Chapter 5
Theory of Program Testing

J. C. Huang
Department of Computer Science
University of Houston
Theoretical discussion of three questions

1. Can we characterize an ideal test-case selection criterion?
2. Will an ideal set of test case be computable?
3. Operational vs. debug testing: which is better?
State of the art of test-case selection

- At present, there does not exist a test method that allows us to conclude, from a successful test, that a program does not contain any error.

- If such a method exists, what properties it should have?
Notations

Let

\( F \) be a program,

\( D \) be the input domain of \( F \),

\( F(d) \) denote the result of executing \( F \) with input \( d \in D \),
Notations (continued)

OUT(d, F(d))

specifies the output requirement for F (i.e., OUT(d, F(d)) is true if and only if F terminates cleanly for d and F(d) is an acceptable result),
Notations (continued)

OK(d) be an abbreviation for OUT(d, F(d)),

and

T be a subset of D.
Ideal set of test cases

Elements of $T$ are called *test cases*.

$T$ constitutes an *ideal set of test cases* if

$$(\forall t)_T(OK(t)) \supset (\forall d)_D(OK(d)).$$

If we can find an ideal set of test cases, we can conclude from a success test that the program contains no error.
Successful test

A test using $T$ is said to be *successful* if the program executes correctly with every element of $T$. Formally,

$$\text{SUCCESSFUL}(T) \equiv (\forall t)_T(\text{OK}(t)).$$
How a set of test cases is selected?

Typically, a set of test cases is a subset of $D$ selected to satisfy some test-case selection criterion, $C$, which is usually a predicate over $2^D$, the power set (i.e., the set of all subsets) of $D$. 
**Reliable selection criteria**

A test-case selection criterion, C, is said to be *reliable* if and only if programs succeed or fail consistently when executing a set of test cases satisfying C. Formally,

\[
\text{RELIABLE}(C) \equiv \\
(\forall T_1)_{2^p}(\forall T_2)_{2^p}((C(T_1) \land C(T_2)) \supset (\text{SUCCESSFUL}(T_1) \equiv \text{SUCCESSFUL}(T_2)))
\]
Valid selection criteria

A test-case selection criterion, C, is said to be valid for a particular program if and only if there exists a set of test cases satisfying C which will cause the program to fail the test if the program is incorrect. To be more precise,

\[
\text{VALID}(C) \equiv (\exists d)_D(\neg \text{OK}(d)) \supset (\exists T)_C(C(T) \land \neg \text{SUCCESSFUL}(T))
\]
On validity

Note that validity does not imply that every set of test cases selected with C will cause the program to fail the test if the program is incorrect.
Example

Suppose the program is intended to double the input, but instead, it squares the input. To express it in the formalism introduced above,

\[ F(d) = d \times d \text{ for all } d \in D \text{ and } \]

\[ \text{OK}(d) \equiv (F(d)=d+d). \]
Example (continued)

Note that OK(0) and OK(2), but ¬OK(d) for all d not in \{0, 2\}. 
Example (continued)

\[ C_1(T) \equiv (T=\{0\}) \lor (T=\{2\}) \]

(reliable but not valid)
Example (continued)

\[ C_2(T) \equiv (T=\{t\}) \wedge (t \in \{0,1,2,3,4\}) \]

(not reliable but valid)
Example (continued)

\[ C_3(T) \equiv (T=\{t\}) \land (t \in \{1,3,4\}) \]

(reliable and valid)
Example (continued)

\[ C_4(T) \equiv (T=\{t_1, t_2, t_3\}) \land (D \supseteq T) \]

(reliable and valid)
A test as a proof of correctness

A successful test constitutes a direct proof of program correctness, if it is done with a set of test cases selected by a test criterion which is both reliable and valid. Formally,

\[(\exists C)(\text{VALID}(C) \land \text{RELIABLE}(C) \land (\exists T)(C(T) \land \text{SUCCESSFUL}(T))) \Rightarrow \text{SUCCESSFUL}(D)\]
Comment

• In general, it is difficult to prove the validity and reliability of a test-case selection criterion. In some cases, however, it may become trivial.

• For example, the proof of validity becomes trivial if $C$, the test-case selection criterion, does not excludes any member of $D$, the input domain, from being selected as a test case.
Observation

• If C does not allow any member of D to be selected, then C can be valid only if the program is correct.

• Thus, in that case, it is required to prove the program's correctness in order to prove the validity of C.
Observation

• The proof of reliability becomes trivial if \( C \) requires selection of all elements in \( D \) because in that case there will only be one set of test cases.

• The proof also becomes trivial if \( C \) does not allow any element from \( D \) to be selected.
Second question

• Does there exist an effective procedure to find an ideal set of test cases?

• An ideal set of test case for a program is the one with which a successful test will constitute a direct proof of the correctness.
Howden’s result

Theorem 5.2.1: There exists no computable procedure $H$ which, given an arbitrary program $F$ with domain $D$, can be used to generate a nonempty finite set $D \supset T$ such that.

$$(\forall t)_T(\text{OK}(t)) \supset (\forall d)_D(\text{OK}(d)).$$
Third question

Operational testing vs. debug testing: which is more effective?
Debug vs. Operational Testing

Main purposes of software testing:

• to achieve adequate quality - debug testing
• to assess existing quality - operational testing
Debug Testing:

Directed at finding as many errors as possible through proper test coverage or the use of test cases most likely to reveal errors.
Operational testing:

Subject the software to the same distribution of inputs that is expected in operation (production run), and wait for failure to surface “spontaneously”.

A comparison

- Debug testing may be more effective at finding bugs, but if the bugs found are mostly unimportant, the software quality would not be appreciably improved.

- Operation testing will uncover bugs that may be found in actual operation, and thus are more important.
A comparison (continued)

• Effectiveness of debug testing depends on the correctness of assumptions made about possible errors.

• Effectiveness of operation testing depends on the extend to which the testing profile coincides with the operational profile.
How to measure effectiveness?

In most published papers on the subject, the effectiveness (of a test) is measured by the *failure-finding probability (FFP)*, the probability that the test cases will cause at least one failure during the test.
Comment on FFP

- Obviously, test-case selection criteria can be established based on the failure-finding probability of individual test case.

- Furthermore, the adequacy of a test (i.e., stopping criteria) can be decided based on the failure-finding probability of the set of test cases used.
FFP and delivered reliability

Failure-finding probability sheds little light on how the detection and elimination of failures during the testing process affects the delivered reliability.
FFP vs. delivered reliability

- Different failures may make vastly different contributions to the (un)reliability of the program.

- Thus, testing with a technique that readily detects "small" faults, may result in a less reliable program than would testing with a technique that less readily detects some "large" faults.
The debugger’s intuition

• Employing test methods that are designed to expose failures is believed to be a better alternative than simulating normal operation and letting the failures appear.

• The validity of that belief has been studied in [FHLS98].
The significance of this belief

• The validity of this belief is not merely of academic interest.

• Critical software must be tested in many different ways. Because testing is expensive and time consuming, developers and regulatory agencies would like to choose among alternatives instead of using them all.
Current practice

The prevailing practice in the software industry is to achieve high reliability by

1. Testing to discover failures
2. Debugging to locate faults
3. Removing faults with fixes
A popular position

Reliable software can best be developed using formal methods. When properly applied, these methods eliminate at source those failures normally exposed at the unit and subsystem levels by debug testing. Therefore, unit debug testing should be reduced in favor of additional system-level random testing.
Example

In the "Cleanroom" development methodology, to give an extreme example, debug testing is generally not used at all, particularly by those doing the development. Apart from its ability to provide reliability estimates, it is argued that operational testing detects any remaining failures that could occur, with probabilities that are in proportion to their seriousness.
Comment

• However, experienced developers, say, of flight-control software, are profoundly disturbed by the suggestion that they abandon debug testing.

• As an indication of the depth of traditional testers' reaction to this position, Beizer has attacked Cleanroom as "lead[ing] to false confidence."
Current status

Attempts to support or refute beliefs about debug testing have been inconclusive.
Empirical studies

Attempts to establish a correlation between the degree of debug testing and the resulting software quality are at best preliminary.

On the other hand, case studies of development using formal-methods show great variation, both in the care with which the method is defined and applied and in the results.
Analysis of partition testing

A number of theoretical studies have compared random testing with debug ("partition") testing.

Although random testing is a surprisingly good competitor for partition testing, it is seldom better, and scenarios can be constructed in which partition testing is much better at failure exposure.

Thus, the question remains.
The question to study

Under what conditions (on the program and the testing method) will debug testing deliver better reliability than operational testing?
Debug testing might be better

Certainly conditions exist favoring each alternative.

If many debug tests fail and the corresponding fixes substantially decrease the overall failure probability, then debug testing may be superior to operational testing in which fewer tests happened to fail.
Operational testing might be better

It may happen instead, however, that many fixes originated by debug testing are less effective, in terms of improving reliability in operation, than a few fixes originated by operational testing.
The case of ultra-reliability

is of particular interest. When the failure set has a very small chance of being encountered in operation, operational testing has a correspondingly very small chance of inducing failures and thus allowing the removal of failure regions. Debug testing is therefore the only option that allows some hope of further improving reliability.
The case of ultra-reliability (continued)

However, simply choosing debug testing is no guarantee that the results will be better than with operational testing. It may still happen that debug tests encounter only failure points whose probability in the operational profile is so low that fixes are worthless, or simply that it does not encounter any failure region. That is, the debug test regime chosen, or the tester's experience, may be ill-matched to the failure regions present in the software.
The case of ultra-reliability (continued)

Furthermore, even if debug testing does achieve ultra-reliability, it cannot demonstrate that ultra-reliability has been achieved; only an infeasible amount of operational testing can demonstrate that.
Debug-tester’s nightmare

Detection and removal of many minor problems, while failing to detect the serious problems.
Conclusions

[FHLS98] considers the question of whether low operational failure probability (and hence better reliability) may be better obtained by looking for failures (debug testing), or by sampling from expected usage (operational testing).
Conclusions (continued)

Debug testers always have the potential advantage that, by adjusting the test profile and subdomain definitions, they might improve the behavior of debug methods. While operational testers have no such freedom, they do have the advantage that the operational profile, and operational testing, *define* the desired result.

Studies like this one can thus be viewed as advice to the debug tester, on how to choose a test profile that will yield superior reliability.
Conclusions (continued)

If the debug tester has good intuition about which points are likely to be failure points and, moreover, about which of these failure points are likely to belong to large failure regions, such insight can be used to devise testing strategies that deliver much lower expected failure probability than operational testing.
Conclusions (continued)

If the tester lacks such intuition or is unable to map that intuition into an appropriate input distribution, then operational testing may be indicated.
Conclusions (continued)

There are obvious cases in which debug testing is superior (roughly, because its detection rates are greater than the failure probability). Similarly, operational testing can be obviously superior (roughly, because detection rates in many subdomains are smaller than the failure probability, so debug tests there are wasted).
Conclusions (continued)

Debug testers should be aware of the potential confusion between detecting failures and achieving reliability, a confusion that occurs when testing finds only unimportant failures.

"Unimportant" of course refers to the weighting of the operational profile, which may well be unknown. But there is usually some intuition about the frequency with which a problem might arise in use, and if a debug technique consistently turns up low-frequency problems, it may be counterproductive to use it.
Conclusions (continued)

The analysis of debug testing without subdomains suggests that, if limited resources are available, only debug methods that focus on the most important failure regions are appropriate.
Conclusions (continued)

The results here may be of particular relevance to those who have a responsibility for assuring ultrahigh reliability in safety-critical systems.
Conclusions (continued)

Even when an ideal debug testing strategy yields high probability of reaching a reliability target, small deviations from the ideal may perform much worse. There is thus a need to demonstrate the reliability that has actually been achieved, and debug testing is unable to do this.
Possible application

The results of this work can be used to improve the possibilities of rational decision-making because it describes effectiveness in terms of other meaningful measures. Even for decisions that are based on intuitive judgment, it can flag—and thus avoid—illogical decisions, by showing non-obvious implications of the decision maker's premises.