal of a software system is to solve customers problem
  - A minor annoyance for one user can make the product useless for another
What does product quality mean?

- Product
  - A general purpose text editor application
- What does quality mean for
  - Developer of the application (technical view)
  - Salesman (sales view)
  - Product manager (product view)
  - Tester (tester’s view)
  - Coder -user
  - Note taker -user
  - Report writer –user
  - Occasional -user
The McCall quality model
ISO 9126 Quality characteristics

Functionality

The capability to provide functions that meet stated or implied needs.

Portability

The capability to be transferred from one environment to another.

Reliability

The capability to maintain a specified level of performance, under stated conditions for a specified period of time.

Maintainability

The capability to be modified and make changes.

Usability

The capability to be understood, learned, used and attractive to user.

Efficiency

The capability to provide appropriate performance, relative to amount of resources used, under stated conditions.
Good Enough Quality

- Defined by James Bach
- To claim that any given thing is good enough is to agree with all of the following propositions:
  - It has sufficient benefits
  - It has no critical problems
  - The benefits sufficiently outweigh the problems
  - In the present situation, and all things considered, further improvement would be more harmful than helpful
Software quality

- In general, it is wise to strive for good quality and try to avoid making too many mistakes during the development project
  - Bad quality leads to rework, bigger costs, delays, and problems in operation
- Quality
  - is meeting the specifications
  - is meeting the expectations of the customer or user
  - is value to someone
    - differs depending on who you ask
  - is difficult to define and hard to measure
- General standards for software quality
  - E.g. ISO 9126, McCall quality model, ...
  - Gives a nice taxonomy, terminology, definitions and conceptual framework
  - Cannot tell what are the most important quality attributes and how to achieve them in a particular context and situation