Chapter 5
Testing

Testing

Testing is an activity performed for evaluating product quality, and for improving it, by identifying defects and problems

IEEE Software Engineering Body of Knowledge
www.swebok.org

Why?

Testing should be done by the developers, it improves their performance.

Testing should be done by an independent team who have no preconceptions about its operation.

Software testing consists of the ***dynamic*** verification of the behaviour of a program on a ***finite*** set of test cases, suitably ***selected*** from the usually infinite execution domains, against the ***expected*** behaviour.

What?

Dynamic opposed to walk throughs, inspections etc.

Selected: key challenge is to identify tests which are most likely to expose failures. Multiple tests which challenge the same code are a waste of time to write and a waste of time to run.

Failure is undesired behaviour,

Fault is the cause of the failure

Scale: Unit; Component; Integration; System
Characteristic: Functional; Robustness; Performance; Usability

Context i

Different system modelling and testing strategies are recommended for developing programmes in different contexts

Obvious, but …

eg normal techniques are not suitable for real time programming where a timing is part of the specs

Distributed/parallel computing is a harder and less developed way of organising computing & represents a new set of problems which *may not have well developed solutions*.

Testing is normally divided into a number of types, which reflect a way of developing code.

That means testing strategies are divided differently according to their target application area.

That means testing strategies are divided differently according to their target application area.

Add to that different suppliers of testing software have an interested in differentiating their products.

And that the boundaries between different sorts of tests are fuzzy and you have a recipe for confusion and multiple conflicting names.

Solution

Describe the testing techniques which I have used in particle physics.

Large code base: multi-million line codes

Diverse developer population: hundreds of people spread across the world, with no centralised control.

Resource sensitive: we have more data than we can comfortably handle: RAM, CPU time; disk space are all in short supply.

Developing against hard deadlines.

High stakes: production of results for publication depends on the accuracy of the software and an erroneous result will have a destructive effect on the career of the people involved.

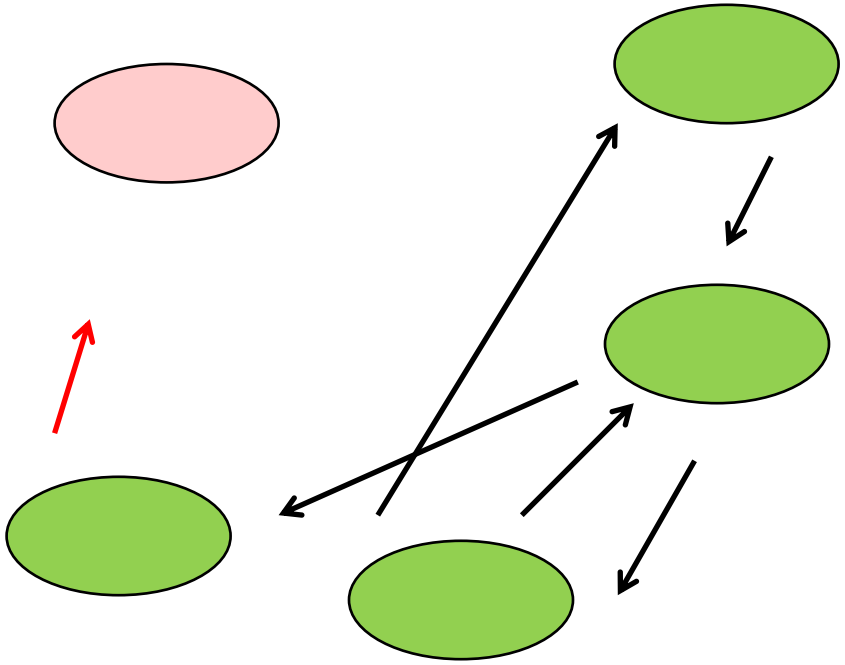
Solution

Rigorous testing is essential.

I will talk about testing types, concentrating on a high level view with emphasis on commonality and will lump together ideas which others may choose to separate.

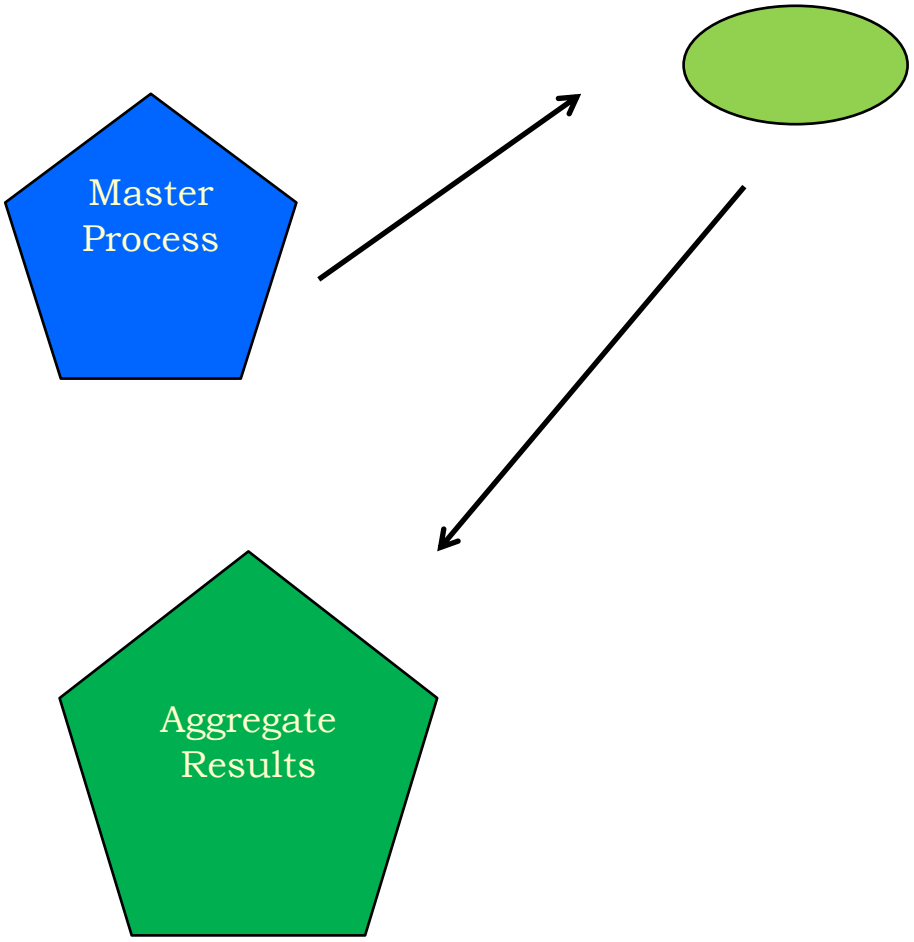
Advantage: less ideas to learn;
a hierarchy to fit ideas into

Interacting systems with partial failure



What now?

Parallel system with task distribution



If the subtask does not return – then resubmit.

But what happens if after the duplicate task completes the original task responds.

It is vital that the standard non-distributed testing procedures are carefully followed.

Conventional testing

White Box	Code logic visible to tester
Black Box	Provide input and verifies output

White Box	Unit tests (Integration tests)
Black Box	Unit tests Integration tests Functional tests Stress/Load tests Acceptance tests

Unit tests	Does the smallest operational unit work correctly (method)
Integration tests	Do the programming units (classes) work together
Functional tests	Do the functions/services provided by the application work in isolation
Stress/Load test	Operation in practice
Acceptance test	Is the sponsor/purchaser happy

Unit Test

Smallest unit of code which does something.

In object oriented code it normal corresponds to a method (including the constructor)

It is here that we most normally white box test.

We may well develop the algorithm by hand and only then enter it as code.

We will normally work through a number of cases to ensure that the algorithm works.

Implement the code and then check the operation against the test cases.

Unit Test

Those test cases can then become black box tests.

Deliver the input to the function and check the output.

Each test should be completely independent of all the others.

Implies objects should be constructed independently for each test.

Tests may pass in a set but fail when run in a different order

One test may leave the object in a state which allows a second test to pass – unwanted side effect.

Errors must detected; reported; and recorded automatically.

Implies a testing framework

Integration Test

There exists Classes each with its associated methods, which have been tested.

Solving a problem should involve the interaction of these classes.

An integration test will create a number of objects whose interactions mimic the interactions which will be seen the complete application.

Integration tests may be white box, working through the interactions by hand; or black box.

White box bigger than integration is very hard to do.

Functional Test

An application will normal provide a number of services or functions to the user.

Ensuring that each of those services operates correctly in isolation is the definition I prefer for functional tests.

Some authors refer to anything which tests a "function" as a "functional test". This is then contrasted with non-functional tests.

'When **I** use a word,' Humpty Dumpty said, in rather a scornful tone, 'it means just what I choose it to mean — neither more nor less.'
Alice in Wonderland: Lewis Carroll

I haven't found a better word to describe the idea of a functional

Functional Test

The requirements document(s) should include the specification, which allow the statement: “the services runs as defined in the specification”.

A test or set of tests which allows that statement to be made is what I would call a functional test.

The existence of such a test allows part of an application to be shipped, which can be useful.

My example we can identify one sort of elementary particle, an electron say but not other sorts

Load Test

Under typical operating conditions does the application respond appropriately.

Mimics the system on which the application runs and ensures that it fulfils the requirements.

Processor type; available RAM; available disk space; network response time typical; database size and simultaneous users; other applications running.

Under those circumstances does the system run “correctly”

Load Test

Primary question may be do we have appropriate response time.

Response time is application dependant.

Word processor: does it respond to the user's input, consistently.

Business analysis software: does it run over night.

Database for telesales operation, does it respond in never more than x seconds.

Load Test

In the case where response is more central to the operation of a system

Fire Alarm: soft real time

Fly by wire aircraft: real time

Then the timing test is likely to more central to the testing procedure and to occur earlier in the testing stack.

Load Test

Failure …

Can we solve with a hardware upgrade ?

Search for a bottleneck

java-source.net listed

x open source utilities

BEWARE OPTIMISING TOO SOON

Should the algorithm efficiency be tested sooner

White Box testing during design

JUNITPERF allows measurement/optimisation at Unit test stage

Should you test early	YES
Should you optimise early	Only with good reason

Stress Test

This is what happens when the system is overloaded or under provisioned.

Too many users or too little memory.

An overloaded resource: network, database

Suitable software JMETER.

Graceful degradation

Slow response.

Rejection of commands or refusal to allow new users.

Errors

System crash

Return erroneous results

Stress Test

Some people define it as operation under conditions which are challenging.

Not useful

I think it is more useful to think of it as in what circumstances does the system break,

How serious is such a failure?

How far from failure is normal operating conditions do problems occur.

What problems can occur:

- Erroneous response or action
- Crash: error in preserved state
- No response (arbitrary slow)
- Crash: no error in preserved state
- Refusal to operate; load shedding

Distributed system; crash and arbitrary long response time may not be distinguishable

Test type

Regression tests

Distinction sometimes made: tests made when anything is changed.

When operation discloses an error in the code.

When a planned enhancement is implemented.

Not full set of tests, because too expensive in time

Nothing is more expensive than a programme that produces unreliable results.

"There is no point in producing a programme with errors"
Tony Hey. Prof of Computing

"There is never time to do it right, but there is always time to do it again"

Regression tests

I do not believe that the distinction is real.

Any programme should pass all tests before shipping, and all tests need to be passed after **any** changes.

Acceptance tests

Does it fulfil the requirements.

Doesn’t help in assuring correct operation

Does ensure that the customer/sponsor gets what they asked for.

The Night Before Implementation

Twas the night before implementation and all through the house,
Not a program was working, not even a browse.
The programmers hung by their screens in despair,
with hopes that a miracle would soon be there.
The users were nestled all snug in their beds,
while visions of inquires danced in their heads.
When out in the hallway there arose such a clatter,
I sprang from my desk to see what was the matter.
And what to my wondering eyes should appear,
but a super programmer with a six-pack of beer.
His resume glowed with experience so rare,
he turned out great code with a bit-pusher's flair.
More rapid than eagles, his programs they came,
and he cursed and muttered and called them by name.
On update! On add! On inquiry! On delete!
On batch jobs! On closing! On functions complete!
His eyes were glazed over, fingers nimble and lean,
from weekends and nights in front of the screen.
A wink of his eye, and a twitch of his head,
soon gave me to know I had nothing to dread.
He spoke not a word, but went straight to his work,
turning specs into code, then turned with a jerk.
And laying his fingers upon the "ENTER" key,

The Night Before Implementation

....

the system came up and worked perfectly.
The updates updated, the deletes they deleted,
the inquires inquired, and closings completed.
He tested each whistle, and tested each bell,
with nary an abend, and all had gone well.
The system was finished, the tests were concluded,
the users' last changes were even included.

And the user exclaimed with a snarl and a taunt,
"It's just what I asked for, but not what I want!"

Quality

Testing ensures code quality

it produces it only indirectly

It must be part of the development process

not an add on at the end

Test quality is also important, poor tests will not ensure good code.

How do we ensure that tests are doing their job?

How do we write good tests?