Basic Principles of Software Quality Assurance

Adapted from S. Somé, A. Williams

Dilbert on Validation

Where does the customer fit in V&V?

Three General Principles of QA

- Know what you are doing
  - Understand what is being built, how it is being built and what it currently does.
  - Follow a software development process with
    - Management structure (milestones, scheduling)
    - Reporting policies
    - Tracking

Three General Principles of QA

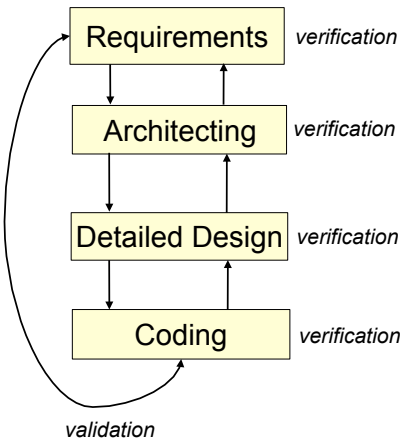
- Know what you should be doing:
  - Having explicit requirements and specifications.
  - Follow a software development process with
    - Requirements analysis,
    - Acceptance tests,
    - Frequent user feedback.

Three General Principles of QA

- Know how to measure the difference.
  - Have explicit measures comparing what is being done from what should be done.
  - Four complementary methods:
    - Formal methods – verify mathematically specified properties.
    - Testing – explicit input to exercise software and check for expected output.
    - Inspections – human examination of requirements, design, code, ... based on checklists.
    - Metrics – measures a known set of properties related to quality

Verification and Validation

- Verification:
  - Are we building the product right?
  - Performed at the end of a phase to ensure that requirements established during previous phase have been met.
- Validation:
  - Are we building the right product?
  - Performed at the end of the development process to ensure compliance with product requirements.

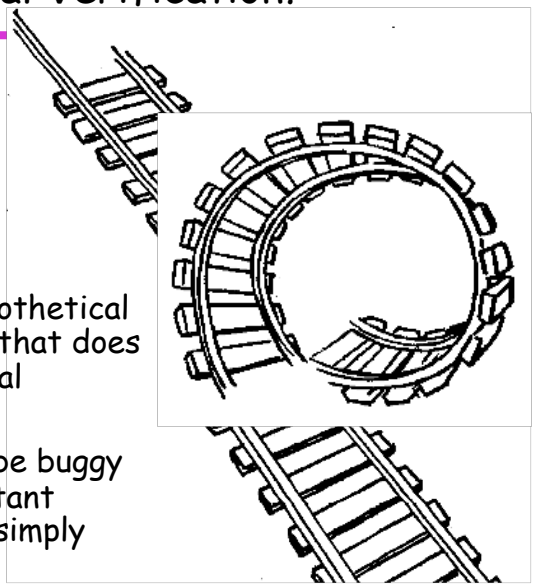


Do these 2 definitions make sense in the context of an Agile process with short iterations??

V&V Approaches

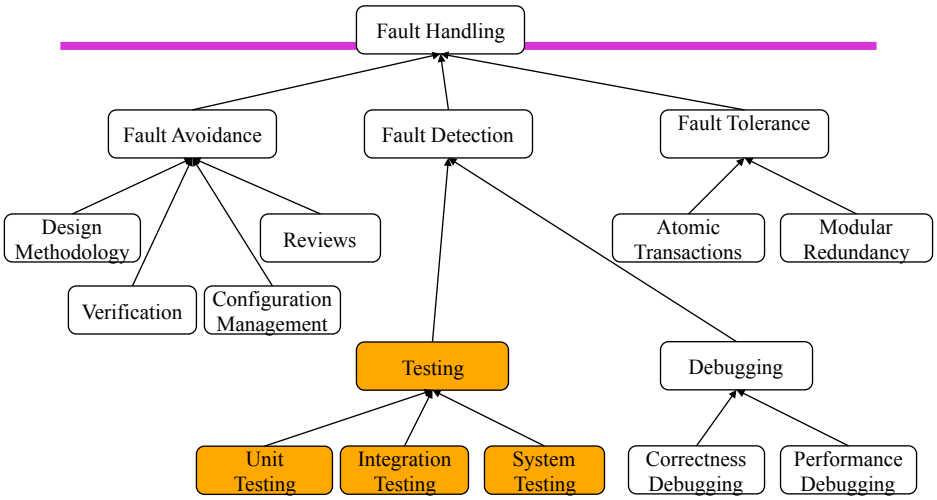
- Walkthroughs and inspections
  - Not formal, yet systematic and cost-effective
- Correctness proofs
  - Theorem proving
  - Complex, undecidable, not for all programs…
- Symbolic execution
  - Somewhere in between testing and theorem proving
- Model checking
  - State-based model + logical formula (property)
  - Decidable if state space is finite (and small enough)
- Testing
  - Incomplete, but practical. Can this be model-driven?

Formal Verification?



- Risks
  - Assumes hypothetical environment that does not match real environment
  - Proof might be buggy (omits important constraints; simply wrong)

Fault Handling Techniques



What is Software testing ?

- Examination of a software unit, several integrated software units or an entire software package by running it.
  - execution based on test cases
  - expectation – reveal faults as failures
- Failure: incorrect execution of the system
  - usually consequence of a fault / defect
- Fault/defect/bug result of a human error

Objectives of testing

- To find defects before they cause a production system to fail.
- To bring the tested software, after correction of the identified defects and retesting, to an acceptable level of quality.
- To perform the required tests efficiently and effectively, within the limits of budgetary and scheduling constraints.
- To compile a record of software errors for use in error prevention (by corrective and preventive actions), and for process tracking.

When does a fault occur ?

- Software doesn't do something that the specification says it should do.
- Software does something that the specification says it shouldn't do.
- Software does something that the specification doesn't mention.
- Software doesn't do something that the specification doesn't mention but should.
- Software is difficult to understand, hard to use, slow, or – in the software tester's eyes – will be viewed by the end user as just plain not right.

Testing Axioms

- It is impossible to test a program completely.
  - large input space
  - large output space
  - large state space
  - large number of possible execution paths
  - subjectivity of specifications

Testing Axioms

- Large Input/State space

```
int exFunction(int x, int y)
{
   ...
}
```

- Exhaustive testing ⇒ trying all possible combinations of `x` and `y`
  - Range for just one `int` in Java: `Integer.MIN_VALUE` to `Integer.MAX_VALUE`; that is: 4,294,967,295 different possibilities
    - The range is not infinite – but it is very, very large.
  - Combinations of 2 `int` variables: square of the above.

Testing Axioms

- Large number of possible execution paths:

```
for (int i = 0; i < n; i++)
{
   if (a.get(i) == b.get(i))
   {
      // do something
   }
   else
   {
      // do something else
   }
}
```

- Number of potential execution paths is 2^n
  - If n = 40, the number of paths could be 1,099,511,627,776
- Binder has a formula to compute the upper limit on the # of tests…

Testing Axioms

- Software testing is a risk-based exercise
  - Need to balance cost and risk of missing bugs
- Testing cannot prove the absence of bugs
- The more bugs you find, the more bugs there are
- The "pesticide paradox"
  - A system tends to build resistance to a particular testing technique.

Testing Axioms

- Not all bugs found are fixed.
- Difficult to say when a bug is a bug.
  - only when observed (latent otherwise)
- Product specifications are never final.
- Software testers sometimes aren't the most popular members of a project team.
- Software testing is a disciplined technical profession that requires training.

What Testing can accomplish

- Reveal faults that would be too costly or impossible to find using other techniques
- Show the system complies with its stated requirements for a given test suite
- Testing is made easier and more effective by good software engineering practices

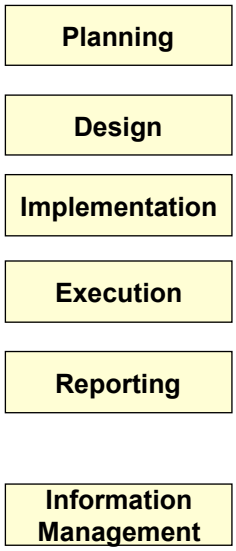
Risk

- Risk is the probability of an observed failure, multiplied by the potential cost (usually, estimated) of the failure.
  - Example:
    - Dropping one regular phone call: low probability, low potential cost (customer hits redial)
    - Dropping a 911 emergency call: low probability, high potential cost (lives)

Risk-based testing

- Choosing what to test is often influenced by risk.
- What are the probabilities that the following were implemented incorrectly…?
  - A "typical" scenario
  - Exceptional scenarios
- Empirically, the probability of a defect is much higher in the code for exceptional scenarios, and testing resources are often directed towards them.
  - Keep the cost factor of risk in mind as well.

Testing is a Process



- Planning
  - Scope, resources, schedules
  - Architecture, tools needed
- Design
  - Test selection
  - Coverage goals
- Implementation
  - Creation of (automated?) test scripts
- Execution
  - Running the tests
- Reporting
  - Problem reports
  - Progress tracking
  - Coverage measurement
- Information management (ongoing through process)
  - Change control of artifacts
  - Archive of test results

Terminology

- Warning: many organizations, tools, or standards use inconsistent terminology ☹.
- Stimulus: an action / request / command sent to a system under test.
- Response: something that can be observed from a system under test.
- Test case: a sequence of stimuli and expected responses with a specified purpose and decisive result.
  - Test script: a test case as a set of instructions, for either manual or automated execution.
- Test suite: a collection of related test cases.
- Verdict: the decision as to the result of a test case.

Testing basics

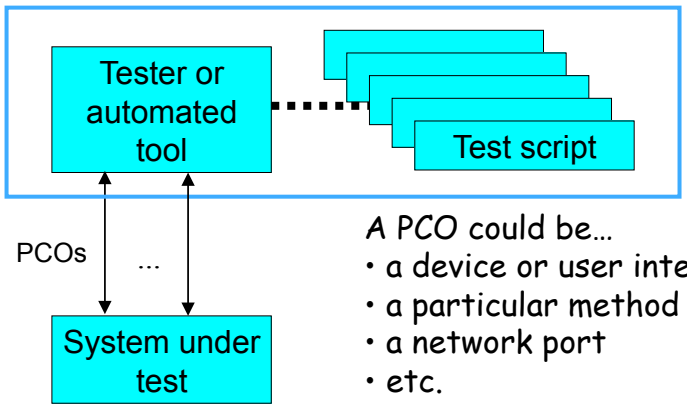
- What do we need to do testing?
  - Test script
    - Stimuli to send to system under test (SUT).
    - Responses expected from SUT.
    - When / how to determine the test verdict?
      - i.e. What are the decision points, and how is the decision made?
  - For an automated test execution system, additional requirements are:
    - Test controller, to manage execution.
    - Mechanism to read test script, and connect test case to SUT.

Verdicts

- Running a test results in a verdict at one or more decision points in a test case.
  - Pass:
    - The expected responses were observed in the proper sequence with the proper timing, and no other anomalies were observed.
  - Fail:
    - One or more of the expected responses were not observed.
  - Inconclusive / Error:
    - Anomalies were observed, or the purpose of the test could not be achieved.
      - Example: the test could not be set up properly.
  - Conf (used infrequently; from "conforms"):
    - An response was received that is correct in terms of content, but outside of the correct time window.

Test Architecture

- Includes defining the set of Points of Control and Observation (PCOs)



A PCO could be…
- a device or user interface
- a particular method to call
- a network port
- etc.

Do you observe directly the result of a function or via a change in the GUI? Distribution (including Web apps) makes this a lot more complicated!

Test Scripts

- What should the format of a test script be?
  - natural language? (for manual testing)
  - tool dependent?
  - a standard test language?
  - a programming language?
- Meszaros's work shows writing test scripts is a lot of work and must be done carefully (in order to avoid smells)

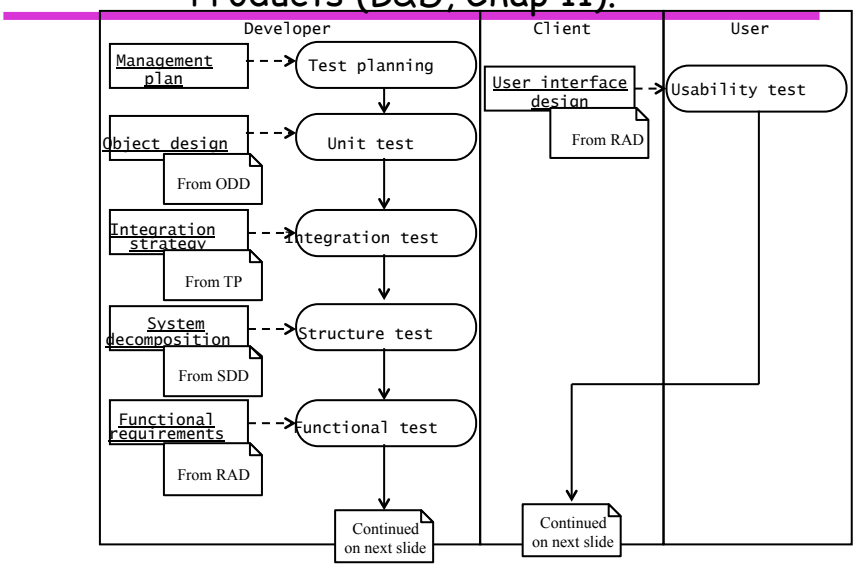
Test Script Development

- Creating test scripts follows a parallel development process, including:
  - Requirements
  - Design
  - Debugging
  - Configuration management
  - Maintenance
  - Documentation
- Result: they are expensive to create and maintain, especially for automated test execution.

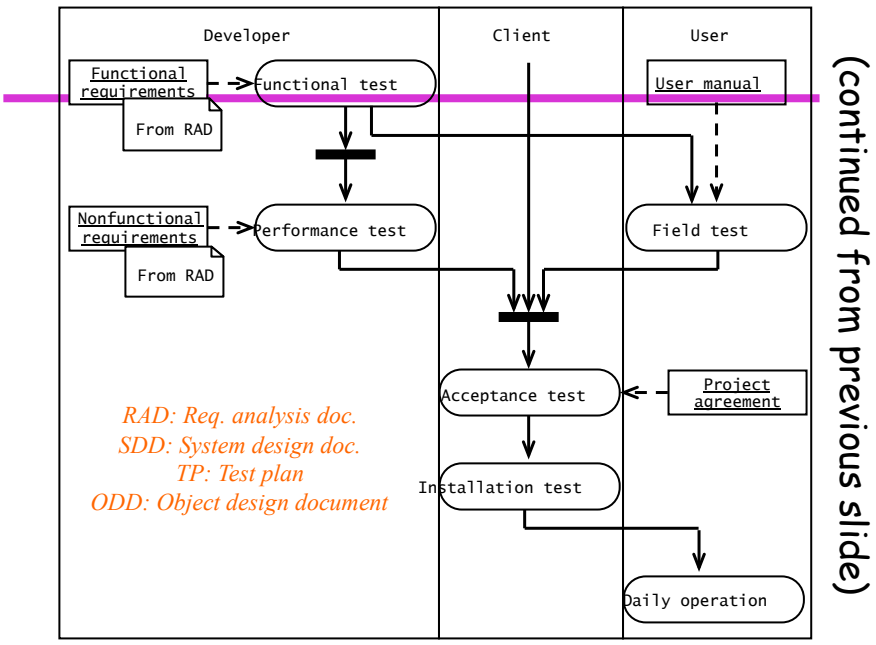
Test Results

- Documenting of test results is a crucial element of testing.
  - When was a test suite run (day, time, etc.)?
  - What is the summary count of each type of verdict?
  - What are the specific verdicts for each test case?

Testing Activities and their Related Work Products (B&D, Chap 11).



This is anything but Agile…



(continued from previous slide)