Testing Randomized Software by Means of Statistical Hypothesis Tests

Ralph Guderlei*, Johannes Mayer,
Christoph Schneckenburger†
Institute of Applied Information Processing
Ulm University
89069 Ulm, Germany
ralph.guderlei@uni-ulm.de
johannes.mayer@uni-ulm.de
christoph.schneckenburger@uni-ulm.de

Frank Fleischer
Medical Data Services/Biostatistics
Boehringer-Ingelheim Pharma GmbH & Co. KG
88397 Biberach, Germany
frank.fleischer@boehringer-ingelheim.com

ABSTRACT

Software testing research has mostly focused on deterministic software systems so far. In reality, however, randomized software systems (i. e. software systems with random output) also play an important role, e. g. for simulation purposes. Test evaluation is a real problem in that case. In previous work, statistical hypothesis tests have already been used, but test decisions have not been interpreted. Furthermore, those tests have only been applied if theoretic values on the distribution of program outputs had been available and not in case of golden implementations. In the present paper, we propose a general approach on how to apply statistical hypothesis tests in order to test randomized software systems. We exactly determine the confidence gained through these tests. We show that after passing a statistical hypothesis test, it can be guaranteed that at least the tested characteristics of the system under test are correct with a certain probability and accuracy of the result. Our approach is also applicable in case of golden implementations. Therefore, knowledge about the outputs' distribution is not necessary in that situation, which is a great advantage. Two case studies are described that have been conducted in order to assess the proposed approach. One of the case studies is based on a software system for the simulation of stochastic geometric models (among others) that evolved from the GeoStoch research project and is now used at France Télécom R&D, Paris, in order to calculate costs for communication networks and to plan new network structures.

*Work supported by the Deutsche Forschungsgemeinschaft in the research training group "Modellierung, Analyse und Simulation in der Wirtschaftsmathematik"
†Work supported by a scholarship of the Wilken Foundation, Ulm

SOQUA '07, September 3–4, 2007, Dubrovnik, Croatia

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software / Program Verification—*Statistical methods*; D.2.5 [**Software Engineering**]: Testing and Debugging—*Testing tools*; G.3 [**Probability and Statistics**]: Reliability and life testing

General Terms

Reliability, Verification

Keywords

Test oracle, test evaluation, statistical hypothesis test, randomized software

1. INTRODUCTION

Usually, software testing is conducted with deterministic programs. In this case, the expected output can be computed in advance or just-in-time according to the specification. There are lots of approaches that can be used to evaluate test results (cf. e. g. [3]). Although the output of such programs usually is deterministic for a given input, the oracle problem [28] is far from being solved for deterministic software systems. The situation is even worse for randomized software systems. Those programs are characterized by the property that their outputs are random, i. e. for a certain input value, the output follows a certain well-defined (but mostly unknown) probability distribution. In this case, it obviously is not possible to compute one single expected output value. Thus there is an oracle problem [28]. One approach that has been employed so far is that of statistical hypothesis tests [17, 18, 26]. However, an important aspect is missing so far: It has not been considered what a passing or failing statistical hypothesis test implies for the correctness of the SUT. Due to the construction of the statistical hypothesis tests, the probability that a correct implementation is falsely rejected has been specified (in [26]). However, the much more important error probability that an incorrect implementation falsely passes has not been regarded. This error probability has been controlled in [17, 18], but in this case the other error probability has not been controlled. Thus, there is no approach yet that allows to control the error probability in each case.

The present paper proposes a general approach on how to apply statistical hypothesis tests to software testing. One

major contribution is that the error probabilities are determined for each case and thus an approach is developed with arbitrarily small error probability gained by repetition of statistical hypothesis tests. The probability of the correctness of an oracle decision can thus be explicitly stated for a given accuracy of the comparison. This has not been possible so far.

Two case studies have been conducted in order to evaluate the approach. One case study is based on an implementation of the inverse probability distribution function of the Gaussian distribution. With random input, this system can be regarded as a randomized system. This seamingly trivial study object has been chosen for two reasons: On the one hand, due to its determinism, other testing approaches are also possible and can be used to compare the fault detection effectiveness. On the other hand, since the expected output distribution is Gaussian, tests for variance are also possible. This enables us to compare the effectiveness of the variability tests with that of the variance tests.

The second case study has been conducted with a module of the GeoStoch software that is currently used by France Télécom R&D, Paris. The GeoStoch system [12, 19] has been developed at Ulm University, Germany. It implements methods from stochastic geometry, spatial statistics, and image analysis. There is an ongoing resarch project between Ulm University and France Télécom R&D, Paris that is concerned with the analysis and the modelling of network structures in the sector of telecommunication. In particular, urban street systems but also telecommunication networks on a nationwide scale are investigated where the focus is on the random geometrical structure as well as on cost characteristics like shortest path lengths on these structures (cf. [7, 8, 9]). Thereby, an efficient cost and risk analysis is enabled that can provide helpful tools for France Télécom with respect to the calculation of connection costs in already existing networks or the planning of new network systems. Naturally, due to the large amount of money and effort involved in the operation and planning of telecommunication network systems, it is a key necessity to check whether the implementations in GeoStoch provide correct results, for example with respect to the simulation of basic random geometrical structures involved in the models.

The present paper is organized as follows: Section 2 discusses relevant related work with respect to testing randomized resp. non-deterministic software systems. Section 3 describes how to model randomized software systems by means of random variables. After an introduction to statistical hypothesis tests in Section 4, asymptotic hypothesis tests that are useful to test software systems are presented in Section 5 together with the estimation of error probabilities and the description of the repetition of tests. The two case studies are described in Section 6. After a discussion of the methods and results in Section 7 followed by possible threats to validity, the paper is concluded in Section 8.

2. RELATED WORK

Statistical hypothesis tests have been used in [26] in order to test a part of a randomized system, namely UrbanSim. However these tests require that the program outputs follow a Gaussian resp. Poisson distribution. Therefore, those tests cannot be applied in the more usual case of an unknown distribution. Nevertheless, having Gaussian or Poisson distributed outputs and knowing the expected value of a correct implementation, this approach allows to test the (possibly multi-dimensional) mean of the program output against the given expected value. An upper bound for the probability of falsely rejecting a correct implementation is given. However, the probability of the much more fatal error of not detecting a bug is not assessed theoretically. Instead some empirical analysis is presented with varying number of faults inserted into the program under test. The number of failing tests is analyzed for the buggy programs. Another problem of the approach is that in case of the theoretical distribution of the program output being unknown, the expected value is tested assuming both Poisson and Gaussian distribution at the same time. Hence if a test fails, it is not known whether there is a bug in the program under test or whether the wrong distribution has been assumed (or whether the failure occured due to the given confidence of the test—a general problem of any statistical test). Finally, the repetition of tests is described in the general guideline presented on p. 222 of [26], but the evaluation of the outcome is rather subjective: "If the frequency of failing is significantly higher then α, ..."—and it is not specified, what is meant by "significantly" here.

Asymptotic hypothesis tests have been applied in [17, 18] in order to test image processing and analysis software. There, the random variables modeling the (transformed) program output have been compared with respect to their mean values. In order to control the error probability for falsely not rejecting a faulty implementation, the hypotheses have been swapped such that the null hypothesis has stated that the implementation is not correct. Thus, the type I error is the more severe error, i. e. a faulty implementation passes. However, in this case, the other error probability has not been controlled. Furthermore, besides this quite high error probability, it has not been mentioned how to increase confidence by repetitions and how to apply this approach in case of theoretical values being unavailable, but "only" a golden implementation, i. e. an alternative/legacy implementation.

The approach described in [17, 18] has been applied in the context of credit risk in order to test a simulation software [4]. The effectiveness of the tests have been assessed by means of seeded faults. This study has shown that the approach is quite useful to test randomized software that is otherwise difficult to test.

A testing method called Monte Carlo testing [6, 21, 22] is based on the same theoretical model as our approach (cf. Section 3). Roughly speaking, Monte Carlo testing is testing based on Monte Carlo simulation. The key element is a 0/1-valued decision function t which represents the test result for a single test run. In theory, the expected value $\mathbb{E}(t)$ indicates whether the program passes the test. As $\mathbb{E}(t)$ is unknown, the tests are executed multiple times and $\mathbb{E}(t)$ is estimated using the empirical mean $\overline{t_n}$. Only theoretical aspects of the approach are examined in the cited papers. Therefore, we propose a concrete testing algorithm, assess the confidence gained by the outcome of a test, and evaluate the effectiveness of our approach in two empirical studies.

Another approach to deal with randomized software is discussed in [2]. Bible et al. propose to make randomized software deterministic, e. g. fixing the seed in case of pseudo-random number generators. This approach works for regression testing, but only if the program is not refactored (e. g. the algorithm changed for performance reasons). This ora-

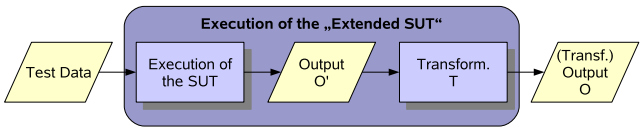

Figure 1: General test setup with transformation of outputs

cle only works for exactly the same program and not for a semantically equivalent program that e. g. only acquires the random numbers in another order. If no previous version of the program is available, it will not at all be trivial to give an oracle for such a deterministic program. Thus, using a golden implementation is only possible if the golden implementation is nearly the same. Therefore, the Gold Standard Oracle [3] cannot be used in most cases.

A completely different approach to the testing of non-deterministic systems is based on non-deterministic finite state machines (FSM) [10, 23]. In this approach, the specification of the system is modeled as a non-deterministic FSM. The non-determinism in the system is often caused by the allowance of multiple return values of the system for some input values, a typical example are network protocols. Then the IUT is tested against the FSM, usually by checking whether the IUT reaches the same state as indicated by the FSM for a given sequence of input data. Several methods describe the generation of test input data from the specification FSM. This FSM based approach has some limitations. Complex systems are hard to model as a FSM and algorithms based on FSMs often suffer from state space explosion, which impedes the practical use.

3. MODELING RANDOMIZED SOFTWARE SYSTEMS

For the testing approach proposed in the present paper, the model introduced by Kozen [14] is an appropriate choice and it is the basis of our model. The output of a program P for an input i is a random variable O_i. For generality, we can furthermore assume that the output has been transformed by a (measurable) function $T(\cdot)$ in order to compute a characteristic of the SUT which is to be tested. Although $O_i = T(O_i')$ may be a transformed version of the original output O_i' (cf. Figure 1), we'll simply talk about the output O_i in the following. Furthermore, we can model the output of a deterministic program whose input is random in the same way by a random variable O_i, where the input i introduces the randomness to O_i. For the testing of such a program against a specification S, it is useful to model the specification also as a family of random variables, one for each possible input. Then, the decision whether the program P implements its specification S can be reduced to the decision whether two random variables, namely S_i and O_i, are "equal" for each input i or not. Two random variables can be defined as "equal" if the two random variables have the same distribution function. A necessary (but not sufficient) condition for this equality is that their mean values and their variances (if existing) are equal.

For the test of a program, this means that the SUT passes the test if the expected value and the variance of the program output is equal to the theoretic expected value and the theoretic variance obtained by the specification. As the program output is random, statistical hypothesis tests have to be used to decide whether the two expected values and the two variances are equal (with a certain probability).

4. BASICS OF STATISTICAL HYPOTHESIS TESTS

The present paper applies statistical hypothesis tests in a software testing context. Thus, only those aspects of statistical hypothesis test theory are summarized in the present paper which are indispensable for the paper's purpose. Further insight into statistical hypothesis tests and all related theory is provided e. g. by [5].

Before applying statistical hypothesis tests to the respective random variables, two hypotheses H_0 and H_1 have to be stated. H_0 is called the null hypothesis and H_1 is called the alternative hypothesis. In principle, statistical hypothesis tests can be conducted for all characteristics of a given distribution (as the expected value or the variance) or for the distribution itself. The general aim of a statistical hypothesis test is to decide whether a hypothesis (the null hypothesis) has to be rejected or not. Since all statistical hypothesis tests are based on randomness, they may lead to an incorrect decision. There are two kinds of errors possible. On the one hand, the null hypothesis may be rejected although it is correct. This is denoted the so-called type I error or the α-error. On the other hand, the null hypothesis although incorrect could not be rejected (denoted as the type II error or the β-error). One major advantage of statistical tests is the possibility to specify the error probabilities, i. e. having at least some criteria about the significance of the decision.

The presented statistical hypothesis tests are based on the assumption that the random variables are (statistically) independent and identically distributed (iid). Identical distribution of the outputs means in our context that the software under test should have no state. This at least applies to the large class of software components which have this property [27]. In case of being not sure whether the random variables are (statistically) independent, a test for independence (e. g. Fisher's exact test) provides more confidence.

Normally, statistical hypothesis tests are constructed based on the knowledge of the (theoretical) distribution of the program output. However, this distribution is usually unknown. Asymptotic hypothesis tests are a promising approach to test at least some characteristics of the distribution. In most cases, the idea is based on the Central Limit Theorem and is independent of the underlying distribution. Therefore, our analysis will concentrate on this attempt. Section 5.2.1 describes how prior knowledge (i. e. the expected value and the variance) can be used in order to apply asymptotic hypothesis tests. The case of a given gold standard implementation is described in Sections 5.2.2 and 5.2.3.

Another possible class of tests are non-parametric statistical hypothesis. One of them is the so-called Wilcoxon test [29] which tests two samples for their equality. However, since the required assumptions are weaker, the test provides in general weaker results. Nevertheless, an empirical study concerning non-parametric statistical hypothesis tests would probably be worthwhile as well since this kind of statistical hypothesis tests can be applied very generally.

5. ASYMPTOTIC STATISTICAL HYPOTHESIS TESTS

5.1 Asymptotic Normal Distribution of the Empirical Mean

In case of unknown distribution, the Central Limit Theorem for independent and identically distributed random variables applied to the program outputs $X_1, ..., X_n$ says that the empirical mean $\overline{X_n} = 1/n \sum_{i=1}^{n} X_i$ centered by the true mean μ and scaled by the (true) standard deviation σ divided by $\sqrt{n}$ asymptotically follows a standard normal distribution. Experience has shown that for a sample size of $n > 30$ this asymptotic distribution is already sufficiently well approximated. Thus, at least the expected value of a program's output can be tested statistically. However, it is in general difficult to test variances directly by using an asymptotic distribution. Although $S_n^2 = \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \overline{X_n})$ is an unbiased estimator for the variance that is asymptotically χ^2-distributed, the convergence towards a χ^2-distributed random variable is too slow for practical use. Thus, in the case of unknown distribution we opted for testing another measure of variability (see Section 5.2.3).

5.2 The Approach of Asymptotic Statistical Hypothesis Tests

Basically, there are two possibilities of how asymptotic statistical hypothesis tests of an expected value can be applied. If a (theoretic) expected value is available as prior knowledge, the output data mean can be statistically tested against the given expected value (Section 5.2.1). The second possibility is to statistically test the (empirical) output data mean against the (empirical) output data mean of a gold standard implementation (Section 5.2.2). In case of variability there are the same possibilities in principle. However, it seems to be rather unrealistic to have a given theoretic variability. Therefore, Section 5.2.3 only discusses the case of testing variability against a gold standard implementation.

5.2.1 Testing against a Theoretic Expected Value

Assume a (theoretic) expected value μ_0 is already known for the program outputs. Then a necessary condition for the correctness of the program under test is, that the unknown expected value $\mathbb{E}X_i$ of the random sample should be equal to μ_0. Therefore, a hypothesis test with the null hypothesis $H_0 : \mu = \mu_0$ and the alternative hypothesis $H_1 : \mu \neq \mu_0$ (i. e. $\mu = \mu_0 + \delta, \delta \neq 0$) has to be conducted. Since in general the variance resp. the standard deviation is unknown, the Central Limit Theorem cannot be applied as described before. However, if σ^2 is estimated by S_n^2, the test statistic is no longer standard normally distributed but follows a Student's t-distribution with $n-1$ degrees of freedom. For real program output data $x_1, ..., x_n$ with arithmetic mean $\overline{x_n}$ and the empirical standard deviation s_n, the null hypothesis will be rejected if

$$\left| \frac{\overline{x_n} - \mu_0}{s_n / \sqrt{n}} \right| > t_{n-1,1-\alpha/2},$$

where $t_{n-1,1-\alpha/2}$ denotes the $(1-\alpha/2)$-quantile of the Student's t-distribution (with $n-1$ degrees of freedom) and $\alpha \in (0,1)$ is to be chosen according to the desired probability of a type I error of the statistical hypothesis test.

5.2.2 Testing of Expected Value using a Gold Standard Implementation

If the (theoretic) expected value of the correct implementation is not available but a gold standard implementation of the program being tested, the approach is quite similar. In this case the hypothesis $H_0 : \mu_1 = \mu_2$ will be statistically tested against $H_1 : \mu_1 \neq \mu_2$. Again $\mathbb{E}X_i = \mu_1$ denotes the expected value of the output of the implementation under test whereas $\mathbb{E}Y_i = \mu_2$ denotes the expected value of the output random variables of the golden implementation. For simplicity we assume that both samples have the same size. Otherwise, the approach can easily be extended. As before, for generality it is assumed that the standard deviation is unknown. However, in this case we furthermore assume that both theoretic standard deviations are equal and that the random variables X and Y are independent, i. e. the sample (X_i, Y_i) is unconnected for any $i \in \{1, ..., n\}$. For $\overline{x_n}$ and $\overline{y_n}$ describing the arithmetic means of the respective output data samples, the joint empirical standard deviation is here denoted as $s_{xy,n} = \sqrt{(s_{x,n}^2 + s_{y,n}^2)}$, where $s_{x,n}$ and $s_{y,n}$ are the respective single empirical standard deviations computed as before. Thus, the null hypothesis will be rejected if

$$\left| \frac{\overline{x_n} - \overline{y_n}}{s_{xy,n} / \sqrt{n}} \right| > t_{2(n-1),1-\alpha/2}.$$

5.2.3 Testing of Variability using a Gold Standard Implementation

In the previous sections we have statistically tested whether two expected values are the same (in a statistical sense). As mentioned before it turns out that this approach is not applicable for testing the equality of variances (if the distribution of the random variables describing the program's output is unidentified). Therefore a similar measure of variability defined as $Vb(X) := \mathbb{E}|X - \mathbb{E}X|$ is statistically tested as mostly done in respective literature. Such defined, it is possible to define random variables $X_i' := |X_i - \mathbb{E}X_i|$ and $Y_i' := |Y_i - \mathbb{E}Y_i|$. The random sample means $\overline{X_n'}$ and $\overline{Y_n'}$ and the empirical standard deviation $S_{xy,n}'$ can be determined as before. The distribution can be derived from the previous formulae. For real data, the null hypothesis is rejected if

$$\left| \frac{\overline{x_n'} - \overline{y_n'}}{s_{xy,n}' / \sqrt{n}} \right| > t_{2(n-1),1-\alpha/2}.$$

5.3 Estimating the Error Probabilities

In the previous section it has been discussed how statistical hypothesis tests can be conducted without knowing the distribution of the program outputs. However, statistical hypothesis tests without any knowledge about the error probabilities of the test decision rule do not make any sense. Therefore, the probability of both error types is estimated in the following. Again, we first have a look at testing using prior knowledge.

5.3.1 Error Estimation for Statistical Hypothesis Tests using Prior Knowledge

In case of H_0, i. e. $\mu = \mu_0$ there may emerge the error that the null hypothesis will be rejected erroneously (α-error). However, by construction of the decision rule the probability of this error will always be less than or equal to α.

In case of H_1, i. e. $\mu = \mu_0 + \delta$, $\delta \neq 0$, the error estimation is more complicated since the probability of a type II error also depends on δ. The dependence of β on δ can easily be explained. For μ_0 very close to the expected mean μ the probability of not rejecting the incorrect hypothesis $H_0 : \mu = \mu_0$ obviously is much greater than for μ_0 much farther from μ.

Hence we may assume for the following computations that $|\delta|$ is small. The probability of the β-error can be estimated as follows:

$$\begin{aligned}\beta &= \mathbb{P}\left(\left|\frac{\overline{X_n}-\mu_0}{S_n/\sqrt{n}}\right| \leq t_{n-1,1-\alpha/2}\right)\\ &= \mathbb{P}\left(t_{n-1,\alpha/2} \leq \frac{\overline{X_n}-\mu}{S_n/\sqrt{n}} + \frac{\delta\sqrt{n}}{S_n} \leq t_{n-1,1-\alpha/2}\right)\\ &= \mathbb{P}\left(t_{n-1,\alpha/2} - \frac{\delta\sqrt{n}}{S_n} \leq \frac{\overline{X_n}-\mu}{S_n/\sqrt{n}} \leq t_{n-1,1-\alpha/2} - \frac{\delta\sqrt{n}}{S_n}\right)\\ &= T_{n-1}\left(t_{n-1,1-\alpha/2} - \frac{\delta\sqrt{n}}{S_n}\right) - T_{n-1}\left(t_{n-1,\alpha/2} - \frac{\delta\sqrt{n}}{S_n}\right)\\ &\leq T_{n-1}\left(t_{n-1,1-\alpha/2} - \frac{\delta\sqrt{n}}{S_n}\right)\end{aligned}$$

The last inequality describes how β can be estimated for each δ using the Student's t-distribution function T_{n-1} with $n-1$ degrees of freedom. But since for practical testing it is much more appropriate to compute δ for a given β-error, some transformation of the latter formula is to be done. For the last term being less than or equal to a fixed $\tilde{\beta}$, δ can be computed such that the probability of a type II error is less than or equal to $\tilde{\beta}$ for all δ with $|\delta| \geq \delta_{\tilde{\beta}}$. With respect to the respective quantiles this inequality implies that:

$$t_{n-1,\tilde{\beta}} \geq t_{n-1,1-\alpha/2} - \frac{\delta\sqrt{n}}{S_n} \Leftrightarrow \delta \geq \frac{S_n}{\sqrt{n}}(t_{n-1,1-\alpha/2} - t_{n-1,\tilde{\beta}})$$

This means that for a given $\tilde{\beta}$ the probability of a type II error (β-error) is less than or equal to $\tilde{\beta}$ if $|\delta| \geq \delta_{\tilde{\beta}} = \frac{S_n}{\sqrt{n}}(t_{n-1,1-\alpha/2} - t_{n-1,\tilde{\beta}})$. Thus the error probability is in every case less than or equal to $\max(\alpha, \tilde{\beta})$, no matter whether H_0 is rejected or not.

5.3.2 Error Estimation for Statistical Hypothesis Tests using a Gold Standard Implementation

For testing of expected value using a golden implementation the error estimation is similar. Again in case of H_0, i. e. $\mu_1 = \mu_2$, the probability of a type I error denotes α by construction of the decision rule.

In case of H_1, i. e. $\mu_2 = \mu_1 + \delta$, $\delta \neq 0$, $\overline{X_n}$ can be separated into $\overline{X_n} = \overline{Z_n} - \delta$, where $\overline{Z_n}$ follows the same distribution as $\overline{Y_n}$. The probability of a type II error can be estimated using the same approach as before and is less than or equal to $\tilde{\beta}$ for δ with $|\delta| \geq \delta_{\tilde{\beta}} = \frac{S_{xy,n}}{\sqrt{n}}(t_{2(n-1),1-\alpha/2} - t_{2(n-1),\tilde{\beta}})$.

The respective errors for asymptotic hypothesis tests of variability against a golden implementation are calculated following the same approach where X_i is replaced by X_i' and Y_i by Y_i'.

5.4 Towards Correct Decision

So far, the construction of the oracle implies that the decision of the oracle is correct at least with probability $1 - \max(\alpha, \tilde{\beta})$ no matter whether H_0 is rejected or not (if $\delta = 0$ or $|\delta| \leq \delta_{\tilde{\beta}}$). Thus, in order to obtain a powerful oracle, it seems to be a good choice to choose α and $\tilde{\beta}$ very small. However, in case of $\tilde{\beta}$ being small, $\delta_{\tilde{\beta}}$ is quite large. Consequently, in order to have a tight decision rule, $\tilde{\beta}$ cannot be chosen too small accepting the disadvantage that the error probability is distinctly too high for practical application.

It is important to notice that our approach describes a typical randomized algorithm for a problem that belongs to the (randomized) complexity class **BPP** [25]. Therefore, probability amplification can be applied to reduce both error probabilities arbitrarily. The oracle is repeated R times—also repeating the test runs (with new random inputs in case of a deterministic SUT)—and the majority of outputs (only "pass" and "no pass" are possible) determines the final output. Hence, it remains to compute the number R of reruns necessary for a given error probability. Let R be an odd number, $Y_1, \ldots, Y_R$ be *iid* random variables that take the values 0 and 1 (the value 0 can be interpreted as an incorrect decision and 1 as a correct one) with probabilities $1-p$ and p, respectively, and let $S := \sum_{i=1}^{M} Y_i$. Then

$$\mathbb{P}(S \leq (1-\nu)Rp) \leq e^{-\frac{\nu^2 Rp}{2}},$$

which is called the *Chernoff Bound* [25], holds for each $\nu \in [0,1]$. Suppose $\alpha = \tilde{\beta} = 1/3$ is chosen. The probability of an incorrect decision after R reruns (i. e. $\mathbb{P}(S \leq R/2)$, since R is odd) can be bounded as follows:

$$\mathbb{P}(S \leq R/2) \quad = \quad \mathbb{P}\left(S \leq \left(1 - \frac{1}{4}\right)R\frac{2}{3}\right) \leq e^{-\frac{R}{48}},$$

where $\nu = 1/4$ has been chosen. Thus the error probability decreases exponentially with the number R of reruns. For example, in the case of 501 reruns, the probability of a wrong decision is less than $3 \cdot 10^{-5}$. If an error probability of at most q should be achieved, R has to be chosen such that $e^{-\frac{R}{48}} \leq q$ holds, i. e. $R \geq -48 \ln q$ resp. $R \geq 2\lceil -24 \ln q \rceil + 1$, since R must be odd.

5.5 Our Proposed Approach

We now will briefly describe how the methods presented can be used in a testing approach. Choose the sample size $n > 30$ and the number of reruns $R \geq 2\lceil -24 \ln q \rceil + 1$, where q is the desired overall error probability of the algorithm. The error probabilities α and $\tilde{\beta}$ are chosen $1/3$ as mentioned before. Then, the approach based on prior knowledge is as follows:

1. The following algorithm must be repeated R times:
   (a) Execute the program under test n times and compute the sample mean $\overline{x_n}$ and the sample variance s_n^2.
   (b) If the inequality in Section 5.2.1 holds, increase the counter of rejections by one.
   (c) Compute δ as described in Section 5.3.1.
2. If there are at least $R/2$ rejections, the test fails. Otherwise, it passes.

This allows us to compute the maximum δ_{max} over all δ's (for each repetition). Thus we can say that the approach correctly makes a pass/fail decision for the SUT with respect to the tested characteristics (expected value and variability) of the possibly transformed output with probability $1-q$ (up to the accuracy δ). Implementations whose output only differs by less than δ_{max} (on average) are allowed to be classified as either correct or incorrect (with respect to the tested characteristics). It is straightforward to apply the above approach also in case of a golden implementation (and also to testing the equality of variabilities).

6. EMPIRICAL STUDIES

Two empirical studies have been conducted in order to assess the proposed approach. The first study is designed in order to compare our approach to other testing approaches. The second study investigates a part of a large software system that is used at France Télécom R&D, Paris.

6.1 Empirical Study 1: Apache Commons Math

The subject of the first study is an implementation of the inverse cumulative distribution function (cdf) Φ^{-1} of the standard normal distribution taken from the Apache Commons Math library [1]. The implementation under test (IUT) is written in Java and consists of about 800 LOC (including all invoked methods). The following well-known implication is used for the statistical hypothesis test:

$$X \sim \mathcal{U}(0,1) \Rightarrow \Phi^{-1}(X) \sim \mathcal{N}(0,1)$$

That means that if the random variable X is uniformly distributed on the interval $(0,1)$, $\Phi^{-1}(X)$ is a standard normally distributed random variable.

As the inverse cdf is a deterministic function, the proposed statistical testing methods are not necessary for testing the implementation. However, this enables us to compare our approach with other testing methods. Furthermore, the fact that the output of the IUT is normally distributed makes it possible to compare the test for the equality of variabilities against a test for the equality of variances (cf. [5]).

We have chosen the number R of reruns as 501 which implies an error probability of less than $3 \cdot 10^{-5}$. All other parameters are chosen as specified in Section 5.5. The sample size n is varied in order to test with several accuracies δ (determined mainly through n, cf. Section 5.3.1).

Mutation analysis is used to measure the effectiveness of the proposed method. For details on mutation analysis refer to [13, 24]. As the implementation is written in Java, the tool muJava [15] has been used in order to generate 1280 mutants. A smoke test revealed that 39 mutants do not terminate within 20 s and 349 mutants terminate by throwing an exception.

As the implementation is a deterministic function, the original implementation can be used as a golden implementation and thus test results can be decided without error. Two different sampling techniques are used to generate the input. On the one hand, an evenly spaced lattice on the interval $[0,1)$ has been chosen and on the other hand random testing has been conducted, i. e. the input was sampled uniformly distributed on $(0,1)$. Thereafter, the output of the mutants has been compared to the output of the original implementation. For each approach 1000, test inputs are generated and passed to each mutant and to the original implementation. Using the lattice-based sampling, 208 mutants have been killed; 204 mutants have been killed with the use of random testing. All of the mutants killed by random testing are also killed by the lattice sampling approach.

In the first part of the present study, the statistical hypothesis tests are based on theoretical mean and variance known for the outputs. Table 1 shows the number of killed mutants for various sample sizes n and consequently various accuracies δ_{max} (which is also the maximum over all δ_{max} for mean, variance, and their combination). The second and the third column contain the number of mutants killed by the statistical hypothesis test for true mean and variance, respectively. The fourth column indicates how many mutants have been killed by the combination of both tests. A mutant is said to be killed by a combination if it is killed by at least one method. Note that the results of the tests are independent even if they are obtained using the same inputs.

Table 1: Results of the statistical hypothesis tests against theoretical values for Study 1

n	Mean	Variance	Mean & Variance	δ_{max}
50	74	106	148	0.199
100	85	114	159	0.140
200	95	139	186	0.099
300	97	141	189	0.081
500	107	148	194	0.063
1000	111	150	194	0.044
1500	111	150	194	0.036

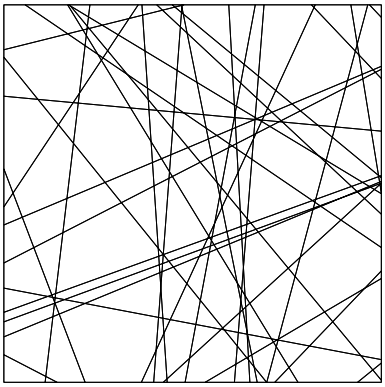

Figure 2: Sample realization of a PLT

In addition to the statistical hypothesis tests for the theoretical values, statistical tests using a golden implementation have also been conducted. Samples generated by mutants and others generated by the original implementation have been compared. The results are depicted in Table 2. The table shows the number of mutants killed by the statistical hypothesis tests for the mean, variances, linear variability, and combinations thereof.

6.2 Empirical Study 2: GeoStoch

In the second part of the study, a more complex implementation is examined. It is a part of the GeoStoch library [12] developed at the Institute of Stochastics and the Institute of Applied Information Processing at Ulm University. This software system is used at France Télécom R&D for the modelling and cost calculation of their telecommunication infrastructure. The subject of the study is an implementation of a simulation algorithm to generate so-called tessellations [16], i. e. mosaics, in our case so-called Poisson line tessellations (PLT), a sample realization is shown in Figure 2.

A tessellation is a sequence of disjoint (w. r. t. their interiors) bounded polygons that completely cover the whole plane (or a sampling window). The PLT is a random mosaic in the sense that the lines are randomly chosen within the plane.

Table 2: Results of the statistical hypothesis tests against a gold standard implementation for Study 1

n	Mean	Variance	Variability	Mean & Variance	Mean & Variability	Variance & Variability	All	δ_{max}
50	50	94	93	137	136	102	145	0.274
100	67	108	146	150	149	151	153	0.205
200	83	119	170	164	172	175	177	0.140
300	91	140	178	189	187	183	190	0.112
500	95	145	189	193	194	189	194	0.086
1000	105	147	192	194	194	192	194	0.063
1500	105	149	192	194	194	192	194	0.052

In contrast to the first study, the implementation is not a deterministic function on random input. The randomness is introduced by the program itself. Therefore, deterministic testing cannot be applied. The program has an input parameter γ that describes the mean total length of edges per unit area in the Poisson-line tessellation model. The output of the program is a spatial structure, so it is challenging to verify the output. To simplify the evaluation of the program outputs, a set of four characteristics $\lambda_1, \ldots, \lambda_4$ is computed from the output and then, these characteristics are tested for expected value resp. variability. This approach is also known as Heuristic Oracle [11]. The theoretical values for $\lambda_1, \ldots, \lambda_4$ are known in explicit form [16, 20] and do only depend on γ:

$$\begin{aligned}
\lambda_1 &= \frac{1}{\pi}\gamma^2 \text{ (mean number of vertices per unit area)} \\
\lambda_2 &= \frac{2}{\pi}\gamma^2 \text{ (mean number of edges per unit area)} \\
\lambda_3 &= \frac{1}{\pi}\gamma^2 \text{ (mean number of cells per unit area)} \\
\lambda_4 &= \gamma \text{ (mean total length of edges per unit area)}
\end{aligned}$$

The GeoStoch library is written in Java and the code examined in this study consists of 1200 LOC. Based on the IUT, 2362 mutants have been generated using MuJava. 1926 mutants could be compiled. In the study, $\gamma = 0.05$ has been chosen and the number of reruns has been fixed to $R = 501$.

A simple smoke test revealed that 36 mutants do not terminate within 30 s of execution time and 570 mutants terminate by throwing an exception. The asymptotic statistical hypothesis tests are based on the formulae for the characteristics. The first four tests have to decide whether the mean of the computed characteristics are equal to their theoretical values, the next four tests check the mean values of the characteristics computed against the mean values of the characteristics computed from the output of the original implementation, and the last four tests compare their linear variabilities.

The results of the tests of the mean characteristics against the theoretical values are shown in Table 3. The first column shows the sample size, in the next five columns, the number of mutants killed by the tests w. r. t. to the single characteristics and their combination are presented. In the last column, the accuracy δ_{max} is given. Here, δ_{max} is the maximum of all δ's for all characteristics and mutants.

The results for the tests against a golden implementation are presented in Table 4. Here, only the combined results over all λ_i are shown. In the second column, the combination of the numbers of mutants killed by the tests for equal means for each of the four characteristics is presented. In the third column, the combination of the number of mutants killed by the tests for equal linear variabilities is given. The combination of these results is presented in the fourth column. As before, δ_{max} in shown in the last column.

Table 3: Results of the statistical hypothesis tests against theoretical values for Study 2

n	λ_1	λ_2	λ_3	λ_4	All	δ_{max}
100	56	58	56	56	64	0.00506
200	58	60	60	58	63	0.00387
300	59	61	61	61	66	0.00328
500	60	62	62	60	66	0.00250
1000	61	62	62	61	66	0.00180

Table 4: Results of the statistical hypothesis tests against a gold standard implementation for Study 2

n	Mean	Variability	All	δ_{max}
100	58	57	58	0.00766
200	55	56	57	0.00537
300	58	65	65	0.00451
500	60	63	66	0.00347
1000	60	63	66	0.00251

In addition to the statistical tests, deterministic tests have been obtained by fixing the seed of the random number generator. In our case, the random number generator is a separate class which can be influenced without modifying the IUT and since mutants of a program are examined, the original implementation can be used as a reference implementation with the same internal structure. For the test, 1000 outputs were generated for each mutant and compared to the outputs of the original implementation. In total 139 mutants have been killed with this method.

7. DISCUSSION

In the first empirical study, the proposed tests are very successful. Almost 95% of the mutants which are killed by the original implementation (i. e. the original implementation has been used as a perfect oracle for random and lattice-based testing) could also be killed by the statistical hypothesis test approach. It has to be noticed that increasing the sample size also increases the number of killed mutants. This

can be observed up to a sample size of $n = 500$. This can be explained by the fact that statistical hypothesis tests become more powerful when the sample size increases. In this case, mutants whose results differ only slightly from the theoretical values are killed. But even with a high sample size, some mutants cannot be killed by the statistical testing approach. Therefore, statistical testing should be combined with other testing strategies in case of deterministic SUTs.

The statistical hypothesis test for the theoretical variance kills more mutants than the statistical test for the theoretical mean does. Additionally, the results of the combination of the two methods clearly show that both tests kill (at least partially) different mutants, as the number of mutants killed by the combination is noticeably larger than the number of mutants killed by only one of the methods. Therefore, testing only for one characteristic does not seem to be sufficient.

In the tests against a golden implementation, the test for equal linear variabilities is the most effective one. A closer examination of the data has shown that this test kills the same mutants as the test for equal variances. Therefore, it can be assumed that the test for equal linear variabilities is a good substitute for the test for equal variances.

In the second study, the results are less favourable. Only 47% of the mutants which are detected with fixed seed are killed using the original implementation as a perfect oracle. This is probably due to the fact the the complex output has been reduced to a set of four numerical characteristics.

In contrast to the first study, the number of mutants additionally killed by the test for equal linear variabilities is very low. This can be explained by the high correlation of the mean and variance in this case. The examined characteristics should follow a Poisson distribution, where the mean and the variance are equal.

Comparing the number of killed mutants to the total number of mutants may lead to the conclusion that only a small number of mutants could be killed, even by deterministic testing. This may be caused by the loss of information in the computation of the characteristics. Therefore, other tests either based on other characteristics or based on the (non-transformed) program output may be more successful.

A drawback of the statistical testing approach is a high consumption of resources, at least about 250,000 program executions are necessary to obtain a stable test result. However, the use of statistical methods allows to draw conclusions about the correctness of the program, as the chosen confidence level α, the sample size n, and the number of reruns determine the probability of a wrong test decision. Therefore, when the SUT passes a test, it can be assumed that the tested characteristic is equal to the characteristic of the specification (with the accuracy δ). But note that the program may behave different for other than the tested characteristics.

As mentioned, the proposed approach can also be applied to deterministic programs with random input. In this case, the usual problem of random testing also applies: The tests will only cover branches and paths that are easy to cover. Other branches and paths will not be covered and thus not be tested. However, other distributions that are more likely to trigger other paths can be used too in our approach. Despite of that, our approach is a suitable first test approach that should be applied in combination with other testing strategies in case of a deterministic SUT.

Threats to Validity

At present, the number of studies is too small in order to propose statistical hypothesis tests as a general approach to testing randomized software. However, several other studies have shown that it is a useful tool (cf. [4, 17, 18, 26]).

The effectiveness of our approach has mainly been compared to that of random testing with a (deterministic) perfect oracle (cf. Study 1). It may be argued that other methods are more effective (than random testing and lattice-based testing). That may be true, but our choice was pragmatic and we think that it is fair to compare the approach with random testing since randomness is also involved in our approach.

We cannot guarantee that the outputs of the mutants of our first empirical study follow a normal distribution. Thus, the power of our statistical hypothesis test comparing the variances is uncertain.

Finally, the outcome of our studies depends on the usefulness of mutation analysis, which is however generally accepted as a good means for evaluation purposes.

8. CONCLUSION

Most testing approaches available are intended for deterministic software systems. However, there are lots of systems that produce random output (e. g. programs for random simulation). In this case, classical testing approaches are not applicable. For instance, the expected results cannot be precomputed for test data, since there are several outputs possible. Furthermore, several executions of the same or equivalent software systems will produce different outputs. Thus, the Gold Standard Oracle [3] is also not applicable.

The present paper has proposed a testing approach based on statistical hypothesis tests. This approach makes it possible to test randomized software. It can be applied in case that theoretical values are known w. r. t. the characteristics of the distribution of outputs. This is however not the only application. We have also shown how this method can be used with a golden implementation. Thus, no knowledge about the distribution of outputs is required in this case. We have furthermore investigated what kind of knowledge can be obtained by executing our approach. We are able to specify and to control the error probability of the decision with respect to the tested characteristics, where the accuracy of the comparison can explicitely be determined. Our approach thus makes it possible to exactly determine the information gained about the correctness of the implementation under test. This failure and correctness estimation and the extension of the approach to golden implementations are the novel contributions of our paper—besides the new empirical studies—compared to previous work on the application of statistical hypothesis test in order to test software [17, 18, 26].

We have presented the results of two empirical studies. The first one has been conducted with a module of an open-source software library, namely Apache Commons Math. The second one is based on an implementation taken from the GeoStoch library being used at France Télécom R&D, Paris. We have compared statistical hypothesis tests with deterministic tests using mutation analysis. The results of our studies are encouraging. The proposed approach was able to kill many of those mutants that have also been detected by the deterministic (w. r. t. the test evaluation)

testing approaches. Thus, statistical hypothesis tests seem to be a very useful tool for testing randomized software. They should however be complemented by other testing approaches, such as testing with fixed seed, if applicable. This combination is especially necessary in case of testing deterministic implementations under test with random input, which can also be done with the proposed approach.

The empirical studies presented are only a first evaluation of the approach. It will be necessary to conduct more studies in order to assess the proposed approach in more detail.

9. REFERENCES

[1] Apache Software Foundation. Apache Commons Math homepage. http://jakarta.apache.org/commons/math.

[2] J. Bible and G. Rothermel. A unifying framework supporting the analysis and development of safe regression test selection techniques. Technical Report 99-6011, Oregon State University, 1999.

[3] R. V. Binder. *Testing Object-Oriented Systems*. Addison-Wesley, 1999.

[4] J. Bohrmann. On random testing and test oracles in the context of credit risk. Diploma thesis, Faculty of Computer Science, Ulm University, 2006.

[5] G. Casella and R. L. Berger. *Statistical Inference*. Wadsworth Group, Duxbury, CA, USA, 2002.

[6] A. Di Pierro and H. Wiklicky. Probabilistic abstract interpretation and statistical testing. In *Proceedings of the Second Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification*, volume 2399 of *Lecture Notes in Computer Science*, pages 211–212. Springer-Verlag, 2002.

[7] C. Gloaguen, F. Fleischer, H. Schmidt, and V. Schmidt. Simulation of typical Cox–Voronoi cells with a special regard to implementation tests. *Mathematical Methods of Operations Research (ZOR)*, 62(3):357–373, 2005.

[8] C. Gloaguen, F. Fleischer, H. Schmidt, and V. Schmidt. Fitting of stochastic telecommunication network models via distance measures and Monte–Carlo tests. *Telecommunication Systems*, 31(4):353–377, 2006.

[9] C. Gloaguen, F. Fleischer, H. Schmidt, and V. Schmidt. Modelling and simulation of telecommunication networks: Analysis of mean shortest path lengths. In R. Lechnerova, I. Saxl, and V. Benes, editors, *Proceedings of the 6th International Conference on Stereology, Spatial Statistics and Stochastic Geometry*, pages 25–36. Union of Czech Mathematicians and Physicists, Prague, Czech Republic, 2006.

[10] R. Hierons. Testing from a nondeterministic finite state machine using adaptive state counting. *IEEE Transactions on Computers*, 53(10):1330–1342, 2004.

[11] D. Hoffman. Heuristic test oracles. *Software Testing and Quality Engineering Magazine*, 1(2), 1999.

[12] Inst. of Stochastics, Ulm University. GeoStoch homepage. http://www.geostoch.de/.

[13] K. N. King and A. J. Offutt. A fortran language system for mutation-based software testing. *Software Practice and Experience*, 21(7):685–718, 1991.

[14] D. Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22(3):328–350, 1981.

[15] Y.-S. Ma, J. Offutt, and Y.-R. Kwon. MuJava: An automated class mutation system. *Software Testing, Verification, and Reliability*, 15(2):97–133, 2005.

[16] R. Maier and V. Schmidt. Stationary iterated tessellations. *Advances in Applied Probability*, 35:337–353, 2003.

[17] J. Mayer. On testing image processing applications with statistical methods. In *Proceedings of Software Engineering 2005 (SE 2005)*, volume P-64 of *Lecture Notes in Informatics*, pages 69–78, Bonn, Germany, 2005. Köllen Druck+Verlag GmbH.

[18] J. Mayer and R. Guderlei. Test oracles using statistical methods. In *Proceedings of the First International Workshop on Software Quality (SOQUA 2004)*, volume P-58 of *Lecture Notes in Informatics*, pages 179–189, Bonn, Germany, 2004. Köllen Druck+Verlag GmbH.

[19] J. Mayer, V. Schmidt, and F. Schweiggert. A unified simulation framework for spatial stochastic models. *Simulation Modelling Practice and Theory*, 12(5):307–326, 2004.

[20] J. Mecke. Parametric representation of mean values for stationary random mosaics. *Mathematische Operationsforschung und Statistik Series Statistics*, 15:437–442, 1984.

[21] D. Monniaux. An abstract Monte-Carlo method for the analysis of probabilistic programs. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 93–101. ACM Press, New York, NY, USA, 2001.

[22] D. Monniaux. Abstraction of expectation functions using Gaussian distributions. In *Proceedings of the 4th International Conference on Verification, Model Checking, and Abstract Interpretation*, volume 2575 of *Lecture Notes In Computer Science*, pages 161–173. Springer-Verlag, 2002.

[23] L. Nachmanson, M. Veanes, W. Schulte, N. Tillmann, and W. Grieskamp. Optimal strategies for testing nondeterministic systems. *SIGSOFT Software Engineering Notes*, 29(4):55–64, 2004.

[24] J. Offutt and R. H. Untch. Mutation 2000: Uniting the orthogonal. In *Proceedings of Mutation 2000: Mutation Testing in the Twentieth and the Twenty First Centuries*, pages 45–55, 2000.

[25] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1995.

[26] H. Sevcikova, A. Borning, D. Socha, and W.-G. Bleek. Automated testing of stochastic systems: A statistically grounded approach. In *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2006)*, pages 215–224. ACM, 2006.

[27] C. Szyperski, D. Gruntz, and S. Murer. *Component Software – Beyond Object-Oriented Programming*. Addison-Wesley / ACM Press, 2nd edition, 2002.

[28] E. W. Weyuker. On testing non-testable programs. *The Computer Journal*, 25(4):465–470, 1982.

[29] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.