Testing Software

What should we do? What can we do?

David Lorge Parnas
Telecommunications Research Institute of Ontario
Communications Research Laboratory
Department of Electrical and Computer Engineering
McMaster University, Hamilton, Ontario Canada L8S 4K1

Why is Software A Special Concern?

(1) It never works the first time it is really used.
(2) It has no natural internal boundaries.
(3) It is sensitive to minor errors - there is no meaning to "almost right". (chaotic behaviour)
(4) It is difficult to test because interpolation is not valid.
(5) There are "sleeper bugs".

These are all manifestations of complexity.

They are "inherent" properties, not signs of immaturity in the field.

Novice Approaches To Testing

(1) Eureka! It ran.

- one test means its done
- usually a very simple case.

(2) A number of tests where the answers are easily checked.

(3) Let it run and run and run.

(4) If an error is noticed, fix and go to 2.

What's Wrong with This?

5) Much of the program may never be tested

6) All we get is a bunch of anecdotes

How Much Testing is Enough?

How important is the product?

How do you want to measure quality?

Some Better Approaches to Testing

1) Test uses every statement at least once.

2) Test takes every exit from a branch at least once.

3) Test takes all possible paths through the program at least once.

These are minimal requirements, but...

They mistakenly assume that program state is more important than data state.

Additional Rules

(4) Consider all typical data states

(5) Consider all degenerate data states

(6) Consider extreme cases

(7) Consider erroneous cases.

(8) Try very large numbers

(9) Try very small numbers

(10)Try numbers that are close to each other.

(11)Think of the cases that nobody thinks of.

Who Does the Testing?

You are a fool if you don't test your program!

Your customer/boss is a fool if they don't test your program?

Many software companies have testing specialists in quality assurance companies.

"Cleanroom" model says that you are not allowed to test your program.

- Increased care yields big improvements in quality.
- Statistical testing is done by others.

The basic issues:

Human beings all tend to overlook the same cases.

How can random testing be better than planned testing?

Can we have planned random testing?

Three Kinds of Testing

Black Box Testing
- •Based on Specification Alone
- •Cases chosen without looking at code

Clear Box Testing
- •Test Choice Based on Code
- •Use coverage criteria described earlier

Grey Box Testing
- •Intended for modules with memory
- •Look at Data structure
- •Assures that state coverage is good
- •May use design documentation as a further check

All have their place
- •Black Box testing can be re-used with new design
- •Black Box testing can be independent of designer
- •Clear Box testing tests the mechanism
- •Grey Box testing gives better coverage for black box with memory. Avoids some duplicate tests

Another Testing Method Classification

Planned Testing
- •Clear Box - based on code coverage criteria
- •Black Box - based on external case coverage

Wild Random Testing
- •Pick arguments using uniform random distribution
- •Can find cases nobody ever thinks of
- •Can violate assumptions yielding spurious errors
- •reliability figures can be obtained but aren't meaningful

Statistical Random Testing.
- •Requires an operational profile
- •Provides meaningful reliability figures
- •Only as good as the operational profile.

Hierarchical Testing Policies

Testing the whole system at once is a disaster.

Finding the fault is a nightmare.

Test Small Units first.

Integrate after components have passed all tests.

Test lower levels of uses hierarchy before using them.

Bottom Line:

A delay before integration will save time after integration.

MEASURES OF SOFTWARE QUALITY

We must assume the existence of a specification

- We need the ability to tell "right" from "wrong"

Correctness:

Does the software always meet the specification?

Reliability:

The probability of correct behaviour is?

Trustworthiness:

Is there a low probability that catastrophic flaws remain after all verification.

Each of these measures is different, each requires a different verification method.

WHEN SHOULD WE USE EACH OF THESE QUALITY MEASURES?

Correctness:

Rarely need it!

Nice to reach for, hard to get.

To a perfectionist, all things are equally important.

Not our real concern, we accept imperfections.

Use formal methods and rigorous proof.

If you have a small finite state space, you can do an exhaustive search.

Binary Decision Diagrams, (BDDs) handle slightly bigger cases.

WHEN SHOULD WE USE EACH OF THESE QUALITY MEASURES?

Reliability:

when we can consider all errors are equally important,

when there are no unacceptable failures,

when operating conditions are predictable,

when we can talk about the expected cost,

when your concern is inconvenience**,**

when we want to compare risks.

Use Testing, both Statistical and Planned.

WHEN SHOULD WE USE EACH OF THESE QUALITY MEASURES?

Trustworthiness:

when you can identify unacceptable failures,

when trust is vital to meeting the requirements,

when there may be antagonistic "users".

We often accept the systems that are unreliable.

We do not use systems that we dare not trust.

Testing does not work for trustworthiness.

Use formal documentation and systematic inspections.

WHAT ARE THE LIMITS OF SOFTWARE TESTING?

Testing can show the presence of bugs but never their absence. (E.W. Dijkstra)

- False in theory, but true in practice,

It is impractical to use testing to demonstrate trustworthiness.

One *can* use testing to assess reliability.

Two sides of a coin:

- I would not trust an untested program!
- At Darlington we found serious errors in programs that had been tested for years!

WHAT ARE THE LIMITS OF SOFTWARE TESTING?

1) It is not usually practical to prove correctness by testing.
2) Testing cannot predict availability.
3) Reliability predictions based on old versions are not valid.
4) Testers make the same assumptions as the programmers.
5) Planned testing is a source of anecdotes, not data (H.D. Mills).
6) Self-tests test for decay, but not built-in defects.

Formal Methods complement testing.

Even in "Black Box" testing, what's inside does make a difference!

The number of tests needed to identify a finite state machine depends on an upperbound for the number of states.

The length of a test-trajectory will depend on the memory characteristics of the system.

WHAT DOES IT MEAN TO TALK ABOUT SOFTWARE RELIABILITY?

Software is not a random process.

It is the input data that introduce randomness.

"Software Reliability" is a measure of the input distribution through a boolean filter.

Software cannot be assessed as a set of components.

Software + Hardware must be assessed as a single component.

Formal Methods contribute directly to trustworthiness and correctness but less directly to reliability because of the importance of the input distribution.

MEANINGLESS MEASURES USED FOR SOFTWARE RELIABILITY

1) The number of errors per 1000 llines.
2) Time derivatives of the number of errors per line.

It is the failure rate that matters.

Formal methods do help the first two!

How much testing is needed to assess reliability?

1) Assume that we have the right input distribution (difficult). We will use tests selected randomly from this distribution.
2) Let 1/h be the required reliability.
3) What is the probability of passing N properly selected tests if each test would fail with probability 1/h?

$$M = (1 - 1/h)^N$$

4) M is the probability that a marginal product would pass a test sequence of length N.
5) Some examples for h = 1000
   - N = 500, M = 0.606
   - N = 1000, M = .36700
   - N = 5000, M = .00672
6) Some examples for h = 1,000,000
   - N = 1,000, 000, M = .36788
   - N = 5000000, N = .00674

Critical Systems, systems with high reliability requirements, are much harder to test.

Expert opinion says you can't do it.

Some companies use users to get more tests.

Real-time systems are harder to test than batch (memory free) systems

In real-time systems, a test is a trajectory, not an input state.

The trajectory must be long enough that sleeper bugs are revealed.

There must be an upper limit to the memory of the system.

Systems must be structured with testing in mind.

Most of the memory must be periodically reinitialised.

Testing must be repeated for each mode of the remaining memory.

Defining the probability distribution of trajectories is the hardest part.

Formal methods are more important in real-time applications

Conclusions

1) Safety-Critical systems require both inspection and testing.

2) Engineering training is essential.

3) Engineers see testing and mathematical analysis as complementary and both are necessary.

Let's worry about first-things first.

If we cannot use these methods for documentation and inspection, they won't be good for the more advanced applications.

The Critical-Software TRIPOD

1) Precise, Organised, Mathematical Documentation and Extensive Systematic Review.

2) Extensive Testing
   - Systematic Testing-quick discovery of gross errors
   - Random Testing -discovery of shared oversights and reliability assessment

3) Qualified People and Approved Processes.

What Should You Do?

- Take Testing Seriously.
- Make a Test Plan
- Keep Complete Logs.
- Think about what you really know after a test has been passed.