An Empirical Study on the Test Adequacy Criterion Based on Coincidental Correctness Probability∗

Xiaoli Zhou†, Linzhang Wang†, Xuandong Li† and Jianhua Zhao†§
†State Key Laboratory of Novel Software Technology
Dept. of Computer Sci. and Tech. Nanjing University, Nanjing, Jiangsu, P.R. China 210093
Email: zhaojh@nju.edu.cn

Abstract—Coincidental correctness occurs when a fault is executed but no failure is detected. Coincident correctness is prevalent and adversely affects the effective of testing. In this paper, we improve the test adequacy criterion based on coincidental correctness probability which is proposed in our previous work and make an empirical study comparing our test adequacy criterion with two coverage-based test criteria. We take the Pearson Correlation Coefficient between the adequacy level and the capability of the test suite to find errors as a metric of the effectiveness. Our experiment involved 20 seeded versions of the programs from SIR. The results consistently approve that our test adequacy can effectively quantizes the capability of test suite to find errors.

I. INTRODUCTION

Currently, coverage-based test adequacy criteria are widely used both in academia and industry. However, high code coverage does not mean high test adequacy regarding to the defect removing. An empirical study reports that there does not exit a causal relationship between code coverage and defect coverage [1].

Coverage-based testing for white box assume that the execution of faulty statement has a high probability to result in observable incorrect output. However, this is not necessarily always the case. A fault can be observed during testing, if and only if the following conditions all be met: 1) the defect is executed or reached, 2) the program has transitioned into an infectious state, 3) the infection has propagated to the output [2]. When 1)(is met and 2) the coincidental correctness occurs[3][4]. The error may be hidden even if the faulty statement is executed. The main reason is the existence of coincidental correctness. Many experimental evidences show that, coincidental correctness is prevalent and responsible for reducing effectiveness of testing [3][4].

In our pervious work [5][6], we propose a test adequacy criterion based on coincidental correctness probability (CCP for short). In this paper, we make an improvement and an empirical study on the CCP-based test adequacy criterion. The main contributions of this paper can be concluded as follows:

i) We improve the algorithm to evaluate the coincidental correctness probability with a refined control-flow analysis.

ii) We take the Pearson Correlation Coefficient as an evaluation metric and make a comparison case study against the test adequacy criteria based on the statement coverage and branch coverage. Experiments in this paper can more accurately reflect the situation in practice, and shows that CCP-based test adequacy criterion has a stronger correlation with errors than other criteria.

Section II shows the approach to estimate the CCP-based adequacy in detail. Section III presents the experimental work and results. Finally, Section IV discusses some related works and presents our conclusion and future works.

II. AN APPROACH TO ESTIMATE THE CCP-BASED TEST ADEQUACY CRITERION

This section describes how to estimate the CCP, and presents the definition of CCP-based adequacy criterion. The basic idea is the same with previous works [5][6], and we improve the algorithm with a refined control-flow analysis.

An execution of a statement is called a statement instance. During the execution, statement instances load or store values in some memory units. Testers usually directly or indirectly check some memory units which are called check points at the end of the execution to determine whether the test is passed. The value stored in a memory unit is defined in some statement instances and then used in other statement instances. Thus these instances have a use-definition dependence for their correctness. If an error is triggered and generates an incorrect value, this incorrect value may be stored into some memory units. If other statement instances use these incorrect memory units, the incorrect values may be transferred till the end of this run or may be covered by other statement instances.

Before everything begins, here are two hypotheses:

i) We estimate the test adequacy only for the test suite which is passed in testing.

ii) We assume that a program has only one error.

A. Instrumentation and static analysis

We identify variables and operations in statements for instrumentation. For every variable, we instrument to get its runtime variable values, its memory address and how it is related (use or definition). Also, we build the static control-flow graph of the program.

B. Build dynamic dependencies

In this section, we described how to construct the dynamic use-definition and control-flow dependencies for runtime statement instances after running the instrumented program.
1) Build the dynamic use-definition dependence graph:
We build a dynamic use-definition dependence based on the memories used or defined by instances. The detailed method is described in [5].

2) Build the control-flow dependence: Control-flow can indirectly affects the correctness of the output of a program. If the conditional expression generates an incorrect value, some statements will be mistakenly executed while some other statements will be mistakenly unexecuted. The influences on the mistakenly unexecuted instances are ignored in this paper as they are difficult to analysis.

Conditional transfer statements and loop statements are two kinds of typical control statements. For a statement instance \( R \), \( CE(R) \) denotes the set of control statements on which \( R \) is dependent. \( CE(R) \) is computed as follows:

i) If \( R \) is an instance of a statement in a branch of an IF statement, the corresponding instance of conditional expression of this IF statement is in \( CE(R) \).

ii) If \( R \) is an instance of a statement in the body of a WHILE statement, all instances of the conditional expression of the WHILE statement from the start of the WHILE statement to \( R \) are in \( CE(R) \).

If \( R \) is in more than one branches or bodies of IF/WHILE statements, \( CE(R) \) will be computed for every IF/WHILE statement according to the above rules.

C. Impact factors for correctness possibility

In this section, we describe how the use-define and control-flow dependencies have an impact on the correctness of the outputs of a program execution and specify the degrees of these effects.

1) Impact factors of Operations: The impact of use-define dependence mainly arises from operations which may cover the incorrect intermediate results. The result of an operation has a probability to be correct even if its operands are wrong. We specify the probabilities that an operation generates an correct result under different situations. Because of space limitation, we only describe how the probabilities about binary operations are specified.

<table>
<thead>
<tr>
<th>Correctness of Left Operand(a)</th>
<th>Correctness of Right Operand(b)</th>
<th>Effect degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>( Q_1 = 1.0 )</td>
</tr>
<tr>
<td>*</td>
<td>F</td>
<td>( Q_2 )</td>
</tr>
<tr>
<td>F</td>
<td>*</td>
<td>( Q_3 )</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>( Q_4 )</td>
</tr>
</tbody>
</table>

We use “a*b” to represent an binary operation with an operator “*” and two operands a, b. In Table 1, \( Q_1, Q_2, Q_3 \) and \( Q_4 \) are the conditional probabilities that the result of “a*b” is correct under different situations. When \( a \) and \( b \) are both correct, “a*b” is definitely correct. So \( Q_1 \) is always 1.0. The values of \( Q_2, Q_3 \) and \( Q_4 \) are differently specified for different operators.

Assuming that \( P_a, P_b \) and \( P_a*b \) are the correctness probabilities of two operands and the result respectively, \( P_a*b \) can be estimated by:

\[
P_{a*b} = P_a \times P_b \times 1.0 + P_a \times (1 - P_b) \times Q_2 + (1 - P_a) \times P_b \times Q_3 + (1 - P_a) \times (1 - P_b) \times Q_4
\]  

(1)

2) Impact factors of control-flow: Control-flow dependence is used to revise the correctness probability estimated from use-define dependence. For an instances \( R \), the revision is:

\[
P'(R) = P(R) \times (\Pi_{R' \in CE(R)} P(R') + (1 - \Pi_{R' \in CE(R)} P(R')) \times Adj(ED))
\]  

(2)

Here, \( P(R) \) and \( P(R') \) are the correctness probabilities of \( R \) before and after the revision, and \( Adj(ED) \) is the effect degree to adjust the probability. If any instances in \( CE(R) \) is incorrect, the result of \( R \) has a small probability to be correct. Thus, we set the \( Adj(ED) \) as 0.2 to adjust the probability of \( R \) under the situation of incorrect runtime control-flows.

D. An algorithm to estimate the coincidental correctness probability

The algorithm to estimate the CCP of one program execution is described as Figure 1. For an execution, we estimate one CCP for each statement in the program assuming that this statement contains an error and all other statements are correct.

Given a serial program and a test case, we collect runtime information by executing the instrumented program. For a statement instance \( R \), we use the \( UG(R) \) and \( CG(R) \) to denote the set of memory units that used or changed by \( R \). We suppose that the \( UG, CG \) and \( CE \) of each instance are already computed as described in previous sub-sections. In the algorithm, we estimate the correctness probability of each instance by order. Finally, the CCP of an execution is the probability that all check points are coincidentally correct.

Input:

\( S \): The statement is assumed to contain an error.

\( InstancesList \): The runtime instances sequence.

\( MemorySet \): The memory unit set used/updated by instances.

\( Checkpoints \): the set of check points.

Output: CCP of the program execution

1: Initialize correctness probability of all memory units in MemorySet as 1.0;
2: for each node \( R \) in InstancesList do
3: if \( R \) is an instance of \( S \) then
4: \( P \)the result value of \( R \) is wrong by default.*
5: Set the correctness possibility of memory units in CG(R) as 0;
6: else
7: \( \star \). Estimation based on use-define dependence*/
8: Estimate the correctness possibility of memory units in CG(R) based on the correctness probability of memory units in UG(R) and effect degrees according to the formulation (1);
9: \( \star \). Revision with control-flow dependence*/
10: if \( CE(R) \neq 0 \) then
11: Revise the correctness probability of memory units in CG(R) according to formulation (2);
12: end if
13: end if
14: end for
15: return CCP = \( \Pi_{K \in Checkpoints} P(K) \)
16: \( \star \). P(K):the correctness possibility of the value in check point \( K \)

Fig. 1. The algorithm to estimate the CCP of a program execution

The CCP of a program execution w.r.t. a statement denotes the probability that there is an error in this statement and it’s triggered but not observed by testers. A higher CCP means that we need more test cases to verify the suspect statement. The granularity of our approach is statement level.

E. The CCP-based adequacy criterion

For a statement \( S \), we get a set of CCPs, one for each test case. The test adequacy level of a test suite \( TS \) for statement \( S \) is defined as the following formulation:

\[
AC(S) = 1 - \Pi_{T \in TS} CCP(S, T)
\]  

(3)

Here, \( AC(S) \) is the adequacy level of the test suite for the statement \( S \). \( CCP(S, T) \) denotes the probability that the error
in $S$ is hidden by coincidental correctness after running a single test case $T$. $\Pi_{T \in Ts}CCP(S,T)$ is the probability that all the test cases in $Ts$ fail to find the error. The lower the $\Pi_{T \in Ts}CCP(S,T)$, the higher the probability of finding errors will be. Thus, the $AC(S)$ denotes the probability to find the error in the statement $S$ if the error exists. For a test suite, we collect a set of $AC$s, one for each statement in the program.

For an adequacy level $AL$ of the whole program, the test suite satisfy adequacy level $AL$ only when all $AC$s of statements is higher than $AL$. This means that, when a test suite satisfy the adequacy level $AL$, all the probabilities that an error is hidden in each statements are lower than $1 - AL$.

III. EXPERIMENTAL WORK
A. Tool implementation and subject programs

Our experimental tool contains three modules: variable instrumentation module, runtime information analysis module and CCP-calculator. The variable instrumentation module is built on the grammar tool ANTLR. This module identifies variables and operations in statements and instruments the program for collecting runtime information. After running the instrumented program, runtime information analysis layer collects runtime information to build the dynamic use-definition and control-flow dependencies as described in the subsection II-B. The CCP-calculator computes the CCP of each test case assuming an error in each statement and finally gives the test adequacy level of the test suite. Because of the time and efforts limitation, programming language treated by our tool is limited to a subset of GNU C[7].

We involved 20 seeded versions of the selected two C programs with suitable size and the corresponding test suites from Software-artifact Infrastructure Repository (SIR)[8]. Table II provides the basic information about the subject programs and the test suites, including the sizes of the programs, test suites size($|T|$), the number of failed test cases($|T_F|$) and the percentage of failed tests. All seeded versions of programs contain an error.

| Program (LOC,Procedures) | $|T|$ | $|T_F|$ | $|T_F|/|T|$ |
|--------------------------|------|------|---------|
| tcas_v1.c                | 173, 9 | 1608 |          |
| tcas_v2.c                | 252   | 0.16 |          |
| tcas_v3.c                | 319   | 0.92 |          |
| tcas_v4.c                | 139   | 0.09 |          |
| tcas_v5.c                | 132   | 0.08 |          |
| tcas_v6.c                | 36    | 0.02 |          |
| tcas_v7.c                | 36    | 0.02 |          |
| tcas_v8.c                | 126   | 0.08 |          |
| tcas_v9.c                | 259   | 0.15 |          |
| tcas_v10.c               | 125   | 0.08 |          |
| tcas_v11.c               | 142   | 0.09 |          |
| tcas_v12.c               | 190   | 0.12 |          |
| totinsفق_w1.c            | 29    | 0.03 |          |
| totinsفق_w2.c            | 199   | 0.19 |          |
| totinsفق_w3.c            | 36    | 0.03 |          |
| totinsفق_w4.c            | 277   | 0.22 |          |
| totinsفق_w5.c            | 145   | 0.14 |          |
| totinsفق_w6.c            | 45    | 0.04 |          |
| totinsفق_w7.c            | 565, 7 | 1052 | 71 0.07 |

B. Metric of Criterion Effectiveness and Experiment Design

Test adequacy aims to reflect that to what extent the test suite can prove the correctness of program. If the correlation between the test adequacy level and the capability to find errors is strong, we think this test adequacy criterion can accurately reflect the capability of the test suite to find errors and also the capability to prove the correctness of the program. Hence, to investigate the effectiveness of our adequacy criterion, we take the correlation between the adequacy level and the probability to detect the error as a main metric.

1) Foundation knowledge about correlation: To quantify the correlation and dependence between two factors, we calculate the Pearson correlation coefficient(PCC for short). In statistics, Pearson correlation coefficient is widely used in the sciences as a measure of the strength of linear between two variables, giving a value between $+1$ and $-1$ inclusive.

Definition 1: Let X and Y be two zero-mean real-valued random variables. PCC is defined [9] as follows:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y}$$

(4)

Where $cov(X,Y)$ is the covariance of the two variables, $\sigma_X$ and $\sigma_Y$ is the standard deviations of X and Y respectively.

If $|\rho_{X,Y}|\leq0$, then X and Y are said to be uncorrelated. The closer the value of $|\rho_{X,Y}|$ is to 1, the stronger the correlation between X and Y. Generally, when the absolute value of correlation is bigger than 0.8, this correlation is considered to be extremely strong.

2) Experiment Design: We compare the CCP-based adequacy criterion with the statement coverage-based and branch coverage-based adequacy criterion.

For every seeded version of programs, we get the PCC between the adequacy level and the capability to find errors, by the following steps:

i) Get the preliminary information. We increase the adequacy level from 0.1 to 0.9 with a gap of 0.1. For each adequacy level, a test suite is built by adding distinct test cases from the test-case pool provided by SIR till the adequacy level is reached. For each adequacy level, we build 10000 test suites and record the percentage of test suites that can find the error. For each criterion (CCP-based criterion, statement coverage and branch coverage), we can get the 9 pairs of $(AdqLevel, ErrFindProb)$, where $AdqLevel$ ranges from 0.1 to 0.9.

ii) Calculate PCC. For each criterion, we calculate the PCC between the $AdqLevel$ and the $ErrFindProb$ using the formulation (4).

C. Result and analysis

This subsection shows the results of the experiments described in III-B2.

1) The preliminary information of correlation: The couples of $(AdqLevel, ErrFindProb)$ of CCP-based criterion are showed in Figure 2. Every line in the graph presents the 9 couples of $(AdqLevel, ErrFindProb)$ for a seeded version of programs. The detailed results for other two criteria is not presented here because of space limitation. For all seeded programs, the $ErrFindProb$ grows consistently with the $AdqLevel$. For most seeded versions of programs, when the $AdqLevel$ tends to be 1, $ErrFindProb$ also tends to be 1.
These methods can not identify the coincidental correctness independently and their accuracy relies on the whole test suites. Boundary value analysis also suffers from coincidental correctness. Work in [13] studies the cases where shifts in boundaries can get undetected due to coincidental correctness and proposes a solution to generate tests that not suffer from predictable coincidental correctness.

In our previous work[5][6], we propose a new test adequacy criterion based on coincidental correctness probability. In this paper, we make an improve the estimation algorithm and make a empirical study which shows that our adequacy criterion has a stronger correlated relationship with bugs found than other criteria based on the statement coverage and branch coverage.

The effect degrees of operators and the control-flow in this paper are specified according to our intuitive knowledge. In the future, we will try to find more appropriate effect degrees for different operators and the control-flow using some approaches like data mining/machine learning.

REFERENCES