2016 IEEE/ACM 38th IEEE International Conference on Software Engineering Companion

Impact of CS programs on the quality of test cases generation: An empirical study

Omar S. Gómez
Prometeo-Senescyt and,
Escuela Superior Politécnica
de Chimborazo
Av. Panamericana Sur km 1.5
Riobamba 060106, Ecuador
ogomez@espoch.edu.ec

Sira Vegas
Universidad Politécnica de
Madrid
Campus de Montegancedo
28660 Boadilla del Monte
Spain
svegas@fi.upm.es

Natalia Juristo
Universidad Politécnica de
Madrid
Campus de Montegancedo
28660 Boadilla del Monte
Spain and,
Department of Information
Processing Science
University of Oulu, 90014,
Oulu, Finland
natalia@fi.upm.es

ABSTRACT

Background: Although most Computer Science (CS) programs offered by higher education institutions usually include a software engineering course, some works report a lack of formal training in software testing. **Aim:** With the aim of studying the possible impact of knowledge acquired from CS programs on software testing, this paper reports an investigation composed of four experiments. The experiments conducted in Spain, Mexico and Ecuador examine the quality of test cases (TC) generated using black-box and white-box methods. The subjects of the experiments were undergraduate and graduate students who were exposed to different levels of CS knowledge. **Method:** We pool together the data from the four experiments and apply logistic regression to investigate possible relations of the quality of test cases with students' level of exposure to CS knowledge. **Results:** The quality of test cases generated by students depend significantly on the amount of CS program studied. The odds of generating test cases that reveal failures against those that do not reveal decrease when students are exposed to a low level of CS knowledge. **Conclusions:** Software testing plays a key role in what is an increasingly complex process of developing and maintaining software products today. The results of our empirical study provide evidence in favor of greater formal training in software testing as part of CS programs.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: General; D.2.8 [**Software Engineering**]: Metrics—*Software science*

ICSE '16 Companion, May 14-22, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2889160.2889190

Keywords

software testing education, software testing, controlled experiment, logistic regression

1. INTRODUCTION

Due to software's inherent complexity [6], it cannot be exhaustively verified [28]. Generate and run enough test cases to adequately test the software product is critical.

A test case specifies the input values and the expected output. The purpose of generating test cases is to exercise the software product, seeking for failures [19].

The effectiveness of test cases can be determined by the number of failures that they are capable of revealing [8]. Therefore that test case quality is higher, the more failures they reveal.

Two of the test cases generation methods commonly discussed in the literature [15, 5, 28, 20, 13], referenced by academia [33, 11] and applied in industry [30, 7, 12] are black-box [17, 28] and white-box [27, 28]. Black-box testing (also known as functional) views the software product as a black box, ignoring its internal mechanisms and focusing on specifications. White-box testing (also known as structural) views the software product as a glass box examining the internal structure to generate test cases.

There are works discussing the effectiveness of test cases [4, 35, 18, 38, 8, 39, 25], most of them from the viewpoint of test automation. We have not found any work examining the quality of test cases from the viewpoint of the level of knowledge of the person whom generates it.

Although most CS programs offer at least one course on software engineering (SE), there is evidence to show that formal software testing training is deficient. For example, Chan et al. [7] found that less than 20% of software testing teams in Hong Kong had received formal software testing training at university. It is common practice for CS programs to briefly touch upon the topic of software testing as one of the issues addressed in SE courses [37]. According to [36], some US universities offer around 30 software testing courses as part of undergraduate CS programs but no one of the CS programs in the US has as prerequisite a software testing course.

The goal of our research is to study what impact knowl-

Table 1: Quality of test cases generated by undergraduate and graduate students and practitioners developers reported in [4]

Subjects	Test case effectiveness
(under)graduates	74.59%
practitioners	62.11%

edge acquired from CS programs offered by higher education institutions may have on test case quality. We examine the quality of test cases generated using black-box and white-box methods by students participating in four experiments we conducted. In each experiment, students were exposed to different levels of CS knowledge.

Our findings show how exposure to knowledge acquired in CS programs may have an impact on software testing, particularly on the quality of the test cases generated.

The remainder of the paper is structured as follows. Section 2 presents the related work. Section 3 describes the experiments that we conducted. Section 4 describes the analysis and interprets the results. Section 5 discusses the findings of our study. Finally, Section 6 outlines the conclusions.

2. RELATED WORK

Our work most closely resembles the research reported in [4]. One of the issues examined by Basili et al. [4] is test case effectiveness in terms of how well able they are to reveal failures. To do this, authors use undergraduate university students (upper-level computer science major) together with graduate students and professional software developers.

Based on the metric reported in [8], Equation (1) is able to determine the effectiveness of a test case.

$$effectiveness_{cp} = \frac{N_{frtc}}{N_f} \times 100\,. \quad (1)$$

In Equation (1), N_{frtc} indicates the number of failures revealed by the test case (where one failure is associated with one fault located in the software product), whereas N_f represents the number of faults that a particular software product contains.

A set of programs injected with several faults can be exercised provided appropriate test cases are generated. In [4] test case effectiveness (test case quality) is reported in terms of the percentage of faults that they reveal. Table 1 shows how effective the test cases generated by the two groups of subjects that participated in the study reported in [4] were.

According to Table 1, undergraduate and graduate students generate test cases that reveal a higher percentage of faults in a set of four programs for verification injected with 34 faults[1] than practitioners.

Although there are similar works [24, 31, 22, 23, 10, 14] related to [4], It is not possible to determine the quality of the test cases generated, as they do not report effectiveness in terms of how good test cases are at revealing failures.

3. STUDY SETTING

This study is related to the experiments reported in [4, 24, 31, 22, 23, 10, 14] concerned primarily with the effectiveness of several software testing methods. One of the activities within this family of experiments is to generate test cases using black-box and white-box methods (although this activity is reported only in [4]). In this work, we study the quality of the test cases generated by the subjects applying either the black-box or white-box method.

In this research we use data from four experiments that we run. Two were conducted at Madrid Technical University (Spain) and the other two at the Autonomous University of Yucatan (Mexico) and the Technical School of Chimborazo (Ecuador), respectively. The experiments were conducted at different sites with undergraduate and graduate students.

The first experiment was conducted in winter 2009 as part of the 10-semester undergraduate computer engineering program at the School of Computer Engineering, Madrid Technical University (UPM)[2]. The second experiment was conducted in winter 2012 again at the UPM with participants from the undergraduate computer engineering program and the graduate students taking either the European Master on Software Engineering[3], the Master in Software and Systems[4] or the PhD Program in Software and Systems [5]. The third experiment was conducted in March 2013 as part of the eight-semester undergraduate software engineering degree at the School of Mathematics, Autonomous University of Yucatan (UADY)[6], whereas the fourth experiment was run in December 2014 as part of the 10-semester undergraduate systems engineering program at the School of Computing and Electronics, Technical School of Chimborazo (ESPOCH)[7]. Table 2 shows the relevant characteristics of the students participating in these experiments.

3.1 CS curricula

The main focus of undergraduate CS programs is on teaching the theoretical and practical foundations of computing. Regarding mathematics, it is common that CS programs include courses on differential and integral calculus, linear algebra, probability and statistics, as well as discrete mathematics. Computing-related courses include the theory of computation, operating systems, numerical computation, compilers, distributed systems, data structures, computer networking, computer graphics, software engineering, artificial intelligence, algorithm design and analysis, programming and so on. The mix between the number of mathematics and computing courses varies in CS programs mostly depending on the type of school or department hosting the program. For example, if it is offered by a school of engineering, the CS program is very likely to include courses on calculus or differential equations [1].

Assuming that a CS program is basically composed of courses related to mathematics and computing, plus co-curricular courses that may be related to other branches of knowledge, Table 3 shows the distribution of the CS program modules at the universities where the experiments for this study were conducted.

[1] where nine faults correspond to program 1, six to program 2, seven to program 3 and 12 to program 4.

[2] https://www.fi.upm.es/?id=gradoingenieriainformatica
[3] http://www.dlsiis.fi.upm.es/emse/
[4] http://muss.fi.upm.es/
[5] http://www.dlsiis.fi.upm.es/docto_ss/
[6] http://www.matematicas.uady.mx/index.php/planes-estudio/licenciaturas/licenciatura-en-ingenieria-de-software
[7] http://www.espoch.edu.ec/index.php?action=facultades&id=4

Table 2: Characteristics of students used in the four experiments

Exp.	Academic year	CS program	University	Country	n
1 (2009)	5th-yr undergraduate	B.S. Informatics	UPM	Spain	32
2 (2012)	1st-yr graduate	M. Soft. Eng. (EMSE and MUSS)	UPM	Spain	18, 10
3 (2013)	1st-yr undergraduate	B.S. Soft. Eng.	UADY	Mexico	15
4 (2014)	4th-yr undergraduate	B.S. Soft. & Syst. Eng.	ESPOCH	Ecuador	9

Table 3: CS Program degree areas distribution

Area	UPM (B.S.)	UADY (B.S.)	ESPOCH (B.S.)	UPM (M. EMSE)	UPM (M. MUSS)
Mathematics	23%	21%	14%	0%	0%
Computer related	65%	62%	61%	87%	85%
Co-curricular	12%	17%	25%	13%	15%

As Table 3 shows, the ratio of computing courses to mathematics courses in these CS programs is about three to one. As regards the master's programs selected for this study (EMSE and MUSS), the courses are composed of more topics related to computing, especially software engineering, than other co-curricular courses such as seminars or courses related to thesis development.

Table 4 describes the level of exposure to CS knowledge that subjects had at the time of participation in the experiments. We measure such exposure level as the percentages of full courses completed by the student. For example, UPM undergraduate students had taken 79% of the computing-related courses at the time of the experiment, where 100% of computing-related courses accounts for 65% of courses in the field of computing (according to the CS program degree areas distribution, see Table 3).

As shown in Table 4, students taking master's programs have completed 100% of the courses of an undergraduate CS program.

3.2 Research question

The research question addressed in this study is defined as: Is the level of CS program exposure related to the quality of test cases generated with black-box and white-box methods by undergraduate and graduate students?

The null hypothesis (H_0) used to answer this question is defined as follows: "The level of exposure of a CS program does not influence the quality of the test cases generated using black-box and white-box methods."

The quality of the test cases is measured as a percentage of the number of faults revealed by the generated test cases to the total number of faults injected in an instrumented program.

3.3 Description of experiments

In the first experiment (UPM 2009) a 3^3 factorial design [21] was used to study some aspects of the black-box (equivalence partitioning) [17, 28], white-box (decision coverage) [27, 28] and code reading by stepwise abstraction[26] methods which were applied to three instrumented programs. This design is reported in [22, 23]. In a similar way, the second experiment (UPM 2012) used a 2^2 factorial design [21] where only the black-box and white-box testing methods were applied to two instrumented programs. In the third experiment (UADY 2013) a 2^2 factorial design [21] was used to study some aspects of the black-box and white-box methods applied to two instrumented programs. This experiment is reported in [14]. In these three experiments, the factor-level (method and program) combinations were assigned at random to students. Finally, the design used for the fourth experiment (ESPOCH 2014) was fully randomized where the black-box and white-box treatments were assigned randomly to subjects. Table 5 shows the differences and similarities between the experiments used in this study.

In all four experiments, the numbers of faults revealed with respect to faults not revealed by the test cases are measured based on the treatment combinations for the testing methods (black-box and white-box) applied to an instrumented program.

Figure 1 shows the treatment combinations making up the experimental designs of these experiments. In gray we highlight the treatment combinations selected for the study reported here.

In view of the characteristics of this study, it is an observational empirical study: we draw inferences from the data extracted from part of previous experiments (particularly the treatment combinations black-box and white-box, both on ntree program) [32].

With respect to their execution, the first experiment was conducted at the end of 2009 as part of an information systems evaluation course. This course was taught in the first semester of the fifth and final year of the now phased-out computer engineering program (as per the 1996 curriculum) at Madrid Technical University's School of Computing (now known as the School of Computing Engineering). As part of this course, students learned the foundations of software verification, as well as some verification methods including static and dynamic testing. At this point of their degree program, students had completed about 80% of the computing-related courses (see Table 4). Students had already taken compulsory computing courses like programming methodology, physical foundations of computer science, hardware foundations, theory of computation, computer technology, computer architecture, data structures I and II, systematic programming, operating systems, computer architecture laboratory, computer networking, computer architecture, concurrent programming, program development models, artificial intelligence, software engineering I and II, compilers, networking architectures, operating systems design, databases, computer systems and knowledge engineering. Note that students have some notions of software testing, as this issue is generally addressed in the software engineering I and II courses.

Before executing the experiment, students received train-

Table 4: Exposure to CS knowledge in terms of official courses completed at the time of the experiments

Area	UADY (B.S.)	ESPOCH (B.S.)	UPM (B.S.)	UPM (M. EMSE)	UPM (M. MUSS)
Mathematics	22%	100%	100%	100%	100%
Computer-related	8%	56%	79%	100%	100%
Co-curricular	14%	79%	50%	100%	100%

Table 5: (Dis)similarities of the experiments used for this study

		Treatment			Programs				
Exp.	Training	Black-box	White-box	C. Reading	cmdline	nametbl	ntree	banking	Faults
1 (2009)	Official course	•	•	•	•	•	•		6
2 (2012)	Official course	•	•			•	•		6
3 (2013)	Workshop	•	•		•		•		6
4 (2014)	Workshop	•	•	•	•	•	•	•	6

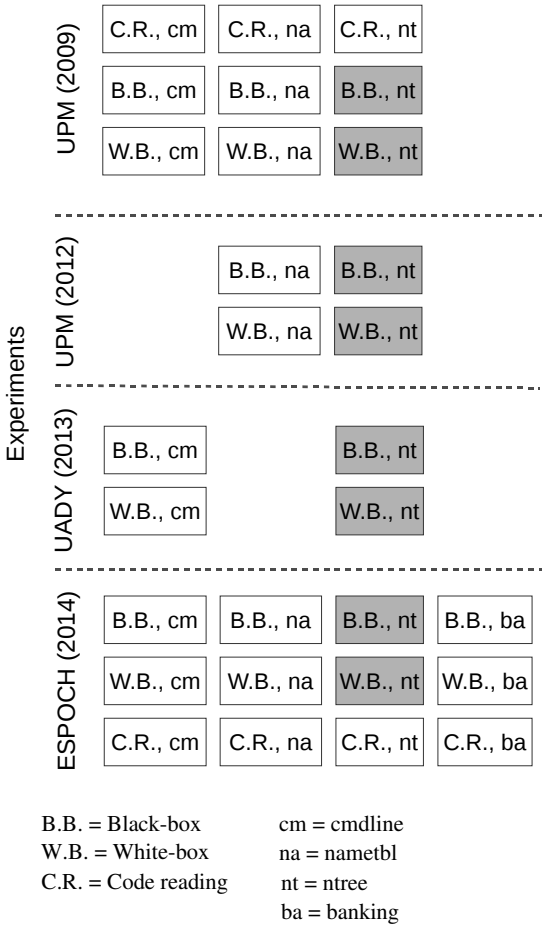

Figure 1: Selected experiment treatment combinations.

ing in how to apply the black-box [17, 28], white-box [27, 28] and code reading by stepwise abstraction [26] methods.

A total of 49 students participated in this experiment, each applying each of the methods to three different programs written in the C programming language. Each method was applied in a different session lasting about three hours. These programs were injected with six fault types [22, 23], further details of the used fault classification are available in the work of [3].

Each program has an approximate length of 250 LOC. For more details on the structure of this experiment, see the published reports in [22, 23]. For our research we have used data from this experiment regarding the test cases generated by 32 subjects that applied the black-box [17, 28] and white-box [27, 28] methods to one of the instrumented programs (identified as ntree).

The second experiment was again conducted at UPM's School of Computing. This experiment was performed as part of the verification and validation course offered to undergraduate students and as part of a graduate course for students of the European Master on Software Engineering (EMSE), Master in Software and Systems (MUSS) and PhD in Software and Systems. At undergraduate level, this course is offered in the same year and same semester as the experiment discussed above, whereas at graduate level it is offered during the first year of all three of the above postgraduate programs. Graduate students had taken 100% of CS program courses (see Table 4). A total of 63 students participated in this experiment: 31 undergraduate and 28 master's students (of which 18 were taking the EMSE and 10 the MUSS) and four were PhD students. The graduate students came from different parts of the world, on which ground we can assume that they will have taken a range of CS programs.

In this experiment, students applied black-box [17, 28] and white-box [27, 28] methods to two programs injected with six fault types according to the classification proposed in [3]. Each program has an approximate length of 250 LOC. Students worked either on two versions of the instrumented program: one in Java and the other in C (written using the same coding style). Also, two versions of the materials used by students such as forms and instructions were prepared: one in Spanish and the other in English. The students used their preferred language and programming language. We use data regarding test cases generated by 28 master's students (18 taking the EMSE and 10 taking the MUSS) applying

both the black-box [17, 28] and white-box [27, 28] methods to one of the instrumented programs (ntree) from this experiment.

The third experiment was conducted during the first two weeks of March 2013 [14] as part of a programming course. This course is taught in the second semester of the first year of the software engineering degree program (as per the 2004 curriculum) at the School of Mathematics of Autonomous University of Yucatan (UADY). The students participating in this experiment had taken the foundations of programming and foundations of software engineering courses in the previous semester, that is, had completed only about 8% of the computing-related courses at this point of their degree program (see Table 4). As part of the programming course, students received training in the operation and application of black-box [17, 28] and white-box [27, 28] and code reading by stepwise abstraction [26] methods in the weeks running up to the experiment. Although students were trained in all three testing methods, it was decided that for the experiment they apply only two (black-box and white-box) to two programs in order to increase the number of observations. Both programs were instrumented with six fault types according to the classification proposed in [3]. Each program has an approximate length of 250 LOC. The subjects used a web tool developed by the first author to record information regarding the test cases generated and the observed failures. From this experiment we have taken data regarding the test cases generated by 15 subjects who applied black-box [17, 28] and white-box [27, 28] testing methods to one of the instrumented programs (ntree).

The fourth experiment was performed in the second and third weeks of December 2014 at the School of Computing and Electronics of the Technical School of Chimborazo (ESPOCH) as part of a software verification workshop. The workshop consisted of eight sessions held on alternate days. Each session had a duration of three hours. The first sessions were used to train students in black-box [17, 28], white-box [27, 28] and code reading by stepwise abstraction [26] methods, and the experiment was conducted during the last sessions. A total of 10 systems engineering degree students (2013 program) regularly attended this workshop. The group of participants in this experiment was conformed by students taking the seventh and eighth semester of the fourth and final year of their degree program. At this point of the degree program, the students had completed approximately 56% of the computing-related subjects (see Table 4). Students had previously taken computing-related courses like foundations of programming, structured programming, object-oriented programming, data structures, databases I and II, computer architecture, web applications, operating systems, software engineering I and II, software architecture, advanced programming and embedded development.

The testing methods were applied in a controlled environment where the students generated and executed the test cases using a remote terminal. Four instrumented programs, following the same coding style as those used in [22, 23], were written in Java and used in this experiment. They had an approximate length of about 250 LOC. The programs were instrumented with six injected faults as per the classification proposed in [3]. Data regarding the test cases generated by nine subjects who applied black-box [17, 28] and white-box [27, 28] on one of the instrumented programs (ntree) is used here from this experiment.

As regards functionality, the selected program (ntree) implements an n-ary tree on which a number of operations can be performed to manage this data structure.

Regarding the procedure for applying the black-box method to generate test cases, each subject is given the instrumented program specification without access to the source code [17, 28]. Each subject uses this specification to define the valid and invalid equivalence classes. He or she then builds test cases using the equivalence classes as a benchmark. Afterwards, the subject runs the specified test cases. This activity ends when the subject records the observed outputs of the test cases. The next activity is to reinspect the specification in order to identify possible failures revealed in the outputs of the specified test cases. The method application is complete when the subject records the observed failures.

As regards the procedure for applying the white-box method to generate test cases [27, 28], each subject is given the printed source code but not the specification of the program. Based on the source code, each subject builds the test cases trying to cover 100% of the potential program code control flow. After specifying the test cases, the subject uses an instrumented version of the program to run his or her test cases. After running the test cases, the subject records the observed outputs. Afterwards the subject uses the program specification in order to identify possible failures in the observed outputs. The testing method application is complete when the subject records the observed failures.

Table 6 shows the information on the faults revealed or not revealed by the test cases generated by the subjects participating in these experiments and summarizes the revealed and unrevealed faults by software testing method.

4. RESULTS

4.1 Analysis

As the response variable to be examined (test case quality) is dichotomous (that is, reports whether or not each of the six faults injected in the program that the test case reveals), the best strategy for its analysis is to use a logistic regression model [2, 16]. Logistic regression is useful for studying the relationship between a nominal response variable and one or more nominal or continuous predictor variables. This model predicts or estimates the natural logarithm of the odds ratio (OR) of the response variable (in this case the probability of generating test cases that reveal failures against test cases that do not reveal failures). The initial logistic regression model used in this study is introduced in Equation (2).

$$log(odds) = ln\left(\frac{\pi}{1-\pi}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2\,, \qquad (2)$$

where π is the probability of the respective result in terms of the response variable (test case quality), β values represent the regression coefficients, whereas the variables x_1 and x_2 represent the predictor variables, namely the level of CS knowledge (1st-yr ugraduate, 4th-yr ugraduate, 5th-yr ugraduate and 1st-yr graduate), as well as the testing method (black-box and white-box), respectively.

As odds ratios fit a curved regression model, the natural logarithm is applied to take advantage of the strengths of a linear regression model (in a transformed odds ratio space) such as the linear relationship between the response variable and the predictor variables. In a logistic regression model,

Table 6: Number of faults revealed or not revealed by the test cases generated by students

Exp.	CS exposure	Sw. Testing Method	Revealed Faults	Unrevealed Faults	Total
3 (2013)	1st-yr undergraduate (8%)	black-box	7	35	42
		white-box	9	39	48
4 (2014)	4th-yr undergraduate (56%)	black-box	8	22	30
		white-box	6	18	24
1 (2009)	5th-yr undergraduate (79%)	black-box	45	57	102
		white-box	47	43	90
2 (2009)	1st-yr graduate (100%)	black-box	30	42	72
		white-box	47	49	96

Table 7: Logistic regression results in terms of log odds ratio

Coefficients	Estimate	Std. Error	z value	Pr (>|z|)
(Intercept)	-0.15418	0.18875	-0.817	0.4140
1st-yr ugrad	-1.36532	0.31631	-4.316	1.59e-05
4th-yr ugrad	-0.88565	0.34786	-2.546	0.0109
5th-yr ugrad	0.08277	0.21191	0.391	0.6961
White-box	-0.02253	0.18904	-0.119	0.9051

Table 8: Logistic regression results without the testing method predictor variable (in log odds ratio)

Coefficients	Estimate	Std. Error	z value	Pr (>|z|)
(Intercept)	-0.16705	0.15484	-1.079	0.281
1st-yr ugrad	-1.36442	0.31621	-4.315	1.6e-05
4th-yr ugrad	-0.88277	0.34699	-2.544	0.011
5th-yr ugrad	0.08367	0.21177	0.395	0.693

log-transformed odds ratios have positive and negative values concentrated around the value 0. The antilogarithm is then applied to interpret the results of the model in terms of odds ratios, returning values equal to 1, greater than 1 or less than 1.

The odds ratio in this study is determined by the probabilities of generating test cases that reveal failures with respect to test cases that do not reveal failures depending on the student's level of exposure to CS knowledge and the applied testing method. Thus, the probabilities of generating test cases that reveal failures with respect to test cases that do not reveal failures may be equal (OR = 1), greater (OR > 1) or less (OR < 1). Table 7 shows the results (log-transformed odds ratios) of the logistic regression given in Equation (2). The intercept represents the reference category, in this case, graduate students applying black-box.

According to the results shown in Table 7, the coefficients represented by the level of exposure to CS knowledge (1st-yr, 4th-yr and 5th-yr ugrad) provide information that is relevant to the model, that is, **the probabilities of generating test cases that reveal failures differ significantly depending on the level of exposure to CS knowledge**. Two significant differences were found with respect to this variable. The first was between first-year undergraduate students and graduate students ($\alpha < 0.001$) and the second was between fourth-year undergraduate students and graduate students ($\alpha < 0.05$).

According to the results of this model, the testing method does not provide relevant information, and there is found to be **no significant difference generating test cases using either white-box or black-box**.

The model can be reduced by removing the variable that is not relevant to the model, in this case the testing method. The new reduced model is shown in Equation (3).

$$log(odds) = ln\left(\frac{\pi}{1-\pi}\right) = \beta_0 + \beta_1 x_1\,. \qquad (3)$$

The reduced model is then compared with the full model using the F-test, where values of p that are not significant suggest that the removed variable does not provide information that is relevant to the model, and the reduced model is therefore better. Comparing the full model with both variables (level of exposure to CS knowledge and testing method) to the reduced model that contains only the level of exposure to CS knowledge variable, the resulting p-value of 0.9051 is not significant and is therefore in favor of the reduced model. Table 8 reports the results of the logistic regression (in terms of the log-transformed odds ratios) without the testing method variable according to the reduced model described in Equation (3).

After removing the irrelevant predictor variable, we can estimate the variability in the response variable which is explained by the resulting model. Applying Nagelkerke's coefficient of determination R^2 [29], the resulting model explains 44% of the variability regarding the quality of the test cases generated according to students' level of exposure to CS knowledge. To evaluate the goodness of fit of this model, we look at whether or not the predictor residuals [16] are a better fit for an empty model (with only one intercept). The goodness of fit is evaluated by comparing the residual deviations in both models (using a χ^2 test), where a significant p-value suggests that the respective model is a much better fit than the empty model for the observations. After evaluating both models, we get a p-value < 0.001, which suggests an acceptable goodness of fit in the respective model.

4.2 Interpretation

The estimates shown in Table 8 are natural log transformed. For the interpretation of the odds ratio, we apply the antilogarithm. Table 9 shows the coefficients in terms of odds ratios and their confidence intervals.

The intercept in this model represents the reference category, that is, graduate students, meaning that the odds ratio of the coefficients (1st-yr undergraduate, 4th-yr undergraduate and 5th-yr undergraduate) is estimated with respect to the baseline category (intercept).

According to the results of Table 9, **the probabilities of generating test cases that reveal failures against test cases that do not reveal failures are less at lower lev-**

Table 9: Estimated coefficients in terms of odds ratios and their confidence intervals at 95%

Coefficients	OR	2.5%	97.5%
(Intercept)	0.84615	0.62349	1.14537
CS exposure: 1st-yr ugrad	0.25552	0.13382	0.46520
CS exposure: 4th-yr ugrad	0.41363	0.20373	0.80058
CS exposure: 5th-yr ugrad	1.08727	0.71795	1.64796

Table 10: Individual coefficients in terms of odds ratios and their confidence intervals 95%

Coefficients	OR	2.5%	97.5%
CS exposure: 1st-yr grad	0.84615	0.62349	1.14537
CS exposure: 1st-yr ugrad	0.21621	0.12157	0.36097
CS exposure: 4th-yr ugrad	0.35000	0.18386	0.62742
CS exposure: 5th-yr ugrad	0.92000	0.69242	1.22100

els of exposure to CS knowledge. As regards the level of exposure to CS knowledge, the quality of the test cases generated by first-year undergraduate students (with an exposure of 8%) decreases by a factor 0.26 than for graduate students who have a level of CS knowledge exposure of 100%. In the case of fourth-year undergraduate students (exposure of 56%), the quality of the generated test cases decreases by a factor 0.41 with respect to graduate students. In the case of fifth-year and final-year students (CS exposure of 79%), the quality of the generated test cases is equivalent (by a factor of 1.09) to the quality of the test cases generated by graduate students.

In order to determine the odds ratios individually, i.e., without comparison to the baseline category, we added the term of interest to the intercept (with the log-transformed coefficients) and then applied the antilogarithm. Table 10 shows the odds ratios of the terms and their confidence intervals without the reference category (intercept).

According to Table 10, the probabilities of generating test cases that reveal failures over test cases that do no reveal failures decrease at lower levels of exposure to CS knowledge. This trend was observed for first- and fourth-year undergraduate students. This trend does not appear to apply to fifth-year undergaduate students, who, despite having a lower level of exposure to CS knowledge (79%) than graduate students (100%), are capable of generating test cases that reveal slightly more failures, although, as shown in Table 8, this difference is not significant. Although we assume graduate students have achieved 100% of a CS program, they come from different parts of the world, on which ground we infer they have taken a range of CS programs, thus having more dispersion of the level of exposure to CS in comparison to fifth-year undergaduate students which we know the level of exposure to CS they have.

The odds ratios can also be interpreted in terms of proportions (odds ratio term divided by one plus the odds ratio term). Table 11 shows the results of converting odds ratios into proportions.

According to the results shown in Table 11, **students who have had less exposure to CS knowledge generate poorer quality test cases**. For example, the test cases generated by first-year undergraduate students reveal 18% of the failures associated with the faults injected in the instrumented program. Generally, we find that the quality of test cases increases proportionally to the level of exposure to CS knowledge.

Table 11: Individual coefficients in terms of proportions and their confidence intervals at 95%

Coefficients	Proportion	2.5%	97.5%
CS exposure: 1st-yr grad	0.45833	0.38404	0.53387
CS exposure: 1st-yr ugrad	0.17777	0.10839	0.26522
CS exposure: 4th-yr ugrad	0.25925	0.15530	0.38553
CS exposure: 5th-yr ugrad	0.47916	0.40913	0.54975

Table 12: Individual coefficients with respect to testing method in terms of proportions and their confidence intervals 95%

Coefficients	Proportion	2.5%	97.5%
Testing method: B-box	0.39316	0.33199	0.45665
Testing method: W-box	0.39629	0.33913	0.45541

As regards the testing method (the variable removed from the logistic regression model), Table 12 shows the quality of the test cases generated by undergraduate and graduate students with respect to the black-box and white-box methods (in terms of proportions).

As shown in Table 12, **black-box and white-box return equivalent values** suggesting that the test cases generated by these methods reveal about 40% of the failures caused by the faults injected in the selected instrumented program (ntree).

4.3 Threats to validity

The empirical study presented here may be subject to certain validity threats. In the following, we discuss some strategies for counteracting such threats to conclusion, internal, construct and external validity [9].

As regards conclusion validity (also known as statistical validity), the reduced model that we used exhibits an acceptable goodness of fit, and therefore the respective estimates are reliable. As regards internal validity, the same material was used in the training sessions of all the experiments. There is a difference in the length of the training in the experiments. The first two experiments were run as part of an official course related to software engineering, whereas the other two were conducted as part of a workshop. However, any possible difference between the length of the training is not considered significant, as the last two experiments were conducted as part of a workshop with a similar duration and the differences were found to be due to participant type: first- or fourth-year undergraduate students. As regards construct validity, the cause and effect constructs of all four experiments were operationalized in the same manner as in [4]. With respect to external validity, although we do not have information on the exposure to CS knowledge of second- and third-year undergraduate students, the results reported here are a useful starting point. It will be possible to refine the model used here as more undergraduate data become available.

5. DISCUSSION

According to the results of this study, the quality of the test cases differs significantly with respect to students' level exposure of CS knowledge. Compared to graduate and final-

Table 13: Results of this study with respect to the results reported in [4]

CS exposure	Test case effectiveness
1st-yr under graduate	17.78%
4th-yr under graduate	25.93%
5th-yr under graduate	47.92%
1st-yr graduate	45.83%
(Under)graduate [4]	74.59%
Practitioners [4]	62.11%

year undergraduates (in the fifth year of their degree programs), first-year and fourth-year undergraduate students who have less exposure to CS knowledge generate poorer quality test cases.

In response to the research question addressed by this study, the probabilities of generating test cases that reveal failures compared with test cases that do not reveal failures decrease by a factor of 0.21 (effectiveness is 18%) when the level of exposure to CS knowledge is 8% (first-year undergraduate students), by a factor of 0.35 (effectiveness is 26%) when the level of exposure to CS knowledge is 56% (fourth-year undergraduate students), by a factor of 0.92 (effectiveness is 48%) when the exposure to CS knowledge is 79% (fifth-year undergraduate students), and by a factor of 0.84 (effectiveness is 46%) when CS knowledge is 100% (CS program graduates).

Comparing the results of this study with the results reported in [4] (see Table 13), we find that the test cases generated in [4] were more effective than in our study. This might be due to the fact that Basili et al. [4] used several programs instrumented with a greater number of faults injected in each program (a total of 34 faults), whereas the same program injected with the same number of faults (six faults) was selected for this study. As the programs instrumented in [4] contain a greater number of faults, they are more likely to be revealed.

In particular, Basili et al. [4] report better levels of effectiveness among undergraduate and graduate students than among practitioners. We found a similar result in our study with respect to fifth-year undergraduate students and graduate students. Fifth-year undergraduate students generated test cases that were slightly more effective than the ones generated by graduate students. As discussed in [34], this difference is probably due to the fact that practitioners have different educational backgrounds, as well as a different levels of experience and so on. Likewise, the graduate students used in our study come from different parts of the world (which means that they are likely to have different levels of CS knowledge exposure) and some have professional experience, unlike the fifth-year undergraduate students whose characteristics are more alike, all or most having a similar level of knowledge and sharing the same educational and social background.

A key issue worth mentioning is that even the practitioners or graduate students (several of whom have professional experience) employed are unable to generate test cases that reveal over 90% of the failures associated with faults injected in the instrumented programs. This may be due to factors such as deficient domain knowledge regarding the software product under verification, short verification activity deadlines (pressure to release software fast), human factors, as well as insufficient software testing training.

As regards this last factor, the statistical model employed in this study has been able to attribute 44% of the total variability in the quality of the test cases generated by undergraduate and graduate students to the relationship between the level of CS knowledge and the quality of the test cases. The remaining 56% of the variability is unexplained and may be due to other factors such as the above.

6. CONCLUSIONS

This paper reported an observational study composed of four controlled experiments conducted at different times and different sites using different student subjects. This study was conducted in order to examine the possible impact of knowledge acquired from CS programs at higher education institutions on the quality of test cases generated by undergraduate and graduate students.

We used a logistic regression model to examine the data of these four experiments with respect to the number of faults revealed by test cases generated by students that applied the black-box and white-box methods. Student subjects were exposed to different levels of CS knowledge. The results suggest that there are significant differences in the quality of the test cases generated by students with respect to their level of exposure to CS knowledge.

In view of the outlined findings, educators should assure a greater level of exposure to software testing topics in courses offered as part of CS programs because of the increasing complexity of developing and maintaining software products nowadays. A possible path to do this would be to introduce software testing concepts as of basic programming courses which would be gradually expanded upon in courses offered in the final years of the degree program, that is, after students have completed most of the computing-related CS program courses. These courses could be boosted by means of the implementation of capstone projects (possibly with links to industry), applying some of the software testing methods documented in the literature, studied in academia and applied in industry.

7. ACKNOWLEDGMENTS

This study received support from the Prometeo project 20140697BP funded by the Government of the Republic of Ecuador's Department of Higher Education, Science, Technology and Innovation (Senescyt).

8. REFERENCES

[1] ACM. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science, Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM) and IEEE Computer Society.* ACM, New York, NY, USA, 2013. 999133.

[2] A. Agresti. *An introduction to categorical data analysis.* Wiley series in probability and statistics. Wiley-Interscience, Hoboken (N.J.), 2007.

[3] V. Basili and B. Perricone. Software errors and complexity: an empirical investigation. *Commun. ACM*, 27(1):42–52, 1984.

[4] V. Basili and R. Selby. Comparing the effectiveness of software testing strategies. *IEEE Trans. Softw. Eng.*, 13(12):1278–1296, 1987.

[5] B. Beizer. *Software testing techniques (2nd ed.)*. Van Nostrand Reinhold Co., New York, NY, USA, 1990.

[6] J. Brooks, F.P. No silver bullet essence and accidents of software engineering. *Computer*, 20(4):10 –19, april 1987.

[7] F. Chan and T. Tse. Software testing education and training in hong kong. In *Quality Software, 2005. (QSIC 2005). Fifth International Conference on*, pages 313–316, Sept 2005.

[8] Y. Chernak. Validating and improving test-case effectiveness. *Software, IEEE*, 18(1):81–86, Jan 2001.

[9] T. Cook and D. Campbell. *The design and conduct of quasi-experiments and true experiments in field settings*. Rand McNally, Chicago, 1976.

[10] S. U. Farooq and S. Quadri. An externally replicated experiment to evaluate software testing methods. In *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, EASE '13, pages 72–77, New York, NY, USA, 2013. ACM.

[11] V. Garousi and A. Mathur. Current state of the software testing education in north american academia and some recommendations for the new educators. In *Software Engineering Education and Training (CSEE T), 2010 23rd IEEE Conference on*, pages 89–96, March 2010.

[12] V. Garousi and J. Zhi. A survey of software testing practices in canada. *Journal of Systems and Software*, 86(5):1354 – 1376, 2013.

[13] D. Graham, E. V. Veenendaal, I. Evans, and R. Black. *Foundations of Software Testing: ISTQB Certification*. Intl Thomson Business Pr, 2008.

[14] O. S. Gómez, R. A. Aguilar, and J. P. Ucán. Efectividad de técnicas de prueba de software aplicadas por sujetos novicios de pregado. In M. D. Rodríguez, A. I. Martínez, and J. P. García, editors, *Encuentro Nacional de Ciencias de la Computación, (ENC'2014)*, Ocotlán de Morelos, Oaxaca, México, noviembre 2014. ISBN:9780990823605.

[15] B. Hetzel. *The Complete Guide to Software Testing*. QED Information Sciences, Inc., Wellesley, MA, USA, 2nd edition, 1988.

[16] D. W. Hosmer and S. Lemeshow. *Applied logistic regression*. Wiley series in probability and statistics. John Wiley & Sons, Inc. A Wiley-Interscience Publication, New York, Chichester, Weinheim, 2000.

[17] W. Howden. Functional program testing. *IEEE Transactions on Software Engineering*, 6:162–169, 1980.

[18] J. Huang. Measuring the effectiveness of a test case. In *Application-Specific Software Engineering Technology, 1998. ASSET-98. Proceedings. 1998 IEEE Workshop on*, pages 157–159, Mar 1998.

[19] IEEE. Ieee guide for software verification and validation plans. *IEEE Std 1059-1993*, pages i–87, 1994.

[20] P. C. Jorgensen. *Software Testing: A Craftsman's Approach, Third Edition*. AUERBACH, 3 edition, 2008.

[21] N. Juristo and A. M. Moreno. *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers, 2001.

[22] N. Juristo and S. Vegas. Functional testing, structural testing and code reading: What fault type do they each detect? In R. Conradi and A. Wang, editors, *Empirical Methods and Studies in Software Engineering*, volume 2765 of *Lecture Notes in Computer Science*, pages 208–232. Springer Berlin - Heidelberg, 2003.

[23] N. Juristo, S. Vegas, M. Solari, S. Abrahao, and I. Ramos. Comparing the effectiveness of equivalence partitioning, branch testing and code reading by stepwise abstraction applied by subjects. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*, pages 330–339, April 2012.

[24] E. Kamsties and C. M. Lott. An empirical evaluation of three defect-detection techniques. In *Proceedings of the 5th European Software Engineering Conference*, pages 362–383, London, UK, 1995. Springer-Verlag.

[25] H. J. Lee, L. Naish, and K. Ramamohanarao. The effectiveness of using non redundant test cases with program spectra for bug localization. In *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*, pages 127–134, Aug 2009.

[26] R. C. Linger, B. I. Witt, and H. D. Mills. *Structured Programming; Theory and Practice the Systems Programming Series*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1979.

[27] B. Marick. *The craft of software testing: subsystem testing including object-based and object-oriented testing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.

[28] G. J. Myers and C. Sandler. *The Art of Software Testing*. John Wiley & Sons, 2004.

[29] N. J. D. Nagelkerke. A note on a general definition of the coefficient of determination. *Biometrika*, 78(3):691–692, 1991.

[30] S. Ng, T. Murnane, K. Reed, D. Grant, and T. Chen. A preliminary survey on software testing practices in australia. In *Software Engineering Conference, 2004. Proceedings. 2004 Australian*, pages 116–125, 2004.

[31] M. Roper, M. Wood, and J. Miller. An empirical evaluation of defect detection techniques. *Information and Software Technology*, 39(11):763–775, 1997.

[32] P. R. Rosenbaum. *Observational Studies (Springer Series in Statistics)*. Springer, 1 edition, January 2002. ISBN-13: 978-0387989679.

[33] E. Scott, A. Zadirov, S. Feinberg, and R. Jayakody. The alignment of software testing skills of is students with industry practices–a south african perspective. *Journal of Information Technology Education*, 3:161–172, 2004.

[34] D. I. K. Sjøberg, B. Anda, E. Arisholm, T. Dybå, M. Jørgensen, A. Karahasanovic, E. F. Koren, and M. Vokác. Conducting realistic experiments in software engineering. In *ISESE '02: Proceedings of the 2002 International Symposium on Empirical Software Engineering*, page 17, Washington, DC, USA, 2002. IEEE Computer Society.

[35] A. von Mayrhauser, C. Anderson, and R. Mraz. Using a neural network to predict test case effectiveness. In

Aerospace Applications Conference, 1995. Proceedings., 1995 IEEE, number 0, pages 77–91 vol.2, Feb 1995.

[36] W. Wong, A. Bertolino, V. Debroy, A. Mathur, J. Offutt, and M. Vouk. Teaching software testing: Experiences, lessons learned and the path forward. In *Software Engineering Education and Training (CSEE T), 2011 24th IEEE-CS Conference on*, pages 530–534, May 2011.

[37] W. E. Wong. Improving the state of undergraduate software testing education. In *2012 ASEE Annual Conference*, San Antonio, Texas, June 2012. ASEE Conferences. https://peer.asee.org/21511.

[38] C. H. Yan. On effectiveness of equivalent fundamental pairs as test cases for object-oriented software. In *Systems, Man, and Cybernetics, 1999. IEEE SMC '99 Conference Proceedings. 1999 IEEE International Conference on*, volume 1, pages 933–937 vol.1, 1999.

[39] R. Yang, Z. Chen, B. Xu, W. Wong, and J. Zhang. Improve the effectiveness of test case generation on efsm via automatic path feasibility analysis. In *High-Assurance Systems Engineering (HASE), 2011 IEEE 13th International Symposium on*, pages 17–24, Nov 2011.