Random testing is a black-box software testing technique where programs are tested by generating random, independent inputs. Results of the output are compared against software specifications to verify that the test output is pass or fail.\footnote{Random testing is a black-box software testing technique where programs are tested by generating random, independent inputs. Results of the output are compared against software specifications to verify that the test output is pass or fail. In case of absence of specifications the exceptions of the language are used which means if an exception arises during test execution then it means there is a fault in the program.}

### History of random testing

Random testing for hardware was first examined by Melvin Breuer in 1971 and initial effort to evaluate its effectiveness was done by Pratima and Vishwani Agrawal in 1975.\footnote{Random testing for hardware was first examined by Melvin Breuer in 1971 and initial effort to evaluate its effectiveness was done by Pratima and Vishwani Agrawal in 1975.}

In software, Duran and Ntafos had examined random testing in 1984.\footnote{In software, Duran and Ntafos had examined random testing in 1984.} Earlier Howden had termed it functional testing in 1980.

### Overview

Consider the following C++ function:

```cpp
int myAbs(int x) {
    if (x>0) {
        return x;
    } else {
        return x; // bug: should be '-x'
    }
}
```

Now the random tests for this function could be \{123, 36, -35, 48, 0\}. Only the value '-35' triggers the bug. If there is no reference implementation to check the result, the bug still could go unnoticed. However, an assertion could be added to check the results, like:

```cpp
void testAbs(int n) {
    for (int i=0; i<n; i++) {
        int x = getRandomInput();
        int result = myAbs(x);
        assert(result>=0);
    }
}
```
The reference implementation is sometimes available, e.g. when implementing a simple algorithm in a much more complex way for better performance. For example, to test an implementation of the Schönhage–Strassen algorithm, the standard "*" operation on integers can be used:

```c
int getRandomInput () {
  ...
}

void testFastMultiplication (int n) {
  for (int i=0; i<n; i++) {
    long x = getRandomInput ();
    long y = getRandomInput ();
    long result = fastMultiplication (x, y);
    assert(x*y==result);
  }
}
```

While this example is limited to simple types (for which a simple random generator can be used), tools targeting object-oriented languages typically explore the program to test and find generators (constructors or methods returning objects of that type) and call them using random inputs (either themselves generated the same way or generated using a pseudo-random generator if possible). Such approaches then maintain a pool of randomly generated objects and use a probability for either reusing a generated object or creating a new one.[4]

### On randomness

According to the seminal paper on random testing by D. Hamlet

> [...] the technical, mathematical meaning of "random testing" refers to an explicit lack of "system" in the choice of test data, so that there is no correlation among different tests.[1]

### Strengths and weaknesses

Random testing is typically praised for the following strengths:

- It is cheap to use: it does not need to be smart about the program under test.
- It does not have any bias: unlike manual testing, it does not overlook bugs because there is misplaced trust in some code.
- It is quick to find bug candidates: it typically takes a couple of minutes to perform a testing session.
- If software is properly specified: it finds real bugs.

The following weaknesses are typically pointed out by detractors:

- It only finds basic bugs (f.ex. null pointer dereferencing).
- It is only as precise as the specification and specifications are typically imprecise.
- It compares poorly with other techniques to find bugs (f.ex. static program analysis).
- If different inputs are randomly selected on each test run, this can create problems for continuous integration because the same tests will pass or fail randomly.[5]
- Some argue that it would be better to thoughtfully cover all relevant cases with manually constructed tests in a white-box fashion, than to rely on randomness.[5]

### Types of random testing

**With respect to the input**

- Random input sequence generation (i.e. a sequence of method calls)
Random sequence of data inputs (sometimes called stochastic testing) - f.ex. a random sequence of method calls
Random data selection from existing database

**Guided vs. unguided**

- undirected random test generation - with no heuristics to guide its search
- directed random test generation - f.ex. "feedback-directed random test generation"[6] or "adaptive random testing" [7]

**Implementations**

Some tools implementing random testing:

- QuickCheck - a famous test tool, originally developed for Haskell but ported to many other languages, that generates random sequences of API calls based on a model and verifies system properties that should hold true after each run. Check this QuviQ QuickCheck flyer for a quick overview.
- Randoop - generates sequences of methods and constructor invocations for the classes under test and creates JUnit tests from these
- Simulant - a Clojure tool that runs simulations of various agents (f.ex. users with different behavioral profiles) based on a statistical model of their behavior, recording all the actions and results into a database for later exploration and verification
- AutoTest - a tool integrated to EiffelStudio testing automatically Eiffel code with contracts based on the eponymous research prototype.[4]
- York Extensible Testing Infrastructure (YETI) - a language agnostic tool which targets various programming languages (Java, JML, CoFoJa, .NET, C, Kermeta).
- GramTest - a grammar based random testing tool written in Java, it uses BNF notation to specify input grammars.

**Critique**

Random testing has only a specialized niche in practice, mostly because an effective oracle is seldom available, but also because of difficulties with the operational profile and with generation of pseudorandom input values.[1]

An oracle is an instrument for verifying whether the outcomes match the program specification or not. An operation profile is knowledge about usage patterns of the program and thus which parts are more important.

For programming languages and platforms which have contracts (for example Eiffel, .NET or various extensions of Java like JML, CoFoJa...) contracts act as natural oracles and the approach has been applied successfully.[4] In particular, random testing finds more bugs than manual inspections or user reports (albeit different ones).[8]

**See also**

- Fuzz testing - a kind of random testing which provides invalid input to the tested program
- Lazy systematic unit testing#Systematic Testing - a systematic way of exploring "all" method calls, as implemented e.g. by NASA's Java Path Finder (which blends testing with model checking by limiting the state space to a reasonable size by various means)
- Constrained random generation in SystemVerilog
- Corner case
- Edge case
- Concolic testing
References


External links

- Random testing by Andrea Arcuri.
- Random testing by Richard Hamlet, professor emeritus at Portland State University; a valuable list of resources at the end of the paper
- Random Testing wiki at Cunningham & Cunningham, Inc.