EXPERIMENTAL CONTROL IN SOFTWARE RELIABILITY CERTIFICATION

Carmen J. Trammell and Jesse H. Poore
Software Quality Research Laboratory
University of Tennessee

There is growing interest in software "certification," i.e., confirmation that software has performed satisfactorily under a defined certification protocol. Regulatory agencies, customers, and prospective reusers all want assurance that a defined product standard has been met.

In other industries, products are typically certified under protocols in which random samples of the product are drawn, tests characteristic of operational use are applied, analytical or statistical inferences are made, and products meeting a standard are "certified" as fit for use. A warranty statement is often issued upon satisfactory completion of a certification protocol.

The statistical principles that underlie such product protocols have long been advocated by Mills and colleagues [1,2,3,4] and Musa and colleagues [5,6,7] as the basis for software reliability certification. The terminology used by Mills and Musa differs slightly, but their ideas are similarly drawn from scientific approaches to product certification in mature engineering disciplines. The terminology of Mills will be used in this paper.

"Statistical testing" was conceived by Mills and has been advanced by his colleagues at IBM, Software Engineering Technology Inc., and the University of Tennessee. In statistical testing,

(1) expected operational use is represented in a usage model of the software,

(2) test cases are randomly generated from the usage model,

(3) test cases are executed in an environment that simulates the operational environment, and

(4) failure data are interpreted according to mathematical and statistical models.

Methods for the construction of usage models (8,9) and the interpretation of failure data (10) have been given. Usage models are developed *before* testing, and interpretation of failure data occurs *after* testing. Proper experimental control *during* testing is critical to the integrity of the protocol, however, and has not previously been addressed.

This paper outlines specific engineering practices that must be used to preserve the validity of the statistical certification testing protocol. The assumptions associated with a statistical experiment are given, and their implications for statistical testing of software are described. The ideas in this paper have evolved from experience in fifteen Cleanroom projects conducted in the Software Quality Research Laboratory at the University of Tennessee.

The Slippery Slope

It was a typical day in the testing phase of a software development project at ACME Software.

Jane had been testing for hours, and her mind was drifting. She took a break. When she returned and ran the next test case, she noticed something unexpected, but she knew this unexpected event had to have been happening all along. She realized she had been too tired to observe it when it first occurred. She didn't know when it had first shown up.

John and Mary were both running test cases. John saw a screen event and

thought it was expected behavior. Mary saw the same event and recorded it as unexpected behavior.

Joe suddenly realized that there was an error in his part of the code, and he was anxious to fix it. He waited until testers had stopped for the day, made the change, and recompiled. The testers would continue their work the next day using his new version. He knew he had made the change and recompiled properly, so there was no need to bother the test team about this.

Michael looked over the stack of test cases and saw that they varied greatly in length. He knew that they had been randomly generated, so he assumed that they were all equally usable test cases. He rifled through the stack and picked the shortest ones so he could run the most cases in the least time.

Deborah was a new hire assigned to take the place of a certification engineer who left the company abruptly. She worked with the experienced engineer for a day, and then started testing on her own. She couldn't really read the spec to check the details of correct output, so she decided to just use her best judgment and not bother the others unless she was really confused.

Bill had an extremely long test case. In the middle of the test case, the prescribed events led him back to the Main Menu. Ordinarily, a test case would end at this point, but this case called for a second major scenario. Bill decided the case was unreasonably long, and counted the second major scenario as a new test case.

These very common events are threats to the integrity of a statistical approach to software testing. Statistical software testing, as a scientific endeavor in the real world, inevitably requires some compromises in methodological purity, and it is important to understand the nature of the slippery slope. The assumptions underlying a statistical experiment must be understood, the practical threats to experimental integrity must be recognized, and a strategy for experimental control must be employed.

Software Testing as a Statistical Experiment

In statistical certification testing, software testing is viewed as a statistical experiment. A subset of all possible uses of the software is generated, and performance on the subset is used as a basis for conclusions about general operational reliability. In standard experimental parlance, a "sample" is used to draw conclusions about a "population." Figure 1 shows the parallel between a classical statistical experiment and statistical software testing. Under a testing protocol that is faithful to the principles of applied statistics, a scientifically valid statement can be made about the expected operational performance of the software based on its test performance.

The premise that must be accepted as a starting point in this analogy is that it is not possible to test *all* ways in which software may be used. This is apparently not a premise that can be assumed as obvious. In a discussion of software testing with the top software manager in a large aerospace corporation, the infeasibility of testing all possible usage scenarios was cited as the motivation for statistical testing. "But we *have* to test every possible use of the software," he said. "The kind of software we develop could cause deaths if it is not tested completely."

Software with an unbounded input sequence length has a theoretically infinite number of possible usage scenarios. For software with only two user inputs, A and B, the possible scenarios of use are A, B, AA, AB, BB, BA, AAA, AAB, ABA, BAA, and so on. Software with a bounded but large input sequence length has a finite

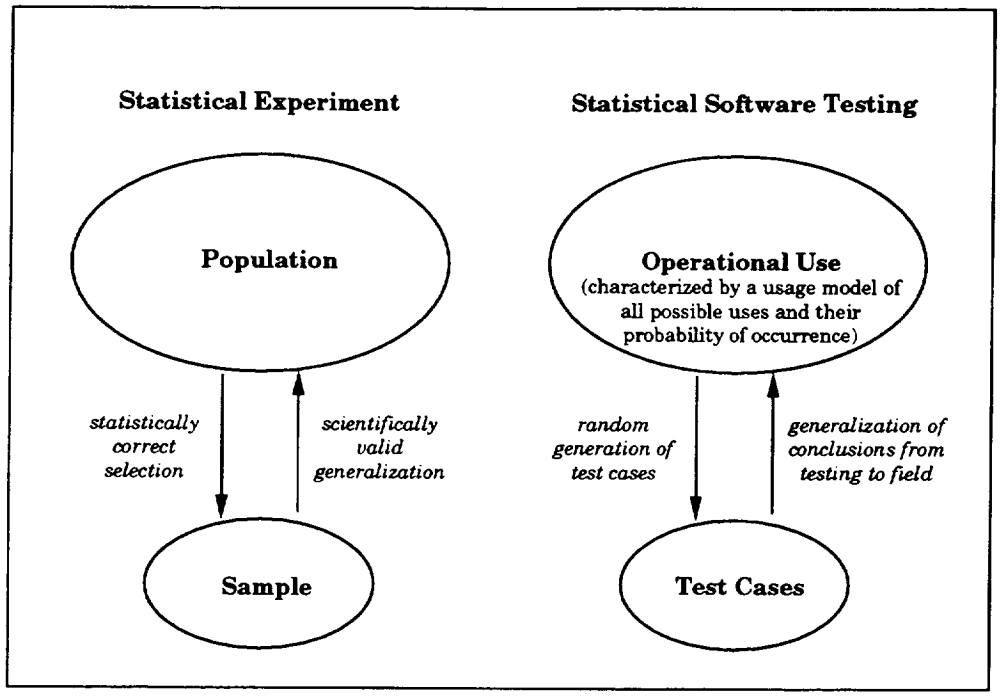


Figure 1. Software Testing as a Statistical Experiment

but astronomical number of possible usage scenarios.

The functional testing community measures test coverage in terms of function coverage. But testing every function is not the same as testing every combination of functions. And testing every combination of functions is not the same as testing every possible sequence of functions.

The structural testing community measures test coverage in terms of code coverage. But testing every line of code is not the same as testing every path. And testing every path is not the same as testing every possible sequence of paths.

There is really no question about whether all possible scenarios of use will be tested. They will not. The only questions are how the population of uses will be characterized, and how a subset of test cases will be drawn. A random sample of test cases from a properly characterized population, if applied to the software with proper experimental control, will allow scientific generalization of conclusions from testing to operational use. Any other set of test cases, no matter how thoughtfully constructed, will not.

Assumptions in a Statistical Experiment

In a statistical experiment, a well-defined procedure is performed under specified conditions, and produces one of two or more possible outcomes. Each performance of the procedure is called a "trial" of the experiment. The outcome data from successive trials of the experiment can be used to estimate the probability of each of the outcomes. Figure 2 portrays the general structure of a statistical experiment.

Several assumptions underlie the validity of inferences from a statistical

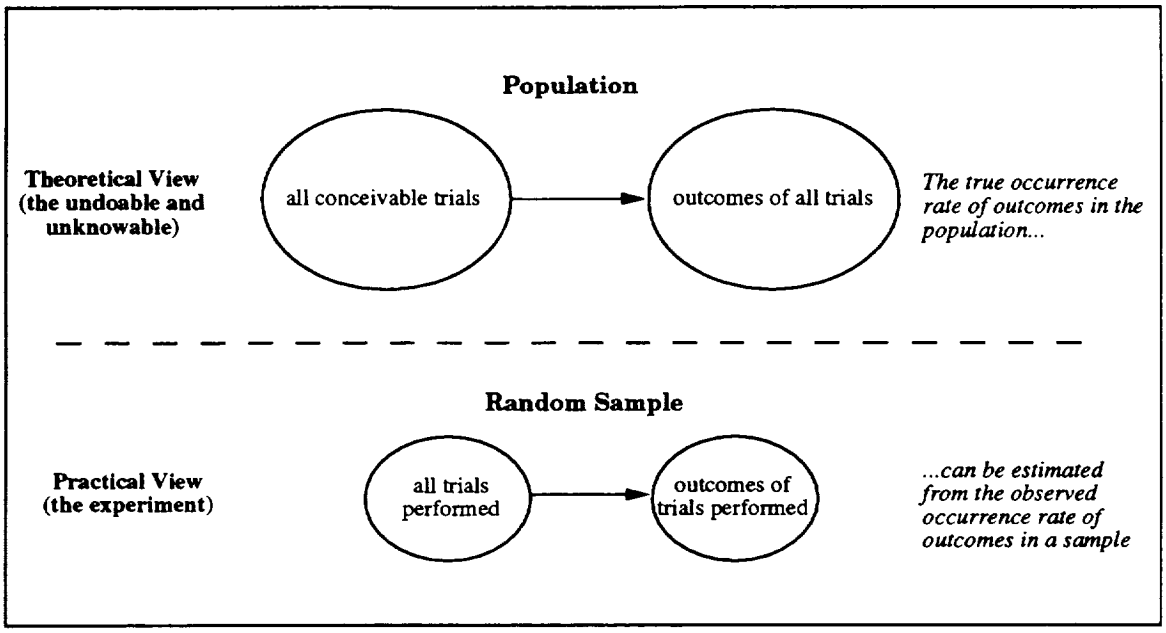


Figure 2. Structure of a Statistical Experiment

experiment, however. The assumptions are as follows.

(1) Each trial is performed under the same conditions.

(2) There is one outcome per trial.

(3) All outcomes are possible in each trial.

(4) Trials are independent.

The implications of these assumptions for the testing protocol must be understood. Proper experimental control in statistical certification testing is essential to the validity of the claims that result.

Meeting the Assumptions of a Statistical Experiment in Statistical Testing of Software

In statistical testing, a trial is ordinarily considered to be a test case. A test case generated from the usage model is a complete usage scenario beginning with some appropriate initial event (e.g., invocation, switchhook up, power on) and ending with some appropriate final event (e.g., termination, switchhook down, power off). Other definitions of a trial are possible, however, such as a single transaction or some other set of transactions. The certifier defines a trial in a manner that is appropriate for the application, and must do so in conjunction with the form of generalization the certifier wants to make about the population.

A statistical test case results in one outcome from a specified set of possible outcomes. The possible outcomes of a test case, for example, may be defined as {success, failure}. Under another design, the possible outcomes might be {success, cosmetic failure, serious nonblocking failure, blocking failure, crash}. Another design still may entail outcomes of {0 failures, 1 failure, 2 failures, ...10 or more

failures}. The challenges in experimental control grow with the complexity of the design since more granular judgments are required.

Regardless of the design of the statistical experiment---i.e., the definition of a trial and the specified set of possible outcomes---the foregoing assumptions about a statistical experiment must be met in the way trials are conducted and evaluated.

The implications of each of the foregoing assumptions is considered next. In the following discussion, a trial will be regarded as a test case that has been randomly generated from the usage model, and the possible outcomes of the test case will be regarded as success and failure.

Assumption 1: Each trial is performed under the same conditions.

What "conditions" are relevant to the conduct of a test case? The entities associated with a test case are, at a minimum,

- the software,
- the input,
- the system environment,
- the basis for evaluation of performance, and
- the tester (human or automated).

The software and the basis for evaluation of performance are entities that can be held constant; the input, the system environment and the tester are not amenable to complete control.

Software. The software used in testing will not change unless it is deliberately modified and recompiled. If it is changed in any way, *the statistical experiment must begin anew*. One may not amass data over several versions of software and treat them as a simple statistical experiment. Such data may be applied to reliability *growth* models that predict growth as a function of performance history and *changes* in the software, but may not be used to estimate parameters of a specific version of the software. Testing of each version of the software is a separate statistical experiment.

Input. To the extent that input varies with classes of usage---e.g., novice vs. expert, literary vs. mathematical subject matter, new vs. mature database---separate statistical experiments may be desirable. Otherwise, input (regardless of its origin in the system under test or another source) may be directly incorporated in the usage model structure and randomized via the usage probability distribution (e.g., percentage access of short and long files). The latter strategy effectively removes input from the set of conditions to which Assumption 1 applies by making it part of the trial rather than part of the background. This strategy also eliminates the distortion that could result from tester bias toward the shortest test cases, the "easiest" ones, the most subjectively interesting ones, etc.

System Environment. The system environment is perhaps the most illusive of the conditions to be controlled. Variability of background will be a feature of the real operational environment, however, so the experimental task is to simulate a test environment with variability that is typical of the actual environment. Concurrent activity, system load, interrupt schedules, etc., make for a constantly changing background. Again, key variables may be directly incorporated in the usage model structure and randomized via the usage probability distribution.

Basis for Evaluation. The basis for evaluation of a test case may be the specification, an independent "oracle," or both. It is not uncommon for a specification to change at any stage of

development, including testing. Consistent evaluation criteria must be applied within a testing experiment, however. Behavior that is regarded as correct (or incorrect) in one test case must be evaluated the same way in any other test case applied to that version of the software.

Tester. A given human tester may vary in the way he or she conducts and evaluates test cases, and the performance of any two testers may vary. Training, alertness, motivation, perception, and any number of other variables may affect the performance of human testers. While complete control over these factors is impossible, most of the variability can be eliminated through

- coordination of all test activities by a chief certification engineer,
- thorough tester training,
- explicit policies about test materials, session length, and data collection,
- documented guidance about issues on which the "test script" is not explicit,
- periodic "recalibration" of testers through paired performance of test cases with the chief certification engineer, and
- timely communication among testing team members with regard to observations and decisions that may affect test judgment.

Assumption 2: There is one outcome per trial.

If the specified set of outcomes (i.e., elementary events) is {success, failure}, then the outcome of a test case is either one success or one failure; it is not both, not two successes, *and not two (or more) failures.* A success is a test case in which the software performs correctly on all inputs in the test case; otherwise, the test case is counted as a failure.

In the strictest sense, then, counting of successes and failures is a simple matter. The number of successes plus the number of failures equals the number of test cases run.

The implication of one-outcome-per-trial is that a test case must be counted as a failure as soon as a failure on any input occurs. This is an unpopular policy, however, because a minor but unavoidable failure that occurs early in every test case will drive the measured reliability of the version to zero even though the software does most everything correctly.

An organization using statistical certification testing must develop a testing policy that accommodates the assumption of one-outcome-per-trial, yet allows testing to proceed in the presence of minor failures. Policy options may be politically difficult (e.g., counting every failure, with the result that status reports show declining reliability) or scientifically suspect (e.g., not counting recurrences of a failure, such that a correct fix and independence of failures must be assumed). Policies each have their advantages and disadvantages. A reasoned policy must be reached and used, however, so that the implications for the integrity of the statistical experiment are understood.

Assumption 3: All outcomes are possible in each trial.

All possible scenarios of usage must be candidates for selection in each trial, such that all the ways the software could succeed and all the ways it could fail are potentially observable.

In addition, this assumption implies that testing must not proceed in the presence of "blocking" failures. If an input is unreachable due to a blocking failure that

is "not counted" upon recurrence, then success or failure that would result from the input cannot be observed. The detection of a blocking failure is grounds for stopping the testing process and creating a new version of the software.

Assumption 4: Trials are independent.

Trials are independent if the outcome of one trial has absolutely no connection with the probability of the outcome of any subsequent trial. For software, trials (i.e., test cases) are independent if the success or failure of one test case has no bearing on the success or failure of any subsequent test case.

It may be argued that this assumption cannot be met since programs build up state information over successive runs. Since state data is the encapsulation of input history, the input in one trial may result in a change in state, and the new state may increase the probability of exposing a program defect---i.e., producing a failure---in a subsequent trial.

The only certain way to avoid dependency between failures is to fix each fault and corresponding state data after a failure, and restart testing with the new version of the software.

Alternatively, it may be possible to either avoid or randomize state data. Two types of state information exist: internal variables and external files. Internal variables exist for the duration of an execution. A test case that ends in termination, therefore, will not carry over internal state data to the next run. External files persist from one execution to the next, of course, but it is often not necessary to use them in sequential runs; their use may be randomized. Test cases of word processing software, for example, may randomly access one of a number of files (e.g., no file; short and long files; narrative and equation-filled files; etc.) according to an expected usage distribution.

Regression testing is a common violation of the assumption of independent trials. If previously used test cases are run on a new version of the software, they should not be counted as new trials.

Independence of trials in statistical software testing is defensible, but requires a deliberate strategy---either fixing failures as they are found, avoiding the carryover of state data, or randomizing state data according to an expected usage distribution.

The Slippery Slope Revisited

ACME Software improved control over its software testing process by establishing a documented testing protocol and training the project team. Things were different in the next project.

Before testing began, the testing team reviewed the specification, the test script, and other reference materials in detail. The group executed the first several test cases together, with each person taking a turn as the tester. The group reconvened at several points in testing for brief "recalibration" sessions. John and Mary's evaluations of test cases were much more consistent this time.

As prescribed by the protocol, Jane took a short break after each testing hour to review her annotations on the test script, update the chief certification engineer on her progress, and confer with other testers. She was much more alert and attentive to detail as a result.

Joe now understood that the product reliability claim would only be valid if engineering changes were made in a controlled way. Everyone understood that any deviation from the protocol was to be discussed by the team so that the impact on the integrity of the testing process could be determined.

Michael and Bill both now understood the "selection bias" that could result from picking and choosing among test cases, subdividing test cases, or otherwise altering the randomly generated sample of test cases. They now executed test cases in the order in which the test cases were generated.

Testers recorded all choices and observations as notes on the test script. Anyone with points of uncertainty---such as new hires---could later go over the specifics with the chief certification engineer to ensure the correctness of evaluations.

The Engineering Practice of Statistical Reliability Certification

If test team members are aware of the threats to experimental integrity, they can approach the innumerable decisions that must be made during testing with an eye toward preserving the validity of results. Recommendations for control over the testing process in the foregoing discussion are summarized here.

Test Preparation

- Define a test case as a usage scenario that is a longer period than the software can retain internal state data (e.g., invocation-to-termination).
- Randomize external state data via the usage probability distribution.
- Define the system environment(s), and either establish different usage models for different environments or sustain the conditions in a given environment throughout testing.
- Train test staff to ensure a common understanding of all test materials and policies, and monitor performance to prevent "drift."

Test Case Execution and Evaluation

- Hold the specification and independent oracle constant for each version of the software that is tested.
- Assign each test case one outcome from the specified set of possible outcomes.
- Run test cases in the order in which they are generated. Do not pick and choose.
- If previously used test cases are rerun on a new version, they should be performed for peace-of-mind only and not counted as new random trials.
- If a "blocking" failure occurs, stop and create a new version.
- If a failure occurs which could conceivably cause a subsequent failure, stop and create a new version.
- Schedule regular communication between test team members for discussion of matters that may affect test judgment.

Surviving the Compromises of Everyday Practice

A sound testing strategy may be compromised in practice if the rationale for the strategy is not well understood, is not embodied in a documented process, or is not practiced as documented. Indeed, "the difference between theory and practice in practice is greater than the difference between theory and practice in theory."

The threats to validity in certification testing can largely be controlled through understanding the assumptions in a statistical experiment, establishing explicit policies to meet them, and monitoring adherence to the policies in practice. Such experimental control is necessary to sound

footing on the slippery slope of applied science.

References

1. Currit, P. Allen, Michael Dyer, and Harlan D. Mills. "Certifying the Reliability of Software." *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 1, January 1986.

2. Mills, H. D., M. Dyer, and R. C. Linger. "Cleanroom Software Engineering." *IEEE Software*, September, 1987, pp. 19-24.

3. Mills, H. D. and J. H. Poore. "Bringing Software Under Statistical Quality Control." *Quality Progress*, November 1988.

4. Cobb, R. H. and H. D. Mills. "Engineering Software Under Statistical Quality Control." *IEEE Software*, November 1990.

5. Musa, J.D., A. Iannino, and K. Okumoto. Software Reliability: Measurement, Prediction, Application. McGraw-Hill: New York, 1987.

6. Musa, J.D. and William W. Everett. "Software-Reliability Engineering: Technology for the 1990s." *IEEE Software*, November 1990.

7. Musa, John D. "Operational Profiles in Software-Reliability Engineering." *IEEE Software*, March 1993.

8. Whittaker, James A. and J.H. Poore. "Markov Analysis of Software Specifications." *Transactions on Software Engineering and Methodology*, January 1993.

9. Walton, Gwendolyn H., J.H. Poore and Carmen J. Trammell. "Software Usage Modeling." *Software Practice and Experience*, to appear.

10. Poore, J. H., Harlan D. Mills, and David Mutchler. "Planning and Certifying Software System Reliability." *IEEE Software*, January 1993.

EXPERIMENTAL CONTROL IN SOFTWARE RELIABILITY CERTIFICATION

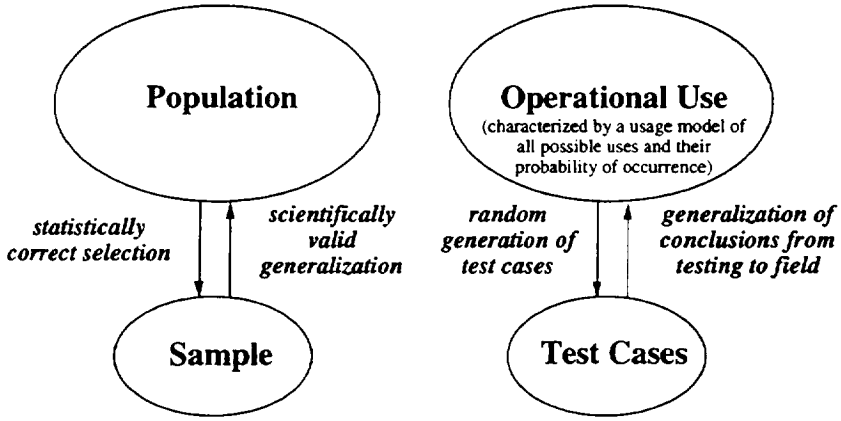
17th Annual Software Engineering Workshop
NASA/Goddard Space Flight Center

Carmen Trammell
University of Tennessee

UNIV. OF TENN. SOFTWARE ENGINEERING FOCUS: ADVANCES IN CLEANROOM PRACTICE

- Software Quality Research Laboratory
- Fifteen Cleanroom projects since 1988
- Student employees, high turnover
- Statistical testing (Mills and Musa) is used

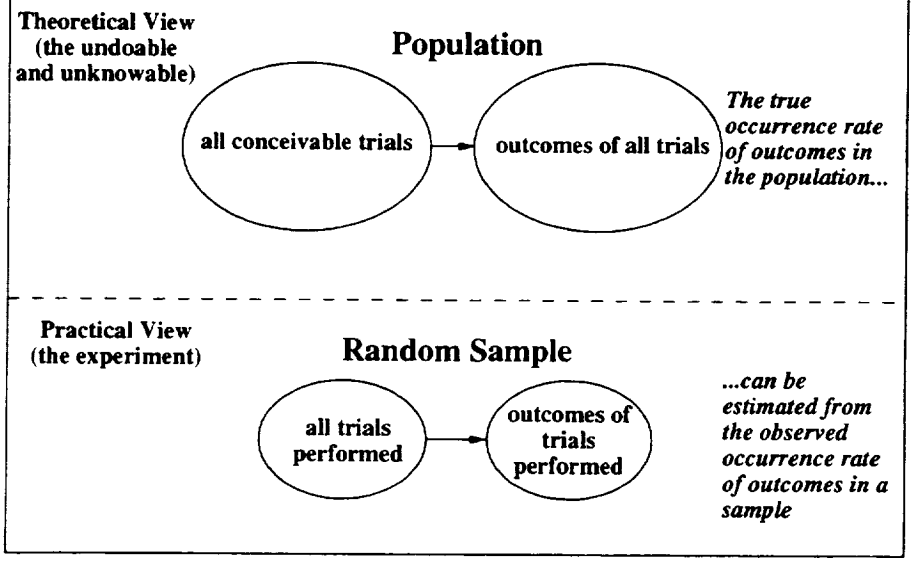
SOFTWARE TESTING AS A STATISTICAL EXPERIMENT



SYMPTOMS OF POOR TESTING PROCESS CONTROL

- **Delayed observation of failures**
- **Conflicting evaluations by testers**
- **Picking and choosing among test cases**
- **Unauthorized engineering changes**
- **Lack of communication by new testers**

STRUCTURE OF A STATISTICAL EXPERIMENT



ASSUMPTIONS IN A STATISTICAL EXPERIMENT

(1) **Each trial is performed under the same conditions.**

(2) **There is one outcome per trial.**

(3) **All outcomes are possible in each trial.**

(4) **Trials are independent.**

ASSUMPTION (1)
Each trial is performed under the same conditions.

- the software
- the input
- the system environment
- the basis for evaluation of performance
- the tester (human or automated)

ASSUMPTION (2)
There is one outcome per trial.

- the set of possible outcomes must be specified, e.g.,
  - { success, failure}
  - {no failures, minor failure, serious failure, crash}
  - {0 failures, 1 failure, ...n or more failures}
- recurrences of failures: to count or not to count?
  - counting recurrences results in reports of declining reliability
  - not counting recurrences requires judgments about independence of failures

ASSUMPTION (3)
All outcomes are possible in each trial.

- **all usage scenarios must be candidates for selection in each trial**
- **the outcome of each scenario must be observable... testing cannot proceed in the presence of blocking failures**

ASSUMPTION (4)
Trials are independent.

- **test cases must exceed the retention of internal state data**
- **external state data should be randomized**
- **regression tests must not be counted as new random trials**

LESSONS LEARNED ARE EMBODIED IN THE CURRENT PROTOCOL

Test Preparation

- Define a test case as a usage scenario that is a longer period than the software can retain internal state data (e.g., invocation-to-termination).
- Randomize external state data via the usage probability distribution.
- Define the system environment(s), and either establish different usage models for different environments or sustain the conditions in a given environment throughout testing.
- Train test staff to ensure a common understanding of all test materials and policies, and monitor performance to prevent "drift."

LESSONS LEARNED ARE EMBODIED IN THE CURRENT PROTOCOL

Test Case Execution and Evaluation

- Run test cases in the order in which they are generated.
- Hold the specification and oracle constant for each version
- Assign each test case one outcome from the set of possible outcomes.
- If test cases are rerun, do not count them as new trials.
- If a "blocking" failure occurs, stop and create a new version. If an observed failure could cause a subsequent failure, stop and create a new version.
- Schedule regular communication for discussion of matters that may affect test judgment.