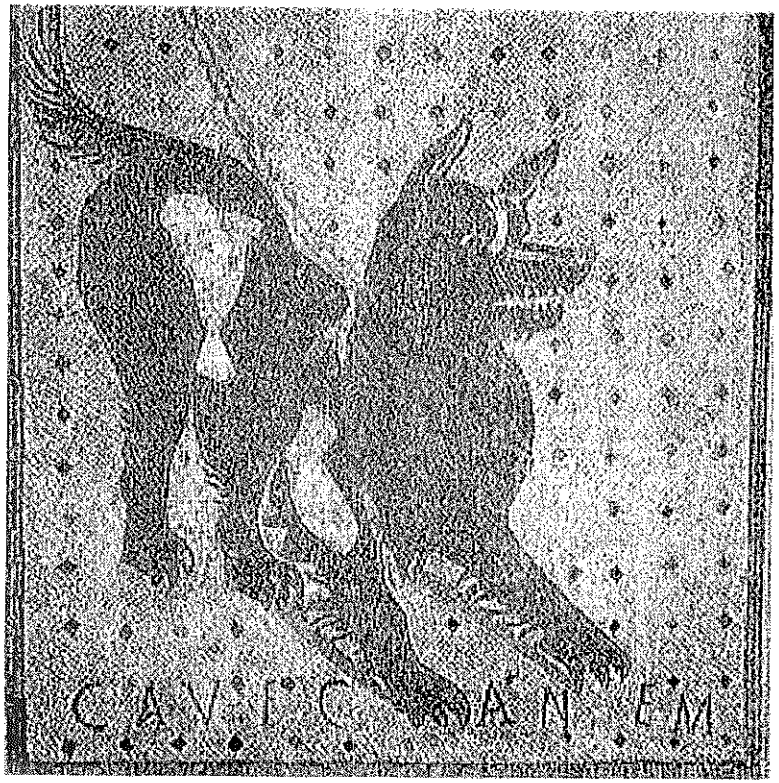


SW Reliability & Testing Metrics

Rome

26 & 27 November 1998

Agenda: 26 November

- 9.00 - 9.30 **TEI GPC Presentation** S. Improta
- 9.30 - 9.35 ***Agenda*** *E. Peciola*
- 9.35 - 10.30 Invited speaker: **Software Testing: Theory, Techniques and Magic** *A. Bertolino*
- 10.30 - 10.50 Coffee Break
- 10.50 -10.55 ***Open Session: Reliability Models*** *E. Peciola*
- 10.55 - 11.20 **The Use of Reliability Growth Models in Project Management** *E. Miranda*
- 11.20 11.50 **Workflow for evaluating software reliability by apply SRE models to field data** *G. Lombardi*
- 11.50 - 12.45 Invited speaker: **Models for comparing the effectiveness of testing** *L. Strigini*
- 12.45 - 14.00 Lunch
- 14.00 - 14.05 ***Open Session: Management Measurement*** *A. Bertolino*
- 14.05 - 14.30 **The Application of Code Based Metrics to a Proprietary Language** *J. Evans*
- 14.30 - 15.15 **Quantitative methods in CMM** *M. Spolverini*
- 15.15 - 15.35 Coffee Break
- 15.35 - 16.30 Invited Speaker: **Modelling the effects of combining diverse fault removal techniques** *B. Littlewood*
- 17.00 - 19.00 Visit to Galleria Borghese Museum
- 19.30 Dinner at 'Il Tinello'

Software Testing: Theory, Techniques and Magic

Antonia Bertolino

bertolino@iei.pi.cnr.it

Istituto di Elaborazione della Informazione, CNR, Pisa (Italy)

Testing is an unquestioned and expensive part of the software development process. The software testing technology is more than three decades old: Beizer, one of the renowned authorities in the field, provocatively claims it is now obsolete. He dates to the mid 60's the earliest published papers on software testing, and to a few years before the first attempts of a systematic testing practice, as distinguished from debugging. In the years, a large variety of testing methods has been invented, several of them are now routinely used, thousands of technical articles on the topic have appeared in conferences and journals, some of which specialized in software testing, and hundreds of automated testing tools have been put in commerce.

Regrettably, this apparent maturity of the testing technology is still not backed by a correspondingly mature theory. A sound theory of testing should provide the tester with a solid framework in which testing methods can be classified and their respective cost/effectiveness evaluated. More importantly, a true scientific discipline should provide managers with a means to assess in objective way the results achieved by a testing method and to control step by step the testing process. Unfortunately, the state of industrial practice is well behind such expectations: as a result, testing costs and schedules remain hardly predictable.

Software testing consists of observing a program's behavior onto a (manageable) set of valued inputs appropriately chosen from the potentially infinite input domain. This dynamic verification can have different goals. The testing techniques described in textbooks having the word "testing" in their title or more commonly used in the industry are generally aimed at revealing as many failures as possible, so that the faults that caused them can be located and removed. Past research in software testing has focused much attention on the "incremental" definition of systematic testing methods, perhaps in the implicit (and clearly hopeless) assumption that the right method, once identified, could magically deliver the right program. However, scarce attempts have been made to establish a meaningful relationship between the use of such methods and the resulting

quality of the program tested according to them. The many studies published have certainly improved our knowledge on the functioning of the methods analyzed, and on how a program should be carefully scrutinized. But, the underlying assumptions of these methods were not always made clear enough, often confusing the effort spent in testing with the results achieved.

As managers know well, "heroic" debugging may be necessary, but never sufficient to guarantee the success of a project. The point is that admittedly we can never hope to test away the last failure, and residual faults can produce vastly different effects, depending on whether and how often they will be excited after release, as well as on their consequences. In contrast with systematic techniques, operational approaches to testing take this fact into account, and are purposely devised to reveal not all failures, but those that can have a greater impact on the delivered reliability. For this purpose, a parallel research direction has developed, in which testing techniques from conventional reliability theory have been adapted and applied to software engineering. Statistical inference methods and several software reliability growth models can now routinely be used to predict the future reliability of a program based on the failures observed during testing.

In this talk I will provide an introductory overview of software testing concepts, theoretical issues and state-of-the-art techniques. A test criterion maps a piece of software to a test suite: it can be used to select the tests or to decide if the tests are adequate. I will discuss state-of-the-art criteria, both deterministic and statistical, with emphasis on their relative properties and limitations. Systematic criteria, based on a deep analysis of the specification or of the code, bear an appearance of scientific approaches. On the other hand, operational approaches, by merely sampling the input space at random, would seem to give lower chances of revealing failures. Yet, studies showed that this is not the case, and that the performances of the two approaches, systematic vs. random, can be comparable, or one can outperform the other, depending on the specific situation and the underlying (and most often implicit) assumptions. What can be said in general is that statistical testing consists of a controlled experiment, and as such offers objective results. On the contrary, as several empirical studies have confirmed, deterministic criteria lay heavily on the skill and ingenuity of the human testers, with the pros and cons of such a reality.

Software Testing: Theory, Techniques and Magic

Antonia Bertolino
IEI-CNR (Pisa)

- Which is a "good" test suite?
- How can I find such a good test suite, or at least a good approximation?
- Is it better to select tests systematically or at random?
- Deterministic testing criteria vs. operational testing approaches

A mature technology?

Beizer claims that testing technology is now obsolete.

In fact we have:

an accepted vocabulary;

an organized body of knowledge (textbooks, handbooks, scientific articles, specialized conferences, standards, ...);

a great variety of techniques and tools, now validated by years of practice

Why are we convening here?

(... what am I investigating?)

The point is that software testing technology is not yet sustained by a sound, accepted theory.

Many fundamental questions still open:

- *testing procedures and testing results are often confused (which is the goal of testing?);*
- *having applied a testing technique and having observed no (or a given number of) failures, what have we achieved?*
- *which is the most adequate testing technique under given circumstances?*
- *having tested a piece of software in a context, what can we infer about the functioning of that software in a different context (component testing)?*

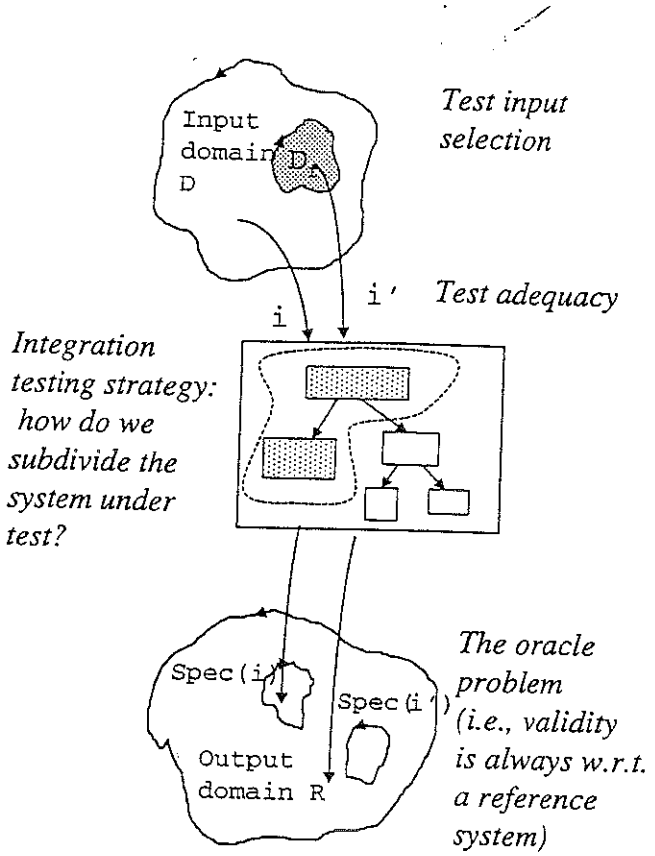
Software testing consists of:

the dynamic verification of the behaviour of a program

on a finite set of test cases

suitably selected from the (usually infinite) input domain

against the specified expected behaviour.



The fault-failure model

A fault will manifest itself as a failure if all of the 3 following conditions hold:

1) EXECUTION: "the faulty piece of code is executed" (a failure-causing input is picked)

2) INFECTION: the internal state of the program is corrupted: *error*

3) PROPAGATION: the error propagates to program output

An ideal testing criterion

B. Goodenough, S. L. Gerhart (1975)

An ideal test criterion C for the selection of test data must be **reliable** and **valid**, where:

- C is said to be *reliable* if, whenever C selects test sets T1 and T2, a program P either is successful both, or fails for both.
- C is said to be *valid* if, whenever P is not correct, C selects at least a test input on which P fails.

If an ideal test suite could be derived,

its successful execution would constitute a proof of correctness (would show the absence of faults).

Or, the only test criterion *guaranteed* to be both reliable and valid is the exhaustive one.

What this theory actually tells us is:

- a criterion's goodness depends on the program P, and specifically on the faults present in it;
- as a consequence, it is not preserved as a program is changed for effect of fault removal.

The notion of "revealing subdomains"

Weyuker and Ostrand (1980)

- A subdomain S of inputs is *revealing* if whenever it contains some failure-causing input, the program fails on any non-empty subset of S.

- In systematic testing what we try to achieve more realistically are subdomains revealing for *specific* types of faults.
  (=> techniques of fault based testing)

How do we identify "good" subdomains?

- A subdomain would be a set of inputs which -based on the specification- should be treated the same,
- and which -based on program structure- are all processed the same way (path-oriented test criteria)

⇒ This is why functional, structural and special value testing approaches should be used in combination

Test cases within a subdomain are picked:

Subdomains are derived based on:

	DETERMINISTIC	RANDOM
FUNCTION	Spec-based	E.g., SMARTS tool
CODE	Branch coverage	Statistical testing (a la Thevenod)
FAULTS	Special value	Mutation
FIELD USAGE	Acceptance testing	Operational testing

Many testing criteria aimed at testing away "all failures"

Approaches are typically systematic and are based on some kind of "coverage".

For each given criterion, there exist *many different* sets of test cases satisfying it, and obviously the "best" test set can produce a very different effect than the "worst" one.

- W.r.t. branch coverage: if a branch containing a fault is never executed, then that fault very likely will never be found.
- But, what about the reverse? Will one execution necessarily reveal it?

THE HARD SYSTEMATIC TESTING APPROACH

"Try hard to reveal program failures, until you cannot find any more.

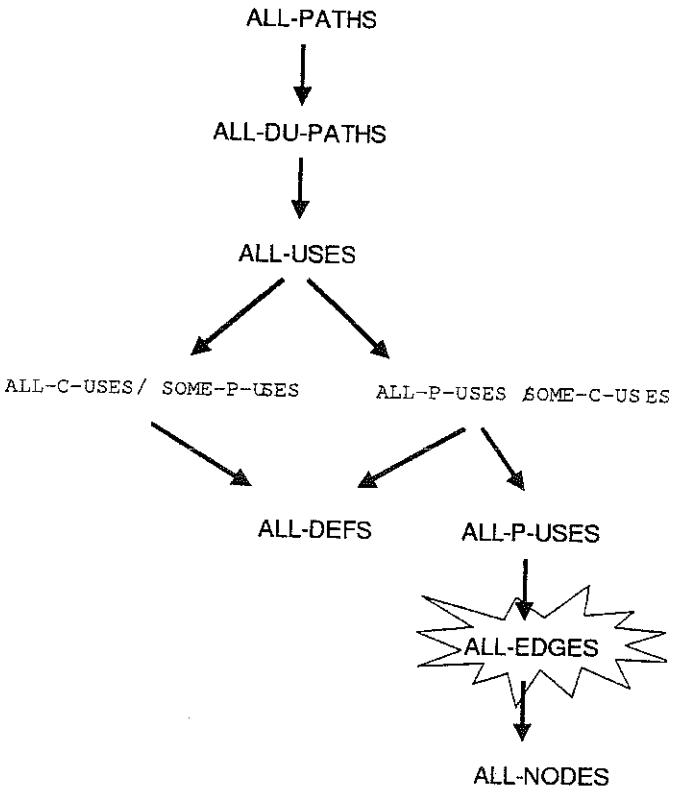
Then you can be quite confident that the program has no more faults left".

correct?!

The fisherman analogy (Hamlet, 1994)

We should not confuse the effort with the result.

A hierarchy of code-based criteria



Data-flow based testing

- Several criteria aimed at capturing the way data (variables) are manipulated
- Fills the gap between branch coverage and all-path coverage (which is impracticable)
- "Should be" better than branch coverage at fault finding

Definition-Use Associations

A dua is a triple [d,u,X], in which:

- variable **X** has a global definition in block **d** and a global use in block **u**

and

- there exists a **definition-clear path** w.r.t. **X** from **d** to **u**

The untestedness syndrome

From: *Every statement should be exercised*

To: *Every statement should be tested*
(should be exercised so that it affects the program output)

```
Program Silly
1  read ( x, y, w)
2  if w > x + 1
3       then x := y + 5
4  z := w
5  w := x + y
6  y := w / 2
7  if z < 1
8       then y := 100
9       else z := 100
10 write (y, z)
```

Consider paths:

1, 2, 3, 4, 5, 6, 7, 8, 10
1, 2, 4, 5, 6, 7, 9, 10

All statements are covered, but statement 3 remains untested. *We could avoid untestedness by forcing data flow dependence*

Problems with code-based testing criteria

- Untestedness syndrome
- "Missing requirements" (using such criteria for test selection is a tautology)
- Infeasible paths (applicable versions)

But, they can be (and should be) automated

Good for measuring "thoroughness" (what am I leaving out?)

Operational Testing

We cannot realistically presume to find and remove the last failure.

Then, we should invest test resources to find out the "biggest" ones.

Reliability measures:

IF the test suite is a representative sample of the field usage,

the test results can be exploited to make predictions about the product's reliability

=> testing is a statistical experiment

In operational testing, what do we do of observed failures?

If we remove them, reliability grows (hopefully ...)

Inference by use of reliability growth models

We leave them, "certification testing": *accept/reject* the product

Inference by confidence intervals, Bayesian estimates...

What is difficult in operational testing?

The part of statistical prediction is no problem (for "modest" requirements): many reliability growth models, practical tools and techniques for choosing the best one among them for one's specific case.

- **For new applications, it is difficult to identify the operational profile.**
- **Large number of tests needed for meaningful predictions (much bigger that usual coverage approaches).**
- **It is easy to automate test generation, but more difficult to check the test outputs.**

But according to experience, it is worth trying....

We should collect and analyze failure data even outside operational testing

Statistical analysis is fundamental to control and manage the testing and development process.

Reliability growth models require that reliability is increasing (as confirmed by trend tests).

Outside, we can use conventional confidence intervals or Bayesian models.

The Bayesian approach

- In statistical inference, we describe our state of knowledge about the world.
- Bayesian probability represents a measure of the plausibility of an event with incomplete knowledge: which is the best inference we can draw given (conditional on) the evidence available?

$$P_{posterior}(H_i \mid E) = \frac{P_{prior}(H_i)P(E \mid H_i)}{\sum_j P(E \mid H_j)}$$

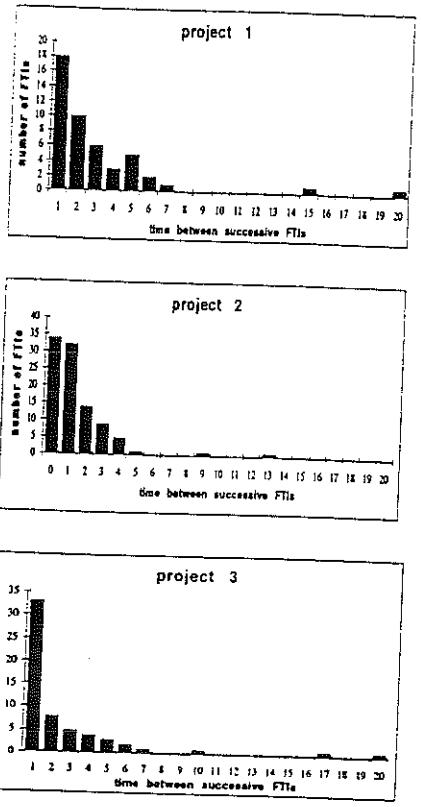
Example:

To predict the expected number of test failures, we grouped failure data into test intervals (TI) and defined a random variable Q as the probability that the next test interval is a failed one.

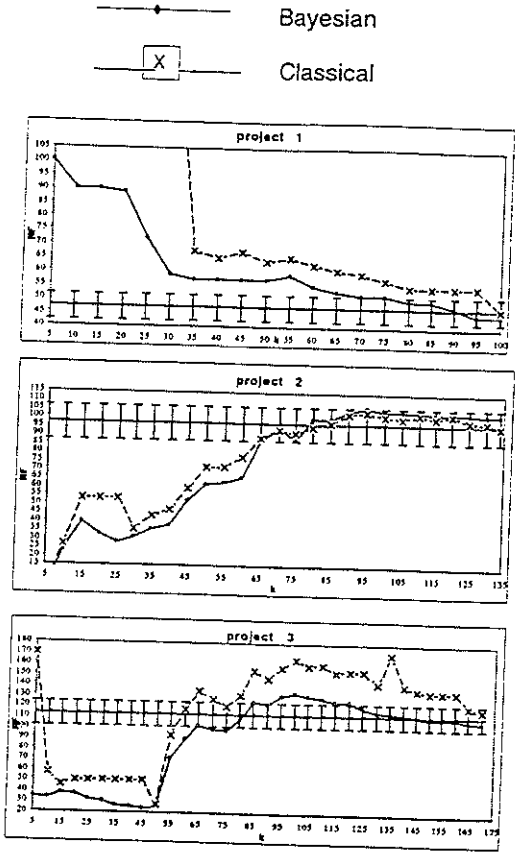
To derive a posterior distribution for Q (after observing k test intervals):

$$P(Q=\frac{1}{i} \mid F_k) = \frac{p_Q\left(\frac{1}{i}\right)\cdot\left(\frac{1}{i}\right)^f\left(1-\frac{1}{i}\right)^{k-f}}{\sum_{j=1}^{M} p_Q\left(\frac{1}{j}\right)\cdot\left(\frac{1}{j}\right)^f\left(1-\frac{1}{j}\right)^{k-f}}$$

Example: Failure data



Example: Predictions



Assessment of the relative effectiveness of different testing methods

"Effectiveness" can be considered from many different points of view:

- failure-detection capability;
- cost, or effort required to satisfy a criterion (subsumption hierarchy);
- acquaintance, "that is the method we have been using here";
- re-usability, portability, etc... of test suites;
- improvement of delivered reliability;
- assessment of future reliability;
- .....

Methods used early in the literature for comparing test criteria:

Analytical comparisons, methods ordered according to some notion of "ease of satisfaction"

Empirical comparisons w.r.t. the respective *probabilities* to detect at least one failure

Analytical comparisons:

Earlier approaches:

"Subsumes" hierarchy: *Method X subsumes method Y if and only if any test that satisfies X also satisfies Y.*

(...is this equal to say that X is "better" than Y?)

Or, one method is assumed as the reference standard, and the extent to which a number of test sets derived according to different methods satisfies the standard is analyzed.

Recent results from analytical comparisons.

(Rapps&Weyuker, 1985)

Pros: - many precise, published studies

- provided a deeper understanding of the theoretical properties of the methods

Cons: - many methods are incomparable

- not related to failure detection ability

(Frankl&Weyuker, 1993)

More meaningful relationships than "subsumes" are introduced, which could be related to the failure detection ability: "narrows", "covers", "partitions", "properly covers", "properly partitions".

For example, using the "properly covers" relationship the *all-uses* criterion is demonstrated better than the *all-branches* criterion.

Branch coverage vs. Any Nonempty Input Subset:

```
Function Q (x:real): real;
{should compute √|x|}
begin
  if x<0 then x:= -x;
  Q:= sqr(x)
end;
```

branch coverage subsumes ANIS, yet

[1, -1] achieves branch coverage and uncovers no failures

[2] (does not even cover all branches and) reveals the failure

Empirical comparisons

Pros: - quantitative evaluation of effectiveness

Cons: - dependence on programs used and on choice of test set for each method

Partition testing vs. Random testing

(Seminal study by Duran&Ntafos , 1984)
(Hamlet&Taylor, 1990)

where:

- partition testing is intended as "the input domain is partitioned into K disjoint subdomains and Ni tests (often 1) are taken from within each subdomain"

- random testing is intended as "N tests are taken over the whole input domain"

The FAILURE-RATE model:

The ratio $\frac{P_r}{P_P}$

is evaluated through experiments and simulation,

Where:

P_r =probability that random testing will reveal at least a failure

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

P_p =probability that partition testing will reveal at least a failure=

$= 1-\prod_{i=1}^{K}(1-h_i)^{Ni}$

For a fair comparison: $N = \sum_i N_i$

Results from empirical comparisons:

D&N:

To show that random testing was almost as effective as partition testing (slightly inferior).

Considering the cost of partitioning, random testing is likely to be more cost effective

H&T:

Originated because D&N's results considered counterintuitive, similar studies, modifying some underlying assumptions and similar results.

Results from empirical comparisons:

The relative performance of the two approaches actually depends on

p_i = the probability that a randomly chosen test belongs to subdomain D_i,

where: $h = \Sigma_1^K p_i h_i$

D&N assumed uniform p_i

Assuming instead for example that p_i is lower for the D_i's with higher probability of failure, then partition testing can be shown to be "clearly superior".

Or, in other words, ...

... if we already know where the failure-causing inputs are most likely concentrated in the input domain, the best test method consists into partitioning it accordingly !!!!

Brought at the extreme consequences, we should test the only subdomain with the highest failure probability.

Failure-rate model: further analytical results

(Weyuker&Jeng, 1991)

If D_i's equal sized and failure-causing inputs equally distributed: $P_p = P_r$

P_p worst-case:

many small D_i's (in the limit of one point) and another one almost as large as the whole input domain, that includes all failure-causing inputs

P_p best-case:

there exist (at least one) subdomain(s) only containing failure-causing inputs

Failure-rate model: further analytical results

(Weyuker&Jeng, 1991)

The parameter of interest in evaluating $\frac{P_r}{P_P}$ is the *density* of the failure-causing inputs within the subdomains induced by the partitioning criterion.

Partition refinement is advantageous if the density in one of the obtained finer partitions is significantly increased.

On the contrary, once a partition only contains failure-causing inputs, any further refinement will be a waste of effort.

The Constant Confidence Rate Model

(Howden, 1993)

•Concentrate testing on the (varying as test proceeds) partition with the higher probability of failure, until all partitions reach the same failure probability.

Weak points in the failure-rate model:

Assumption of the operational input distribution

- not the right profile if our goal is finding faults

Random choice of tests within partitions:

+ avoid bias by tester's skill

- does not reflect the way input data are selected in real systematic testing

Some practical testing considerations:

•Most effective partition methods are *fault-based*: we should practice "suspicion testing".

•Structural coverage is effective *if* a failure is revealed by "many" inputs among those that cause the traversal of the item to be covered, e.g. a branch.

•Partition testing methods performs well when testing for *special cases*.

•On the contrary, partition testing is not good to "inspire confidence" after successful testing.

Conclusions:

- Systematic criteria and operational testing are based on very different assumptions, and pursue different goals
- Coverage testing should be automated and used for adequacy; better data flow-based than the simple branch coverage
- Comparison and choice of test approaches on rigorous grounds … (Strigini's talk)
- … as well as the combination of different methods (Littlewood's talk)

⇨ **In any case, for controlling and managing the test process we should routinely collect and use failure data!!!**