Testing within the development environment

...

Frederic Raison (CWG test)

Why testing ...

Requirements: “Tests shall be provided along with the source code.”

Yes ... but why ?

For quality and efficiency:

a study conducted by Microsoft and IBM showed that writing tests can add 15%-35% to the development time but reduce the number of defects by 40%-90%. *

* (http://research.microsoft.com/en-us/groups/ese/nagappan_tdd.pdf)

Outline

Source code testing principles:

unit tests, non-regression (or smoke) test, pre-integration tests, integration tests, validation tests

How to design unit test cases

How to implement integration-like tests

How to run the tests

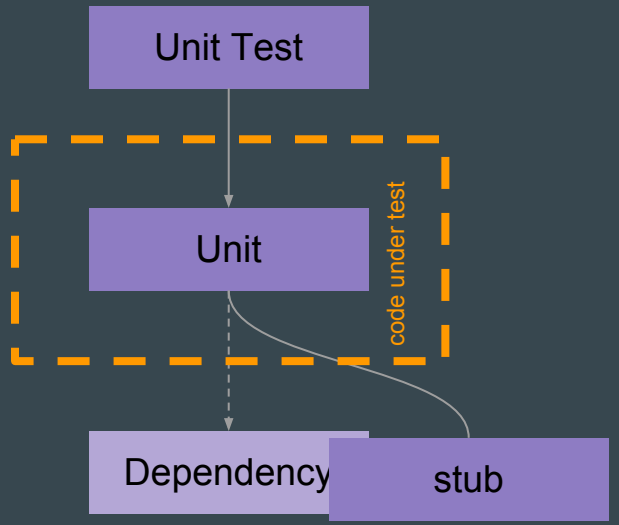
Tests principles

Principles: unit tests

Test individual software components (at class level)

Supply mocks, stubs or fake versions of dependencies (such as a database) so that the test does not rely on any external code and any failures can be pinpointed exactly

To be run during development before any commit



Principles: non regression or “smoke” tests

Check that current modifications don't affect the general behavior (subset of integration tests)

Combine components (classes)

To be run before a major commit with many side effects and before any commit if possible

Run on local virtual machines being representative of target environment

Principles: pre-integration tests

Considered as a subset of integration tests

Verify that all linked pieces of code are working well from an engineering/algorithm point of view

Do not mock or fake anything (use the real database and real network connections)

Run on local virtual machines being representative of target environment

Run before a major commit with many side effects

Principles: integration tests

Same code as pre-integration tests

Do not mock or fake anything (use the real database and real network connections)

Potentially skipped if pre-integration tests are enough representative

Run on target environment (CODEEN)

Principles: validation tests

Subset of integration tests

Verify that all linked pieces of code are working well (functionally/scientifically)

Run on target environment

Typically designed by the corresponding OU/SDC pair

Run before a major commit with many side effects

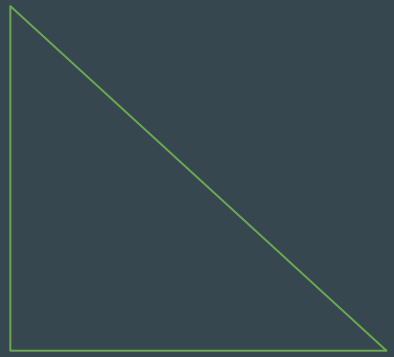
How to design unit test cases

Self assessment test [Myers]*

Imagine you are a software engineer for a company for a few minutes ...

"A program reads three integer values. The three values are interpreted as representing the lengths of the sides of a triangle.

The program prints a message that states whether the triangle is scalene (sides unequal in length), isosceles or equilateral."



Write a set of test cases to test this program

* *"The Art of Software Testing"*, Glenford J. Myers, Corey Sandler, Tom Badgett, Wiley & Sons, 2011

Self assessment test: solution I

1. valid scalene triangle ? (1,2,3 not valid)
2. valid equilateral triangle ?
3. valid isosceles triangle ?
4. 3 permutations of sides of an isosceles triangle ?
5. one side = 0 ?
6. one negative side ?
7. one side is sum of others ? (1,2,3 should not be valid)
8. 3 permutations of previous ?

Self assessment test: solution II

9. one side larger than sum of others ? (no valid triangle)
10. 3 permutations of previous ?
11. all sides = 0 ?
12. non-integer input ?
13. wrong number of values ? (2 instead of 3)
14. for each test case: is expected output specified ?
15. check behaviour after output was produced ?

note: we did not address memory management or instantiation

Self assessment test: comments

Highly qualified professional programmers score an average of 7.8/14

Quizz is intended to show how difficult it can be ...

What is the level of expectations for your code?

Unit test case design I

Normal conditions: (mandatory)

1. Rightness
2. Inverse relationships
3. Cross-check
4. Code logic

Abnormal Conditions and edges or trying the code to fail:

1. Boundary conditions
2. Error conditions
3. Exceptions

Unit test case design II: normal conditions

1 Rightness:

Validate the results against the requirements or from the developer point of view .

Ex: test of a method supposed to select the largest number from a list. Compare the returned result with the identified maximum from a list provided as input .

2 Inverse Relationships:

Ex 1: a method calculating the square root of a number can be checked by squaring the result.
Ex 2: method inserting a value in a file can be checked by searching this value after insertion.

Should be done with tools that are as independent as possible from the method to test (other library, ...).

Unit test case design III: normal conditions

3 Cross-check using other means:

Ex: if an algorithm is used to perform a calculation, another less accurate or less performant one can be used for validation. Or analytical result can be compared to a numerical calculations for values or conditions where it is possible.

4 Code logic:

Ex: “if ((x or y) and z) then decision1 else decision 2” where x, y and z are called "conditions".
Combinations of conditions could be unexpected and lead to a wrong decision. Some decisions are never reached because a different combinations of conditions always lead to the same logical value.

Unit test case design IV: abnormal conditions & edges

1 Boundary conditions: conditions at the limit of the domain in a general meaning (like format), not restricted only to a data partition .

Conformance: does the value conform to an expected format?

Ex 1: try using badly formatted input like e-mail incomplete address ("sdc@foobar.") or a missing ("@").

Ex 2: a missing header in a file used as input.

Ordering: is the set of values ordered or unordered as appropriate?

Ex: try handling a pre-sorted list to a sort algorithm or even a reverse-sorted list

Range: is the value within reasonable minimum and maximum values?

Ex 1: what is happening when a value is far in excess of reasonable expectations, such as a magnitude of 10,000?

Ex 2: crash when a value of index is out of range. Any indexing concept should be extensively tested.

Unit test case design V: abnormal conditions & edges

Reference: does the code reference anything external that is not under direct control of the code itself?

Ex: a method expects some value provided by another object which is not instantiated.

Existence: does the value exist (e.g., is non-null, non-zero, present in a set, etc.) ?

Ex 1: empty or missing values (such as 0, 0.0, "", or null)

Ex 2: query of a table in a database. What if the table does not exist or if the server is switched off ?

Cardinality: are there exactly enough value? (concept of equivalence partitioning)

Ex 1: partitioning data into n segments. The "0-1-nRule" checks if the code behaves correctly, when n=0 , 1 and n >1.

Ex 2: ranking a list of 10 SDC available to process data % network speed. What happen with 0,1 or 10 SDC available?

Unit test case design VI: abnormal conditions & edges

Edges of a partition:

Ex 1: (see http://en.wikipedia.org/wiki/Boundary-value_analysis)

the input parameter 'month' (integer) might have the following partitions:

```
... -2 -1       0 1 ..... 12 13  14  15 .....
 -------------|-----------------|-----------------
```

invalid partition 1 valid partition invalid partition 2

boundary values at 0,1 and 12,13 should be tested.

Time: is everything happening in order At the right time? In time?

Ex 1 with relative time: order of method call. What if methods to access a data system are called in the wrong order (connect, read, ...)?

Ex 2 with absolute time: what if the time system which is used (UTC, local time, ...) is not always the same?

Ex3 concurrency: how threads are accessing data? Are they synchronized?

Unit test case design VI: abnormal conditions & edges

2 Force error conditions: can you force error conditions to happen?

Examples:

- running out of memory
- running out of disk space
- issues with wall-clock time
- network availability and errors
- system load.

3 Exceptions: declared exceptions are tested?

If a method is supposed to throw an exception under certain conditions (trying to read missing file, missing entry, ...), check that the exception is indeed thrown.

Unit test case design VII: Summary

See the checklist in Annexe

Mindset: Try to find where the code can break ...

How to implement integration tests

Implementation : general remarks

The test data sets may be stored along with the test source code repository for unit tests and small sets. For larger amount of data, they shall be stored on SGS distributed storage and pointed at test source code repository.

In general, it is best to put most of the effort into building a solid suite of unit tests, and then adding a few integration tests for each major feature in order to detect any catastrophic incompatibilities or configuration errors.

How to implement integration-like tests I

Define the scope of the test: if dependencies correctly documented, test won't fail because of a missing database or web server.
Use detailed logging statements: when an integration test fails, it is more complicated than unit tests.

non regression or "smoke" tests

define a combination of the most important modules which interact together

the test should be quick

- the input data sets should be limited to the strict minimum and
- select the parameters and options accordingly

the test will pass/fail if the run was successful

pre-integration tests and integration tests

same code but pre-integration test are run locally (LODEEN) and integration tests are run in on the target platform (CODEEN)

define one or two stacks of the modules representative of the processing function

the test will pass/fail if each run was successful

How to implement integration-like tests II

validation tests

- tests are run in on the target platform (CODEEN)
- define one or two stacks of the modules representative of the processing function
- the test will pass/fail if each run produced scientifically validated results

How to run the tests

Write a script in directory “script”

Ex: bash 'ScriptThatChecksFile_test' launches a python executable and checks if it produced the file we expect.

Update the “CMakeLists”

Use tags (SmokeTest, PreIntegrationTest, IntegrationTest,ValidationTest) to select the tests to be run.

elements_install_scripts()
elements_add_test(ScriptThatChecksFile COMMAND ScriptThatChecksFile_test LABELS IntegrationTest)
name of the test — filename of the script

Run the tests

make purge all ARGS="-L IntegrationTest" test

"LastTest.log" is providing a full test log.

Source of information

Check http://euclid.roe.ac.uk/projects/codeen-users/wiki/User_Cod_Tst

Link to Pytest user manual:

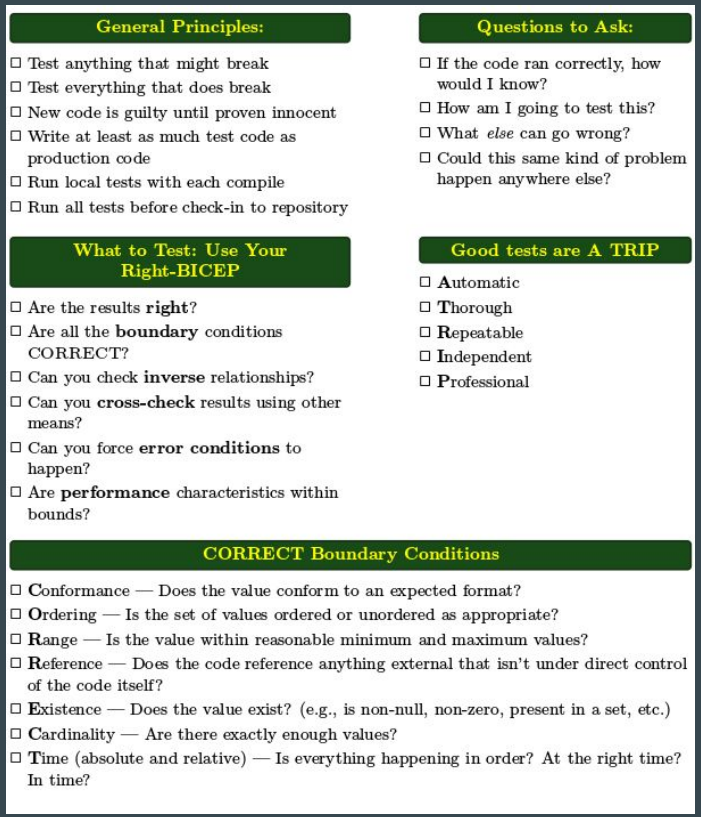
- pytest -params
- fixtures (scope, parametrizing, ...)
- monkey patching to come

Link to boost manual:

- mocks

leave your feedback on the forum: http://euclid.roe.ac.uk/projects/codeen-users/boards

Annexe I: Checklist for unit tests



General Principles:

- ☐ Test anything that might break
- ☐ Test everything that does break
- ☐ New code is guilty until proven innocent
- ☐ Write at least as much test code as production code
- ☐ Run local tests with each compile
- ☐ Run all tests before check-in to repository

Questions to Ask:

- ☐ If the code ran correctly, how would I know?
- ☐ How am I going to test this?
- ☐ What *else* can go wrong?
- ☐ Could this same kind of problem happen anywhere else?

What to Test: Use Your Right-BICEP

- ☐ Are the results **right**?
- ☐ Are all the **boundary** conditions CORRECT?
- ☐ Can you check **inverse** relationships?
- ☐ Can you **cross-check** results using other means?
- ☐ Can you force **error conditions** to happen?
- ☐ Are **performance** characteristics within bounds?

Good tests are A TRIP

- ☐ **A**utomatic
- ☐ **T**horough
- ☐ **R**epeatable
- ☐ **I**ndependent
- ☐ **P**rofessional

CORRECT Boundary Conditions

- ☐ **C**onformance — Does the value conform to an expected format?
- ☐ **O**rdering — Is the set of values ordered or unordered as appropriate?
- ☐ **R**ange — Is the value within reasonable minimum and maximum values?
- ☐ **R**eference — Does the code reference anything external that isn't under direct control of the code itself?
- ☐ **E**xistence — Does the value exist? (e.g., is non-null, non-zero, present in a set, etc.)
- ☐ **C**ardinality — Are there exactly enough values?
- ☐ **T**ime (absolute and relative) — Is everything happening in order? At the right time? In time?