How Do Assertions Impact Coverage-based Test-Suite Reduction?

Junjie Chen[1,2], Yanwei Bai[1,2], Dan Hao[1,2*], Lingming Zhang[3], Lu Zhang[1,2], Bing Xie[1,2]
[1]Key Laboratory of High Confidence Software Technologies (Peking University), MoE
[2]Institute of Software, EECS, Peking University, Beijing, 100871, China
{chenjunjie,byw,haodan,zhanglucs,xiebing}@pku.edu.cn
[3]Department of Computer Science, University of Texas at Dallas, 75080, USA
lingming.zhang@utdallas.edu

***Abstract*—Code coverage is the dominant criterion in test-suite reduction. Typically, most test-suite reduction techniques repeatedly remove tests covering code that has been covered by other tests from the test suite. However, test-suite reduction based on code coverage alone may incur fault-detection capability loss, because a test detects faults if and only if its execution covers buggy code and its test oracle catches the buggy state. In other words, test oracles may also affect test-suite reduction; However, to our knowledge, their impacts have never been studied before. In this paper, we conduct the first empirical study on such impacts by using 10 real-world GitHub Java projects, and find that assertions (i.e., a typical type of test oracles) are significantly correlated with coverage-based test-suite reduction. Based on our preliminary study results, we also proposed an assertion-aware test-suite reduction technique which outperforms traditional test-suite reduction in terms of cost-effectiveness.**

I. Introduction

Regression testing is one of the most widely used and powerful ways to verify developer changes during software evolution. According to existing studies [1], [2], [3], running the entire regression test suite can incur huge execution cost. For example, our industrial collaborators at Huawei reported that executing the entire regression test suite for one of their products cost over weeks. Therefore, it is essential to keep a reasonably small (i.e., enabling faster execution) but still effective test suite (i.e., keeping high fault-detection capability). *Test-suite reduction* [1], which removes tests satisfying redundant test requirements (e.g., statement coverage) from the test suite, has been widely recognized as one of the most effective approaches to reducing the test-suite size without substantially losing the fault-detection capability.

In test-suite reduction, traditional code coverage (e.g., statement coverage) is the dominant criterion. Most test-suite reduction techniques repeatedly remove tests covering code elements that have been covered by other tests from the test suite. However, while traditional code coverage has been widely used, it alone still has widely-known limitations. That is, removing tests covering redundant code elements alone is not equivalent to removing tests detecting redundant faults, and may actually incur fault-detection capability loss. More specifically, code coverage depends on only test inputs, while test inputs alone cannot guarantee effective fault detection. According to the PIE theory [4], a fault can be detected only if the buggy code is covered by a test input and the buggy state can be caught by a test oracle. Therefore, test oracles, which are in charge of checking the correctness of the program behaviors given the test inputs, also contribute to fault detection and should be considered in test-suite reduction. Recently, a huge amount of research effort has been dedicated to study the explicit relation among code coverage, oracle coverage[1], and test effectiveness [5], [6]. However, to our knowledge, no study has ever explored how test oracles impact coverage-based test-suite reduction, although it also suffers from the neglect of test oracles to a large extent.

In this paper, we present the first empirical study that investigates the impacts of test oracles on coverage-based test-suite reduction. In particular, assertions, which are statements checking whether a boolean predicate is true at a certain program execution point, are widely used as test oracles in practice [5], [7]. Therefore, in the empirical study we focus on assertions, and perform an empirical study of four widely-used coverage-based test-suite reduction techniques on 10 real-world GitHub Java projects. Moreover, we measure the effectiveness of assertions by using assertion coverage [8] and assertion count [5], and measure the effectiveness of test-suite reduction techniques using the test-suite size reduction and the fault-detection capability loss [9], [1]. The experimental results show that **both assertion coverage and assertion count are significantly correlated with the effectiveness (including the test-suite size reduction and the fault-detection capability loss) of traditional coverage-based test-suite reduction techniques.** These findings indicate that assertion information (including assertion coverage and assertion count) may aid to improve the effectiveness of traditional test-suite reduction techniques.

II. Studied Reduction Techniques

In this section, we briefly introduce the coverage-based test-suite reduction techniques investigated in our empirical study.

*Corresponding author.

[1]Oracle coverage measures the ratio of code checked by the corresponding test oracle.

Moreover, we explain the techniques based on the statement coverage, but they are applicable to other test requirements.

A. HGS

The HGS technique is proposed by Harrold et al. [10], which reduces a test suite by identifying the tests that are essential to cover some statements, which are called essential tests. The HGS technique is a greedy algorithm, which consists of the following steps. First, this technique groups the statements into $R_1, \ldots, R_d$, where R_i $(i = 1, \ldots, d)$ represents the statements that can be satisfied by exactly i tests in the test suite. Second, this technique selects all tests covering the statements in R_1 because its statements can be covered by only these tests. Third, this technique repeatedly selects the most essential test from the test suite to cover each statement from R_2 to R_d, until all the statements are covered by the selected tests. More specifically, for each statement in R_i, this technique always selects the test that covers the maximum number of statements in R_i. If more than two tests satisfy the preceding condition, this technique breaks the tie by comparing the number of statements in R_{i+1} that are covered by each of these tests. This comparison procedure continues from R_{i+1} to R_d until one of these tests wins. If no test wins, this technique randomly selects one of these tests. The set of selected tests composes the reduced test suite.

B. GE

Chen and Lau [11] proposed the GE technique, which is a greedy algorithm adapted from the HGS technique. This technique initializes the reduced test suite as $\emptyset$, and selects all the tests each of which is the only one test that covers some statements. Then this technique continues selecting the test that covers the maximum number of statements that are not covered by already selected tests until all the statements are covered by the set of selected tests. More specifically, during each selection, this technique first selects a test t satisfying $\forall t' \in (T - T'), |S[t']| \leq |S[t]|$, where T and T' represent the original test suite and the reduced test suite respectively, and $S[t]$ and $S[t']$ represent the set of test requirements satisfied by t and t' respectively, then this technique puts t into the reduced test suite and removes $S[t]$ from the set of test requirements satisfied by the unselected tests. In particular, the termination condition of the selections is $S[T'] = S[T]$.

C. GRE

Chen and Lau [12] proposed the GRE technique, which mixes three strategies including the greedy strategy, the essential strategy, and the 1-to-1 redundancy strategy. The greedy strategy selects the tests covering the maximum number of statements that are not covered by already selected tests, the essential strategy selects all essential tests, and the 1-to-1 redundancy strategy removes all the 1-to-1 redundant tests. A test is 1-to-1 redundant if there exists another test that can cover all the statements the former covers. More specifically, the GRE heuristic first removes all the 1-to-1 redundant tests, and then performs the GE technique, i.e., the essential strategy and the greedy strategy, on the reduced test suite.

D. ILP

Black et al. [13] proposed to model test-suite reduction through integer linear programming (ILP), whose objective is to minimize the number of selected tests. The ILP model for test-suite reduction is represented by Formula (1), where x_j represents whether the jth test in the test suite is selected, and a_{ij} represents whether the jth test in the test suite covers the ith statement. The objective function of this ILP model is to minimize the number of selected tests by satisfying that each statement is covered by at least one selected test.

$$\begin{aligned} &\text{Minimize} \quad \sum_{j=1}^{n} x_j, x_j \in [0,1] \\ &\text{Subject to} \quad \bigwedge_{i=1}^{m} (\sum_{j=1}^{n} a_{ij} x_j \geq 1), a_{ij} \in [0,1] \end{aligned} \tag{1}$$

Besides this single-objective ILP-based technique, Black et al. [13] also proposed another multi-objective ILP-based technique, which balances the objective of finding the minimal set of tests and the objective of finding the most valuable tests on detecting faults. As this paper investigates only coverage-based techniques, we consider only the single-objective ILP-based technique in the empirical study.

III. Empirical Study

In this section, we present the first empirical study investigating the impacts of assertions on coverage-based test-suite reduction. We use two widely-used measurements to represent the strength of test oracles, assertion coverage and assertion count [5]. In summary, this empirical study investigates the following research questions.

- **RQ1**: How does assertion coverage impact coverage-based test-suite reduction?
- **RQ2**: How does assertion count impact coverage-based test-suite reduction?

A. Implementation and Supporting Tools

Following prior work [5], we collect two types of code coverage with Clover[2], and collect assertion coverage with JavaSlicer[3]. To implement the ILP-based reduction technique, we use a mathematical programming solver, GUROBI Optimization[4], which is used to represent and solve the equations formulated by the ILP-based technique. Besides, we use the PIT mutation testing tool[5] to generate mutation faults. All the reduction techniques are implemented as Python programs, which are available at our project website[6].

B. Subjects, Tests and Faults

In our study, we use 10 real-world GitHub Java projects. Table I shows their basic information. In this table, Columns "PSLOC" and "TSLOC" present the lines of product code and test code respectively, which are measured by SLOCCount[7],

[2]https://www.atlassian.com/software/clover.
[3]We remove the tests that JavaSlicer fails to generate backward slices.
[4]http://www.gurobi.com/.
[5]http://pitest.org/.
[6]https://github.com/JunjieChen/ReductionAssertion.
[7]http://www.dwheeler.com/sloccount/.

TABLE I: Subjects

ID	Subjects	PSLOC	TSLOC	#T	#A	Acov(%)
1	Confucius	503	700	84	136	32.01
2	cors-filter	588	1759	72	136	34.01
3	cucumber-reporting	2245	1255	124	175	16.39
4	exp4j	1086	3531	284	633	34.25
5	jackson-core	18715	9880	346	1321	19.86
6	joss	7972	6029	531	548	20.67
7	jsoup	10420	4798	466	1388	22.36
8	lambdaj	3634	4885	265	741	27.66
9	raml-java-parser	8696	2976	192	655	18.54
10	tamper	4330	2768	62	419	30.76
	Total/Avg.	58189	38581	2426	6152	25.65

Columns "#T" and "#A" present the number of tests and assertions used in the study, Column "Acov" presents the assertion coverage, and the last row presents the total/average number of these information for all subjects.

Previous studies [14], [15] show that mutation faults are close to real faults and are suitable for software testing experimentation, since real faults are usually hard to collect and small in number, making it hard to perform statistical analysis. In particular, mutation faults have also been widely used in test-suite reduction research [9], [16]. Therefore, in our study we use mutation faults to investigate the effectiveness of test-suite reduction techniques.

C. Independent Variables

We consider the following independent variables.

1) Assertion Information: Following prior studies on assertions [5], we measure the strength of assertions by two measurements: (1) assertion coverage (also called checked coverage [8]), i.e., the percentage of statements checked by test assertions in a test suite, and (2) assertion count [5], i.e., the number of assertions (e.g., assertEqual) in a test suite.

2) Techniques: We consider four coverage-based reduction techniques in our study, including (1) the HGS technique (abbreviated as HGS), (2) the GE technique (abbreviated as GE), (3) the GRE technique (abbreviated as GRE), and (4) the ILP-based technique (abbreviated as ILP). More details on these techniques are referred to Section II.

3) Code coverage: For each reduction technique, we consider two widely-used code coverage criteria, i.e., statement coverage and method coverage.

D. Dependent Variables

We consider two dependent variables in the study, including the test-suite size reduction and the fault-detection capability loss, which are widely used to measure the effectiveness of reduction techniques [9], [1]. More specifically, we use Formulae (2) and (3) to calculate the test-suite size reduction and the fault-detection capability loss, respectively. In the two formulae, T and T' are the original test suite and the reduced test suite respectively, and $\#Fault(T)$ and $\#Fault(T')$ represent the number of faults detected by T and T' respectively.

$$Size\_Reduction = \frac{|T| - |T'|}{|T|} \tag{2}$$

$$Loss = \frac{\#Faults(T) - \#Faults(T')}{\#Faults(T)} \tag{3}$$

E. Procedure

First, we construct faults for each subject, and use all the mutation faults for each subject following the existing work on test-suite reduction [17].

Second, we construct 1,000 test suites for each subject, recording their assertion coverage and assertion count. More specifically, we randomly select a random number of tests from the original test suite T to construct a test suite with the maximum size of $|T|$.

Third, for each constructed test suite, we collect its statement coverage and method coverage on the subject without any mutation faults.

Fourth, for each constructed test suite, we apply four reduction techniques introduced in Section II by using statement coverage and method coverage, recording the reduced test suite and their assertion coverage and assertion count. For each reduction technique we calculate the test-suite size reduction, the fault-detection capability loss, and the assertion coverage reduction and the assertion count reduction of each test suite.

F. Threats to Validity

The threat to internal validity mainly lies in the implementations of test-suite reduction techniques and the scripts used in our experiments. To reduce this threat, the first two authors of this paper reviewed the source code.

The threats to external validity mainly lie in subjects and faults. In our study, we use 10 real-world GitHub Java projects as subjects, although these subjects may not be sufficiently representative for other projects. Besides, we use mutation faults instead of real faults. In the future, we will reduce these threats by using more projects in other programming languages with real faults.

The threat to construct validity mainly lies in the measurement for test-suite reduction. In the study, we use the widely-recognized test-suite size reduction and fault-detection capability loss to measure the effectiveness of test-suite reduction without considering test costs [18], [19], [20]. In the future, we will repeat the study by considering this issue.

G. Results and Analysis

1) RQ1: Impacts of Assertion Coverage: We analyze the impacts of assertion coverage on test-suite reduction through correlation analysis. More specifically, we calculate the Spearman correlation between the assertion coverage reduction and the effectiveness (including the test-suite size reduction and the fault-detection capability loss) of coverage-based test-suite reduction techniques, because Spearman correlation coefficient is widely used to assess how well the relationship between two variables can be described using a monotonic function, whose results are shown in Table II. Due to space limit, we cannot present the analysis results of four test-suite reduction techniques with two types of code coverage. Here we present the analysis results of the ILP technique with statement/method coverage because different reduction techniques have quite similar results. The complete results of the study are all available at our project website.

TABLE II: Spearman correlation between the assertion coverage reduction and the effectiveness of the coverage-based ILP reduction technique

ID	Test-suite Size Reduction		Fault-detection Capability Loss	
	statement	method	statement	method
1	0.873(*)	0.873(*)	0.613(*)	0.642(*)
2	0.825(*)	0.852(*)	0.395(*)	0.496(*)
3	0.724(*)	0.704(*)	0.449(*)	0.531(*)
4	0.885(*)	0.868(*)	0.266(*)	0.675(*)
5	0.821(*)	0.873(*)	0.721(*)	0.856(*)
6	0.899(*)	0.914(*)	0.884(*)	0.905(*)
7	0.915(*)	0.954(*)	0.789(*)	0.933(*)
8	0.871(*)	0.875(*)	0.516(*)	0.605(*)
9	0.692(*)	0.856(*)	0.425(*)	0.704(*)
10	0.766(*)	0.813(*)	0.738(*)	0.755(*)

TABLE III: Summary on the strength of correlations between the assertion coverage reduction and the effectiveness of coverage-based reduction techniques

Tech.	Strength	Size Reduction		Fault-detection Capability Loss	
		stmtement	method	stmtement	method
HGS	✓✓	9	8	2	3
	✓	1	2	2	2
	❍	0	0	4	5
	✗	0	0	1	0
	✗✗	0	0	1	0
GE	✓✓	9	9	2	3
	✓	1	1	3	3
	❍	0	0	2	2
	✗	0	0	3	2
	✗✗	0	0	0	0
GRE	✓✓	9	9	2	3
	✓	1	1	2	3
	❍	0	0	3	3
	✗	0	0	3	1
	✗✗	0	0	0	0
ILP	✓✓	9	9	2	3
	✓	1	1	2	2
	❍	0	0	3	5
	✗	0	0	3	0
	✗✗	0	0	0	0

In this table, Columns 1 presents subject IDs, Columns 2-3 present the correlation between the assertion coverage reduction and the test-suite size reduction, Columns 4-5 present the correlation between the assertion coverage reduction and the fault-detection capability loss. Besides, the value in each cell represents the Spearman correlation coefficient, where the positive value represents positive correlation and the negative value represents negative correlation. Moreover, we use (*) to show the correlation with statistical significance (the significant level α is set to 0.05).

From Table II, all the correlations between the assertion coverage reduction and the effectiveness (including the test-suite size reduction and the fault-detection capability loss) of the ILP technique are statistically significant. That is, the assertion coverage reduction has significant impacts on coverage-based test-suite reduction, including on its test-suite size reduction and fault-detection capability loss. Furthermore, all the correlation coefficients of both measurements are positive. That is, when the assertion coverage reduction increases, both the test-suite size reduction and the fault-detection capability loss become larger.

Besides, based on the correlation coefficients, the correlations are classified into extremely strong correlations ($0.8 \sim 1$) (abbreviated ✓✓), strong correlations ($0.6 \sim 0.8$) (abbreviated ✓), moderate correlations ($0.4 \sim 0.6$) (abbreviated ❍), weak correlations ($0.2 \sim 0.4$) (abbreviated ✗), and extremely weak correlations ($0 \sim 0.2$) (abbreviated ✗✗) [21]. Therefore, we further summarize the strength of correlations based on this classification, whose results are shown in Table III. In this table, Column 1 presents four reduction techniques, Column 2 presents the strength of correlations, Columns 3-4 present the number of subjects with the corresponding correlation strength on the test-suite size reduction, and Columns 5-6 present the number of subjects with the corresponding correlation strength on the fault-detection capability loss.

From Table III, the correlations on the test-suite size reduction are always extremely strong and strong for all code coverage and all reduction techniques. Besides, the correlations on the fault-detection capability loss are mostly extremely strong, strong and moderate. That is, the assertion coverage reduction tends to have strong correlations with both the size reduction and the fault-detection capability loss of coverage-based reduction techniques, but the former is much stronger than the latter.

2) RQ2: Impacts of Assertion Count: We also analyze the impacts of assertion count on test-suite reduction through Spearman correlation analysis between the assertion count reduction and the effectiveness (including the test-suite size reduction and the fault-detection capability loss) of coverage-based reduction techniques, and also summarize the strength of correlations based on the above mentioned classification. The correlation analysis overall has the same trend with Section III-G1, and the detailed results can be found in our project website. Actually, the observation that the impacts of assertion coverage and assertion count are similar is as expected, because assertion coverage and assertion count also have correlations. That is, when assertion coverage is reduced after reduction, assertion count tends to be reduced too.

3) Summary: Based on the analysis, we have the following major findings:

- Both assertion coverage and assertion count are significantly correlated with coverage-based test-suite reduction.
- When the assertion coverage/count reduction increases, the coverage-based reduction techniques tend to have larger test-suite size reduction and more fault-detection capability loss.
- Both the correlations between the assertion coverage/-count reduction and the size reduction and the correlations between the assertion coverage/count reduction and the fault-detection capability loss are mostly strong.

IV. Discussion

Based on Section III, assertions have significant correlations with coverage-based test-suite reduction. When the assertion coverage/count reduction increases, both the test-suite size reduction and the fault-detection capability loss become larger. Note that since code coverage are preserved after reduction, the assertion difference may be the major reason for the result difference (including the differences on the size reduction and the fault-detection capability loss). That indicates when reducing a test suite, it is necessary to consider assertions. In

this section, we make the first attempt to combine assertions with statement coverage in the ILP reduction technique by forcing the reduced test suite to preserve both statement coverage and assertion coverage, and further evaluate whether such combination works. In particular, in order to distinguish with the traditional statement-coverage-based ILP technique, we call the ILP technique based on the combinational test requirements as the *assertion-aware* ILP technique.

Based on our assertion-aware ILP technique, we conduct a preliminary study to evaluate the effectiveness of such combination. In particular, we use the similar experimental setup as Section III. The only difference with the setup in Section III is that in this study we use the original test suite to reduce. Table IV shows the comparison results between the traditional statement-coverage-based ILP technique and our assertion-aware ILP technique. In this table, Columns 2-3 present the comparison results on the test-suite size reduction, Columns 4-5 present the comparison results on the fault-detection capability loss. From this table, for almost all the subjects, the fault-detection capability loss of our assertion-aware ILP technique is smaller than that of the statement-coverage-based ILP technique, and the test-suite size reduction of the former is also smaller than that of the latter. That is, our assertion-aware ILP technique performs better than the statement-coverage-based ILP technique in terms of the fault-detection capability loss but performs worse than the latter in terms of the test-suite size reduction. As the ultimate goal of software testing is to detect faults, developers would like to sacrifice test-suite size (i.e., small test-suite size reduction) rather than fault detection (i.e., more fault-detection capability loss). From this perspective, our assertion-aware ILP technique is promising, even compared with the traditional statement-coverage-based ILP technique.

$$density = \frac{\#Faults(T) - \#Faults(T')}{|T| - |T'|} \quad (4)$$

Our assertion-aware ILP technique actually provides a trade-off between the fault-detection capability loss and the test-suite size reduction. To investigate whether such a trade-off is worthwhile, we calculate the density of the fault-detection capability loss as Formula 4. In this formula, $\#Faults(T) - \#Faults(T')$ represents the number of faults that become undetected after reduction; $|T| - |T'|$ represents the reduced number of tests after reduction. This density measures the fault-detection capability loss when a test is reduced, which can be used to measure the overall cost-effectiveness of a test reduction technique by considering both the fault-detection capability loss and the test-suite size reduction. In particular, the smaller the density is, the more effective the corresponding reduction technique is.

Columns 6-7 in Table IV present the detailed comparison results between the traditional statement-coverage-based ILP technique and our assertion-aware ILP technique in terms of the density. We can find that our assertion-aware ILP technique performs better than the traditional statement-coverage-based ILP techniques in terms of the density on most subjects. That is, although our assertion-aware ILP technique sacrifices the test-suite size reduction to decrease the fault-detection capability loss, the sacrifice is worthwhile because when reducing a test from a test suite, the number of faults that become undetected is much smaller compared with coverage-based ILP technique.

TABLE IV: Comparison between the statement-coverage-based ILP technique and the assertion-aware ILP technique

ID	Size Reduction		Loss		Density	
	stmt	stmt+assert	stmt	stmt+assert	stmt	stmt+assert
1	0.643	0.548	0.043	0.045	0.463	0.565
2	0.597	0.500	0.059	0.040	0.581	0.472
3	0.863	0.371	0.201	0.074	3.224	2.761
4	0.838	0.778	0.078	0.043	0.445	0.267
5	0.402	0.327	0.026	0.019	2.489	2.257
6	0.652	0.559	0.087	0.047	1.098	0.697
7	0.592	0.511	0.043	0.023	1.283	0.811
8	0.600	0.532	0.044	0.030	0.937	0.716
9	0.656	0.599	0.019	0.022	0.683	0.861
10	0.516	0.419	0.018	0.016	1.438	1.538
Avg.	0.636	0.514	0.062	0.036	1.264	1.095

In summary, our combination indeed improves the traditional statement-coverage-based ILP technique in terms of cost-effectiveness. This conclusion also confirms the impacts of assertions on coverage-based test-suite reduction. In particular, this conclusion indicates the promising future on combining assertions with code coverage in test-suite reduction.

V. Related Work

A. Test-Suite Reduction

As prior work [9], we also classify the existing work on test-suite reduction into two categories.

The first category is algorithms to reduce the size of a test suite. Besides the studied techniques in our empirical study, there are still a lot of work focusing on reducing the size of a test suite. Yoo and Harman [22] viewed this problem as a multi-objective optimization problem, whose objectives contain coverage, cost and fault history, and applied a hybrid algorithm that combines the greedy technique and a multi-objective evolutionary algorithm. Hao et al. [9] proposed an on-demand test-suite reduction approach, in order to allow engineers to set specific upper limits on fault-detection capability loss. Gotlieb and Marijan [23] proposed FLOWER, a new approach to test-suite reduction based on a search among network maximum flows. Blue et al. [24] proposed a interaction-based test suite reduction to complement standard combinatorial test design. Zhang et al. [25] proposed specialized test-suite reduction techniques to speed up mutation testing.

The second category is criteria used in test-suite reduction. Jones and Harrold [26] proposed to use the modified condition/decision coverage. Lin et al. [27] proposed to utilize multiple coverage criteria (e.g., branch coverage and definition-use pair coverage) to increase fault-detection effectiveness. Hsu and Orso [28] proposed a general framework and tool for supporting test-suite reduction based on any number of criteria by leveraging ILP solvers. Shi et al. [17] proposed to utilize killed mutants to reduce test suites in software evolution.

Different from the above previous work, our work conducts the first empirical study to explore the impacts of assertions on coverage-based test-suite reduction.

B. Empirical Studies on Test Oracles

Our work is also related to the empirical studies on test oracles. In the literature, a huge amount of research efforts have been dedicated to empirical studies of test oracles, including the test oracle problem for GUI [29], oracle strategies for model-based testing [30], correctness/understandability of Daikon invariants when used as test oracles [31], [32], the effectiveness of metamorphic testing to alleviate the oracle problem [33], the cost/effectiveness of automated oracles [34], the effectiveness of crowd oracles [35], the comparison of internal oracles on fault-detection capability [36], the usage of manual assert [37], the relationship between software assertions and faults [38], and the impact of assertion quantity and coverage on test effectiveness [5]. However, none of the existing work investigates the impacts of assertions on coverage-based test-suite reduction, which is target of this paper.

VI. Conclusions

In this paper, we conduct the first empirical study to explore the impacts of assertions on coverage-based test-suite reduction. Our results demonstrate that assertions are significantly correlated with coverage-based test-suite reduction. In the future, we will further investigate how to combine assertions and code coverage for test suite reduction and various other software testing problems.

VII. Acknowledgment

This work is partially supported by the National Key Research and Development Program of China No. 2016YFB1000801, and the National Natural Science Foundation of China under Grant No. 61421091, 61522201, 61611130210. This work is also partially supported by NSF Grant No. CCF-1566589, Google Faculty Research Award, and UT Dallas start-up fund.

References

[1] G. Rothermel, M. J. Harrold, J. Von Ronne, and C. Hong, "Empirical studies of test-suite reduction," *STVR*, vol. 12, no. 4, pp. 219–249, 2002.

[2] J. Chen, W. Hu, D. Hao, Y. Xiong, H. Zhang, L. Zhang, and B. Xie, "An empirical comparison of compiler testing techniques," in *ICSE*, 2016, pp. 180–190.

[3] Q. Gao, J. Li, Y. Xiong, D. Hao, X. Xiao, K. Taneja, L. Zhang, and T. Xie, "High-confidence software evolution," *SCI CHINA INFORM SCI*, vol. 59(7), pp. 071 101:1–071 101:19, 2016.

[4] J. M. Voas, "Pie: A dynamic failure-based technique," *TSE*, vol. 18, no. 8, pp. 717–727, 1992.

[5] Y. Zhang and A. Mesbah, "Assertions are strongly correlated with test suite effectiveness," in *FSE*, 2015, pp. 214–224.

[6] L. Inozemtseva and R. Holmes, "Coverage is not strongly correlated with test suite effectiveness," in *ICSE*, 2014, pp. 435–445.

[7] J. Chen, Y. Bai, D. Hao, L. Zhang, L. Zhang, B. Xie, and H. Mei, "Supporting oracle construction via static analysis," in *ASE*, 2016, pp. 178–189.

[8] D. Schuler and A. Zeller, "Assessing oracle quality with checked coverage," in *ICST*, 2011, pp. 90–99.

[9] D. Hao, L. Zhang, X. Wu, H. Mei, and G. Rothermel, "On-demand test suite reduction," in *ICSE*, 2012, pp. 738–748.

[10] M. J. Harrold, R. Gupta, and M. L. Soffa, "A methodology for controlling the size of a test suite," *TOSEM*, vol. 2, no. 3, pp. 270–285, 1993.

[11] T. Chen and M. Lau, "Heuristics towards the optimization of the size of a test suite," in *ICSQM*, vol. 2, 1995, pp. 415–424.

[12] T. Y. Chen and M. F. Lau, "A new heuristic for test suite reduction," *IST*, vol. 40, no. 5, pp. 347–354, 1998.

[13] J. Black, E. Melachrinoudis, and D. Kaeli, "Bi-criteria models for all-uses test suite reduction," in *ICSE*, 2004, pp. 106–115.

[14] J. H. Andrews, L. C. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?" in *ICSE*, 2005, pp. 402–411.

[15] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser, "Are mutants a valid substitute for real faults in software testing?" in *FSE*, 2014, pp. 654–665.

[16] L. Zhang, D. Marinov, L. Zhang, and S. Khurshid, "An empirical study of junit test-suite reduction," in *ISSRE*, 2011, pp. 170–179.

[17] A. Shi, A. Gyori, M. Gligoric, A. Zaytsev, and D. Marinov, "Balancing trade-offs in test-suite reduction," in *FSE*, 2014, pp. 246–256.

[18] J. Chen, Y. Bai, D. Hao, Y. Xiong, H. Zhang, L. Zhang, and B. Xie, "Test case prioritization for compilers: A text-vector based approach," in *ICST*, 2016, pp. 266–277.

[19] J. Chen, Y. Bai, D. Hao, Y. Xiong, H. Zhang, and B. Xie, "Learning to prioritize test programs for compiler testing," in *ICSE*, 2017, to appear.

[20] D. Hao, L. Zhang, and H. Mei, "Test-case prioritization: achievements and challenges," *FCS*, vol. 10(5), pp. 769–777, 2016.

[21] T. D. V. Swinscow, M. J. Campbell *et al.*, *Statistics at square one*. Bmj London, 2002.

[22] S. Yoo and M. Harman, "Using hybrid algorithm for pareto efficient multi-objective test suite minimisation," *JSS*, vol. 83, no. 4, pp. 689–701, 2010.

[23] A. Gotlieb and D. Marijan, "Flower: optimal test suite reduction as a network maximum flow," in *ISSTA*, 2014, pp. 171–180.

[24] D. Blue, I. Segall, R. Tzoref-Brill, and A. Zlotnick, "Interaction-based test-suite minimization," in *ICSE*, 2013, pp. 182–191.

[25] L. Zhang, D. Marinov, and S. Khurshid, "Faster mutation testing inspired by test prioritization and reduction," in *ISSTA*, 2013, pp. 235–245.

[26] J. A. Jones and M. J. Harrold, "Test-suite reduction and prioritization for modified condition/decision coverage," *TSE*, vol. 29, no. 3, pp. 195–209, 2003.

[27] J.-W. Lin and C.-Y. Huang, "Analysis of test suite reduction with enhanced tie-breaking techniques," *IST*, vol. 51, no. 4, pp. 679–690, 2009.

[28] H.-Y. Hsu and A. Orso, "Mints: A general framework and tool for supporting test-suite minimization," in *ICSE*, 2009, pp. 419–429.

[29] Q. Xie and A. M. Memon, "Designing and comparing automated test oracles for gui-based software applications," *TOSEM*, vol. 16, no. 1, p. 4, 2007.

[30] N. Li and J. Offutt, "An empirical analysis of test oracle strategies for model-based testing," in *ICST*, 2014, pp. 363–372.

[31] L. Zhang, G. Yang, N. Rungta, S. Person, and S. Khurshid, "Feedback-driven dynamic invariant discovery," in *ISSTA*, 2014, pp. 362–372.

[32] M. Staats, S. Hong, M. Kim, and G. Rothermel, "Understanding user understanding: determining correctness of generated program invariants," in *ISSTA*, 2012, pp. 188–198.

[33] H. Liu, F.-C. Kuo, D. Towey, and T. Y. Chen, "How effectively does metamorphic testing alleviate the oracle problem?" *TSE*, vol. 40, no. 1, pp. 4–22, 2014.

[34] C. D. Nguyen, A. Marchetto, and P. Tonella, "Automated oracles: An empirical study on cost and effectiveness," in *FSE*, 2013, pp. 136–146.

[35] F. Pastore, L. Mariani, and G. Fraser, "Crowdoracles: Can the crowd solve the oracle problem?" in *ICST*, 2013, pp. 342–351.

[36] T. Yu, W. Srisa-an, and G. Rothermel, "An empirical comparison of the fault-detection capabilities of internal oracles," in *ISSRE*, 2013, pp. 11–20.

[37] C. Casalnuovo, P. Devanbu, A. Oliveira, V. Filkov, and B. Ray, "Assert use in github projects," in *ICSE*, 2015, pp. 755–766.

[38] G. Kudrjavets, N. Nagappan, and T. Ball, "Assessing the relationship between software assertions and faults: An empirical investigation," in *ISSRE*, 2006, pp. 204–212.