BLACK-BOX TEST CASE SELECTION BY RELATING CODE CHANGES WITH PREVIOUSLY FIXED DEFECTS

A Thesis

by

Tutku Çıngıl

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in the
Department of Computer Science

Özyeğin University
May 2022

BLACK-BOX TEST CASE SELECTION BY RELATING CODE CHANGES WITH PREVIOUSLY FIXED DEFECTS

Approved by:

Assoc. Prof. Dr. Hasan Sözer, Advisor
Department of Computer Science
Ozyegin University

Assist. Prof. Dr. Reyhan Aydoğan
Department of Computer Science
Ozyegin University

Assoc. Prof. Dr. Tolga Ovatman
Department of Computer Engineering
Istanbul Technical University

Date Approved: 24 May 2022

For my wife Tuğba, who has been a constant source of support and encouragement during the challenges of graduate school and my son Tuna, who recently joined us.

ABSTRACT

Software continuously changes to address new requirements and to fix defects. Regression testing is performed to ensure that the applied changes do not adversely affect existing functionality. The increasing number of test cases makes it infeasible to execute the whole regression test suite. Test case selection is adopted to select a subset of the test suite, which is associated with the changed parts of the software. These parts are assumed to be error-prone. We present and evaluate a test case selection approach in the context of black-box regression testing of embedded systems. In this context, it is challenging to relate test cases with a set of distinct source code elements to be able to select those test cases associated with the modified parts of the source code. We analyze previously fixed defects for this purpose. We relate test cases that detect these defects with the source files that are previously modified for fixing them. Then, we select test cases related with source code files that are modified in the subsequent revision. The strength of this relation is determined as the number of changes associated with fixed defects previously detected by the same test cases. We conduct a case study on 3 real projects from the consumer electronics domain. Results show that it is possible to detect from 65% up to 85% of the defects detected by the whole test suite by selecting from 30% up to 70% of the test cases.

ÖZETÇE

Yazılım, yeni gereksinimleri karşılamak ve kusurları gidermek için sürekli olarak değişmektedir. Regresyon testleri, uygulanan değişikliklerin mevcut işlevselliği olumsuz etkilemediğinden emin olmak için yapılmaktadır. Artan test senaryoları, regresyon testlerinin tamamının yapılmasını elverişsiz hale getirmektedir. Test senaryosu seçimi ile, yazılımın değişen parçaları ile ilişkili olan test senaryolarının seçilmesi benimsenmektedir. Bu tezde, regresyon testlerinde kara kutu test yaklaşımı ile gömülü sistemler üzerinde test senaryo seçimi için bir yaklaşım sunulmaktadır. Bu kapsamda, yazılımdaki değişen kaynak kodları ile test senaryoları arasında bir ilişki olduğunu ve bu değişen kaynak kodları ile test senaryoları seçmenin mümkün olduğu iddia edilmektedir. Hataları tespit eden test senaryoları, bunları düzeltmek için önceden değiştirilmiş kaynak dosyaları ile ilişkilendirilmektedir. Ardından, bir sonraki yazılımda değiştirilen kaynak kod dosyaları ile ilgili test senaryoları seçilmektedir. Ayrıca, değişen dosya birden fazla hata ile ilgili olabileceği için, değişen kod ve test senaryosu arasındaki ilişki her bir test senaryosu için farklı olarak tanımlanmaktadır. Bu tezde tüketici elektroniği alanında faaliyet gösteren 3 gerçek proje üzerinde çalışılmaktadır. Sonuçlar, test senaryolarının %30'u ile %70'i arasında seçim yaparak tüm test senaryoları tarafından tespit edilen hataların %65'inden %85'ine kadar tespit edilmesinin mümkün olduğunu göstermektedir.

ACKNOWLEDGEMENTS

I would first like to thank my thesis advisor who made this work possible and valuable comments whenever I ran into a trouble spot or had a question about my research or writing. Without his assistance and dedicated involvement in every step throughout the process, this thesis would have never been accomplished.

I would also like to thank software developers at Vestel for sharing their code base and JIRA records with us and supporting our case study. Without their passionate participation and input, the validation survey could not have been successfully conducted.

I would also like to acknowledge Assist. Prof. Dr. Reyhan Aydoğan of the Department of Computer Science at Ozyegin University as the second reader and Assoc. Prof. Dr. Tolga Ovatman of the Department of Computer Science at Istanbul Technical University as the third reader of this thesis, and I am gratefully indebted to their very valuable comments on this thesis.

Finally, I could not have completed this thesis without the support of my wife, Tuğba, whom without her tremendous understanding and encouragement in the past few years, it would be impossible for me to complete my study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Acronyms

CF Changed File. i, 11, 14, 15

PBD Percentage of Bugs Detected. i, 16, 18, 20, 22, 24, 27, 28, 29, 30

PSTC Percentage of Selected Test Cases. i, 16, 20, 27, 28, 29, 30

RFE Ratio of Field Errors. i, 30

STCD Selected Test Case Duration. i, 28, 29, 30

TC Test Case. i, viii, 12, 13, 14, 18, 20, 22, 24

TS Test Step. i, 36

CHAPTER I

INTRODUCTION

Software systems evolve as a result of maintenance activities to fix defects as well as to address changing requirements and technology. Regression testing process ensures that these maintenance activities do not adversely affect existing functionality [1]. This process involves the (re)execution of a set of test cases taking part in the regression test suite. Increasing size of regression test suites and limited resources make it infeasible to execute the whole test suite [2]. The widespread adoption of continuous integration and increased frequency of software builds amplify the cost of regression testing as well. Test case selection, test case prioritization and test suite minimization approaches have been proposed to reduce this cost [1]. In this thesis, we focus on test case selection [3], where we select a subset of test cases within the regression test suite. This subset is associated with the changed parts of the software that are assumed to be error-prone.

Test case selection requires a mapping between the test cases and the source code so that one can make a selection of test cases based on source code modifications. The mapping can be at various granularity levels such as statements, methods, files and modules [4]. However, it is harder to perform this mapping for black-box test cases that are developed and applied from the user point of view. There exists a significant gap between these high-level specifications of user interaction and the low level implementation. One might measure code coverage during the execution of test cases to associate them with source code elements. However, it might not be possible or trivial to collect these measurements for all types of systems such as embedded systems. Moreover, it might not be meaningful to rely on coverage measures even if they are available. There might be certain parts of the source code such as the entry points in the control flow that are involved in

the execution of every test case. As a result, it is challenging to relate black-box tests with a set of distinct source code elements.

In this work, we analyze previously fixed defects for black-box test case selection. We relate test cases that detect these defects with the source files that are modified for fixing the defects. Then, we select test cases related with source code files that are modified in the subsequent revision. There have been studies [5, 6] that propose and evaluate the same method for establishing a relation between test cases and source code files. However, these studies considered a binary relation only. In our study, we analyze the trade-off when the relation strength is considered in terms of the number of changes in source files associated with previously fixed defects. We conduct an industrial case study by collecting data from 8 versions of 3 real projects from the consumer electronics domain. Our dataset includes 482 defects and 38 test cases. Results show that it is possible to detect from 65% up to 85% of the defects detected by the whole test suite by selecting from 30% up to 70% of the test cases. We also observe that a threshold value set regarding the relation strength is useful in balancing the trade-off between cost and effectiveness of the test suite in case of limited resources.

The rest of this thesis is organized as follows. We provide a summary of the related studies in the following section. We explain our approach in Section 3. We present an industrial case study and discuss the results in Section 4. Finally, we conclude the thesis in Section 5.

CHAPTER II

RELATED WORK

Test case selection has been studied for more than three decades [4]. Many solution approaches have been introduced and evaluated for this problem. Some of these approaches [7, 8] are based on similarity analysis performed among the test cases. The idea is to employ various techniques to identify [8] and cluster [7] test cases so that the execution of similar test cases can be avoided and diversification among the selected test cases is maximized. On the other hand, most of the approaches are based on white-box testing and code analysis to evaluate the impact of configuration [9] or code [10] changes and select a subset of the test cases that can best capture the corresponding potential side-effects.

One of the earlier studies [11] perform a mapping between test cases and implementation based on control flow graphs derived from the source code. Control flow graphs are constructed both for the current and the previous version of the program. The differences between these graphs are captured. Then, those test cases are selected, which cover the modified elements of the graph.

Graves et al. presented an empirical study [12] to evaluate and compare the costs and benefits of alternative test selection strategies, including namely *minimization*, *dataflow*, *safe*, *random* and *re-test all*. The simplest strategy is obviously *re-test all*, where the whole test suite is executed. This is the most comprehensive one but it turns out to be prohibitively expensive. Another simple strategy involves *random* selection of test cases. Results of the empirical study suggest that the effectiveness of this strategy grows as the selection ratio grows but the rate of growth decreases [12]. Results also showed that *minimization* techniques select the smallest but the least effective test suites at the same time. The other two strategies, *safe* and *dataflow* turn out to be similar in cost effectiveness and they

find the same types of defects for the same selection ratios. All of these strategies except *random* and *re-test all* rely on coverage measurements unlike our approach. We used *re-test all* strategy in our study as the baseline.

Elbaum et al. presented a multiple case study [13] to evaluate alternative test selection strategies, including namely *re-test all*, *modified non-core-function* [14], *modification-focused minimization* [15], and *coverage-focused minimization* [16]. These are all coverage-based strategies as their names imply or they require a mapping between functions of the program and test cases. Results of the study [13] showed that some attributes of the applied changes can have significant impact on the effectiveness of test selection strategies. In particular, the distribution of changes throughout the functions and files of a program turned out to be a dominant factor, whereas the size of the applied changes did not.

The so-called *fix-cache* approach [6] selects test cases in three steps. First, all the files are ranked according to defect probabilities based on three properties: *i)* lines of code, *ii)* number of high severity error reports associated with the source file, and *iii)* modification frequency. Second, the top 10 percent of the listed files are determined as the set of defect-prone files. Finally, test cases are selected if they are linked to a modified source file, which also takes place in the set of defect-prone files. An empirical evaluation of this approach [5] provides promising results.

We consider the *fix-cache* approach to be the most related one to our approach that is presented and evaluated in this thesis. It employs the same method as depicted in Figure 1 for linking test cases to source code files. The implemented framework [6] daily analyzes the closed defect reports to determine source files updated to fix the defect and relate them to the test case that detected the defect. However, previous studies [5, 6] do not go deeper in the analysis of these relations. Some of the files can be changed only once due a defect detected by a test case, whereas some other files can be changed many times for the same reason. The number of changes suggests a level of strength for relations between test cases and

source files. In our study, we analyze the trade-off involved in a test case selection strategy that considers the variation of this strength. The traditional *fix-cache* approach considers a binary relation instead [6]. It employs a fault prediction algorithm [17] to determine a set of defect-prone files and uses this set to narrow down the selection. In our approach, we use a threshold parameter regarding the relation strength to control the size of the test suite. In the following, we present our test case selection approach.

CHAPTER III

TEST SELECTION APPROACH

Figure 1 illustrates the way that test cases are related to source code files. In our approach, we utilize the bug[1] repository as well as the code repository of the project.

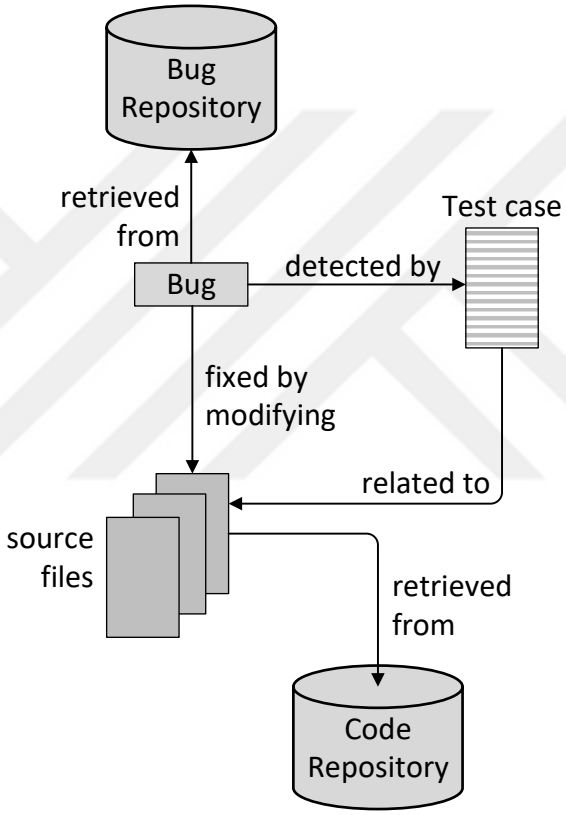


Figure 1: Relating test cases to source files.

We collect bugs that are previously reported and fixed for the project. Each of these bugs is detected by at least one of the test cases in the regression test suite and it is fixed as a result of a set of source code modifications. Hence, a bug is related to a test case as well as a set of source files that are modified to be able to fix the bug. We establish a relation between these source files and the test case that detected the bug. Our test case selection approach employs these

[1]We use the terms *bug* and *defect* interchangeably in the rest of the thesis.

relations, where a test case is selected if it is related to any of the source files that are modified in the last revision.

An illustrative toy example regarding the approach is depicted in Figure 2 with four test cases, three bugs and six source files. Hereby, *Test Case 1* detected *Bug 1*. *Source File 1* and *Source File 3* are changed to fix it. *Test Case 2* detected *Bug 2* and *Source File 4* is changed to fix it. *Test Case 3* has not detected any bugs yet. Finally, *Test Case 4* detected *Bug 3*. *Source File 5* and *Source File 6* are changed to fix it. Given this information, if *Source File 4* and *Source File 5* are modified before the next release, our approach would select *Test Case 2* and *Test Case 4* from the regression test suite since, there are the test cases that previously detected bugs associated with the modified source files.

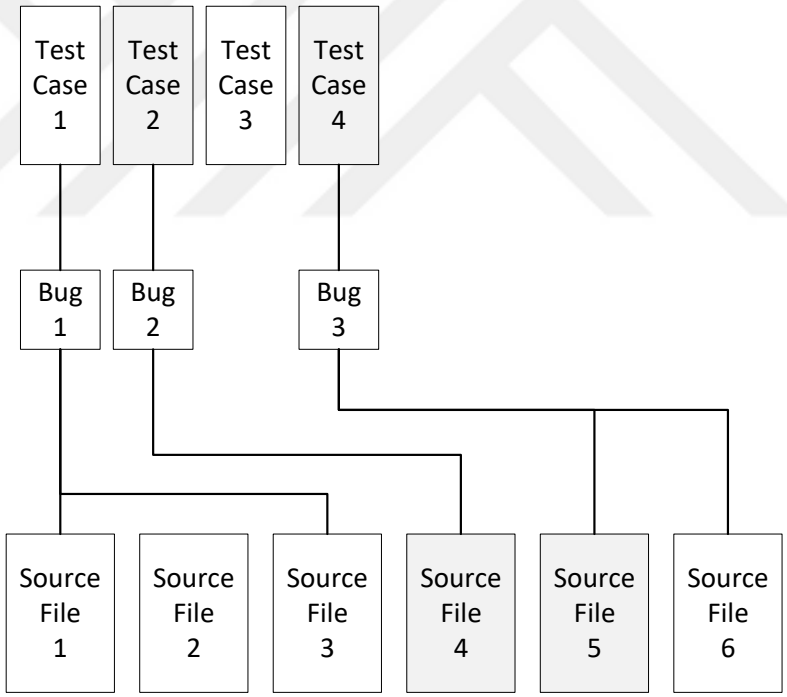


Figure 2: An illustrative sample scenario.

We observed two peculiarities to be addressed when this approach is applied in practice. First, it is not always possible to associate a bug with a unique test case. A bug can be reported after the execution of a set of tests. In such cases, we associate the bug with all of these tests. Second, the relation between tests

and source files are not always binary. A file can be modified multiple times to fix various bugs detected by a test case. The more such modifications exist, the stronger the relation becomes. We use a threshold parameter, S for adapting the considered relation strength for the selection of test cases. Let $F = f_1, f_2, ..., f_n$ be a set of files subject to modification and $T = t_1, t_2, ..., t_k$ be the set of test cases to be selected. The number of times file f_i has been modified[2] to fix bugs that are associated with test case t_j is denoted as $c(f_i, t_j)$. Then, the binary decision to select t_j or not, $s(t_j)$, is given as listed in the following.

$$s(t_j) = \begin{cases} 1, & \text{if } \exists f \in F, \text{ s.t. } \frac{c(f,t_j)}{\sum_{i=1}^{n} c(f_i,t_j)} \times 100 \geq S \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

Hereby, a test t_j is selected only if there exists a file that is modified in the last revision and that file was previously changed to fix bugs detected by t_j. In addition, the ratio of these changes must be greater than S with respect to all the changes performed on all the files to fix all the bugs detected by t_j. Our rationale is that a file can be related to a test if it is distinguished among other files in terms of the relation strength. The value of the parameter S determines the expected difference in relation strength as the selection criterion. No difference is expected and we consider a binary relation if $S = 0$, in which case a test is selected even if a modified file is changed only once to fix a bug detected by that test before. The selection of a value for S leads to a trade-off. Smaller values increase the test suite size. On the other hand, larger values can decrease the defect detection ability. In the following section, we present an industrial case study for the evaluation of our approach as well as this impact of the threshold parameter on results.

[2]Note that a file can be modified multiple times in various commits to fix more than one defect.

CHAPTER IV

INDUSTRIAL CASE STUDY

In this section, we present an industrial case study that is conducted in the context of Vestel [1], which is one of the largest TV manufacturers in Europe. Vestel produces TV products for 157 different brands from 145 different countries. There are hundreds of test suites being used in the company for testing these systems. These test suites are created for different types of tests such as performance tests, certification tests, connectivity tests and regression tests. In our project, we focus on regression tests only. We aimed at answering the following 4 research questions in this study.

- **RQ1**: How much cost reduction can be achieved with test case selection by relating code changes with previously fixed defects?
- **RQ2**: How is the effectiveness of the selected test cases in terms of defect detection ability?
- **RQ3**: What is the impact of the considered relation strength on the cost and effectiveness of test case selection?
- **RQ4**: How is the effectiveness of the selected test cases in detecting defects caught and reported by real users from the field?

The main goal of test case selection is to reduce the cost of regression testing by selecting a subset of the test cases from the regression test suite. *RQ1* is defined for measuring to what extent this goal is achieved. We measure the cost both in terms of the number/ratio of test cases selected and test execution time. Test duration reflects the real cost especially when there is a high variance among the execution

[1] `http://www.vestel.com.tr`

times of test cases [18]. This is the case for our study since some of the test cases are not automated. The drawback of test case selection is the risk of compromising from defect detection ability, while reducing cost. *RQ2* is defined to evaluate our approach from this aspect. The adjustment of the threshold parameter, S regarding the considered relation strength leads to a trade-off as explained in the previous section. *RQ3* is defined for evaluating the impact of this parameter on results. Finally, *RQ4* aims at investigating the effectiveness of the approach in capturing field errors reported by the end users. In the following, we introduce our experimental setup prepared for answering these research questions.

Our approach for test selection and evaluation of defect ability rate with selected test cases is depicted in Figure 3.

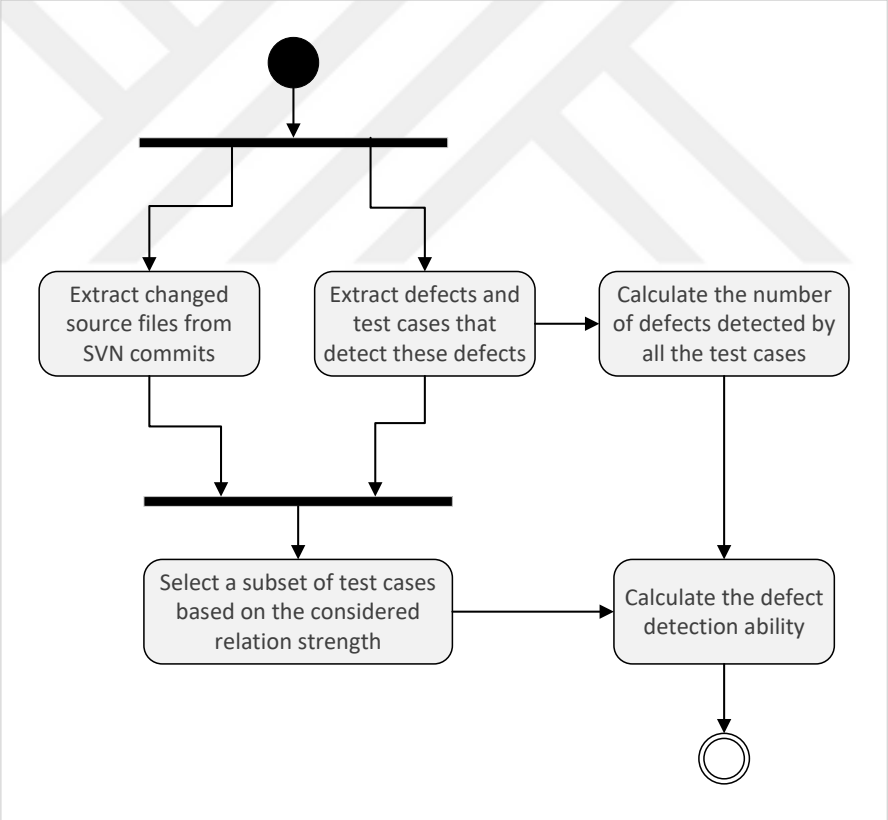


Figure 3: Test selection and evaluation of the defect detection rate

We compare the defect detection ability of the selected set of test cases with respect to that of all the test cases. We explain our experimental setup in the following section.

4.1 Experimental Setup

We used 3 Smart TV projects as experimental objects, enumerated as *P1*, *P2* and *P3*. We collected data regarding 8 software versions of these projects as listed in Table 1. These projects and versions were selected to increase the availability of data for analysis. They were the most active ones with the most number of end users.

Table 1: The list of enumerated projects used in the study, their versions and SVN revisions.

Project ID	**Software Version**	**SVN Revision**
P1	V1.37.0.0	319.405 → 320.082
P1	V1.48.0.0	323.377 → 323.742
P1	V1.71.0.0	334.026 → 334.618
P2	V1.44.0.0	328.479 → 328.822
P2	V1.47.0.0	329.316 → 329.819
P2	V1.53.0.0	332.343 → 332.631
P3	V0.51.19.0	311.881 → 312.796
P3	V0.51.28.0	324.786 → 325.791

We retrieved all the source file updates performed between the initial and final commits registered on the SVN [2] repository, which correspond to the transition to the new software version as listed in the last column of Table 1. We analyzed a total of 438 source code files, which are enumerated as shown in Table 2.

Table 2: A sample list of enumerated source files that are subject to change.

File ID	**Changed File Name**
CF1	/branches/Lu***/.../Sub***.cpp
CF2	/branches/Lu***/.../NovoFol***.cpp
CF3	/branches/Lu***/.../NovoSys***.cpp
CF4	/branches/DTV***/.../Novo***.cpp
CF5	/branches/DTV***/.../Ope***.cpp
...	...
CF438	/branches/Ky***/.../Tun***.cpp

There are 38 test cases in the regression test suite as listed in Table 3 together with their duration in minutes. The total execution time is 437 minutes (more than 7 hours) if all of the test cases are executed. As an example, the test steps

[2] `https://subversion.apache.org/`

of the *NFX Videoplay and Exit* case are shown in appendix B. We can observe a high variation in test execution times. 18 test cases take 5 minutes or less to complete each. However, the execution time for *TC8* and *TC32* is half an hour, for instance.

Table 3: The list of test cases in the regression test suite and their duration in minutes.

ID	Test Case Name	Duration
TC1	SmartCenter Video Display	25
TC2	Send UART Command	1
TC3	Switch to HDMI Source	2
TC4	Standby Off-On	2
TC5	Extended Logging	15
TC6	Channel Navigation	5
TC7	Oppap Installation	5
TC8	Set Timer-Record and Play	30
TC9	Google Assistant Control	15
TC10	HBB Videoplay and Exit	10
TC11	Web Page Videoplay and Exit	15
TC12	AMZ Videoplay and Exit	10
TC13	NFX Videoplay and Exit	10
TC14	YT Videoplay and Exit	10
TC15	MB Videoplay and Exit	10
TC16	Switch to Encrypted Channels	2
TC17	Global Search	4
TC18	First Time Installation	5
TC19	Power Off-On	3
TC20	Install SatCodex	5
TC21	Manual Search	2
TC22	Change Menu Values	5
TC23	Alexa Voice Command	15
TC24	Automatic Search	20
TC25	Change Volume	1
TC26	Open Portal and Exit	5
TC27	Add-Delete Favourite Channel	5
TC28	HD+ Installation	5
TC29	Feature OSD Control	20
TC30	Transponder Control	20
TC31	EIB Control	20
TC32	Timeshifht Control	30
TC33	Additional Logging	15
TC34	Interactive Channel Navigation	20
TC35	Internet Update	50
TC36	FVP Videoplay and Exit	10
TC37	Channel List Filter	5
TC38	Switch to Wireless Display Source	5
	Total	**437**

We collected bug reports from Jira[3], which is the bug tracking system used by the company. An example bug report is shared in Appendix A. In total, 482 defects reported in the past software versions were analyzed. Test cases that are associated with the detection of these defects as well as the source code files that are modified to fix these defects were extracted. It turns out that it is not always possible to associate a bug with a unique test case. A bug can be reported after the execution of a set of test cases, in which it is not possible to single out a particular test case to be responsible for detecting the bug. In such cases, we associated the bug with all of the test cases in the set. The maximum size of these sets was 5 in our case study. Table 4 shows a sample list of bugs and a set of test cases associated with the detection of these bugs for *P1*. In addition to the 482 defects detected during test executions, we also collected defect reports from the field for *P3*[4] to answer *RQ4* since *P3* has the most number of end users.

Table 4: A sample list of bugs and a set of test cases (TC) associated with the detection of these bugs for *P1*.

TC / Bug	**#1**	**#2**	**#3**	**#4**	**#5**
P1-10115	TC5				
P1-10070	TC7	TC8	TC11	TC13	
P2-5589	TC3	TC19			
P2-5149	TC11	TC12	TC13	TC14	TC15
P2-4205	TC14	TC4			
P3-13790	TC18	TC19	TC11		
P3-7473	TC25	TC15	TC22	TC2	

We identified the files that are changed for fixing a bug and associated these with the corresponding test cases. For instance, the set of files that are changed for fixing the bug *P2-5589* in Table 4 are associated with *TC3* and *TC19*. Table 5 lists the changed files and the number of changes applied to fix defects associated with 3 test cases for *P1*.

[3] `https://www.atlassian.com/software/jira`

[4] We collected defect reports from the field only for *P3* because the product corresponding to this project have been used in the field for a long period of time as opposed to the other projects.

Table 5: The list of changed files and the number of changes (#) associated with fixing defects for 3 test cases for *P1*.

TC1		**TC2**		**TC3**	
File ID	**#**	**File ID**	**#**	**File ID**	**#**
CF2	1	CF3	4	CF4	4
CF72	1	CF15	12	CF5	2
CF73	1	CF16	1	CF35	1
CF77	2	CF11	1	CF36	1
CF93	2	CF98	6	CF7	4
CF94	1	CF100	1	CF37	1
CF95	3	CF170	2	CF38	3
CF85	1	CF160	1	CF39	1
CF180	2	CF241	1	CF40	1
CF228	2			CF41	2
CF229	3			CF3	1
CF121	1			CF13	1
CF104	1			CF6	2
CF107	1			CF90	5
CF100	1			CF91	6
				CF96	2
				CF110	1
				CF111	1
				CF107	8
				CF142	1
				CF104	2
				CF105	1
				CF109	1
				CF121	2
Total	**23**	**Total**	**29**	**Total**	**58**

We can observe in Table 5 that the number of files, the number of changes in these files and the variance of this number is different for the listed test cases. For instance, *CF15* was changed 12 times to fix defects that are associated with *TC2*. This number is not even close to the number of changes recorded for the other files. Although there are more number of changed files associated with *TC1*, the number of changes ranges between 1 and 3. We decided to take this variation into account and adapt the test case selection process with a threshold parameter, S as explained in the previous section. For instance, *TC1* is selected if any of the

15 files associated with this test case is modified as listed in Table 5 when $S = 0$. On the other hand, it will be selected only if either *CF95* or *CF229* is changed when $S = 10$. This threshold value means that the number of changes previously recorded for the considered files must be greater than 10% of the total number of changes (23). Therefore, only those files with the number of changes greater than 2.3 are considered to be relevant for this setting. We performed experiments with gradually increasing S values ranging between 1 and 10. We extended this range up to 20 when we observe no significant impact on results for values of S up to 10. Results are presented and discussed in the following.

4.2 Results and Discussion

Results are depicted in two sets of charts for each project. Figures show the percentage of the selected test cases and the percentage of bugs detected by these test cases for *P1*, *P2* and *P3*, respectively. Figure 7, 8, 9, 13, 14, 15, 18 and 19 depict bar charts that show the number of defects detected by each test case for *P1*, *P2* and *P3*, respectively. The percentage of selected test cases (PSTC) is simply calculated by multiplying the number of selected test cases by 100/38 since there are 38 test cases in total. The percentage of bugs detected (PBD) is calculated as follows:

$$PBD = \frac{\#\ of\ defects\ detected\ by\ the\ selected\ test\ cases}{\#\ of\ defects\ detected\ by\ all\ the\ test\ cases} \times 100 \qquad (2)$$

Figure 4, Figure 5 and Figure 6 depicts the results for *P1*. In all the charts here, as well as the charts regarding the other projects (i.e., Figures 10, 11, 12, 16 and 17), PSTC is depicted with a red line, whereas PBD is depicted with a green line. Both of these values are aligned with the percentage values listed on the y-axis between 0 and 100. The x-axis represents various threshold values, i.e., S. Figure 4 depicts the results regarding version V1.71.0.0 of *P1*. We can see that both PBD and PSTC remain constant when $S \leq 2$. Hereby, 85% of the bugs can be detected by selecting only 60% of the test cases. We also observe that the amount of decrease in PBD and PSTC is aligned (the corresponding lines are almost parallel to each other) when S increased to 3, 4 and 5. However, there is a sudden drop in PBD when S is increased to 6. We observe similar trends for the other versions of *P1* as well, although the break points in lines are different. PBD and PSTC remain constant when $S \leq 4$ for version V1.48.0.0 (See Figure 5). A significant decrease of PBD is observed when S is increased to 9. We observe a balanced decrease in both PBD and PSTC for smaller values as potential trade-off points. PBD and PSTC remain constant when $S \leq 10$ for version V1.37.0.0 (See Figure 6). PBD quickly declines when $S > 12$.

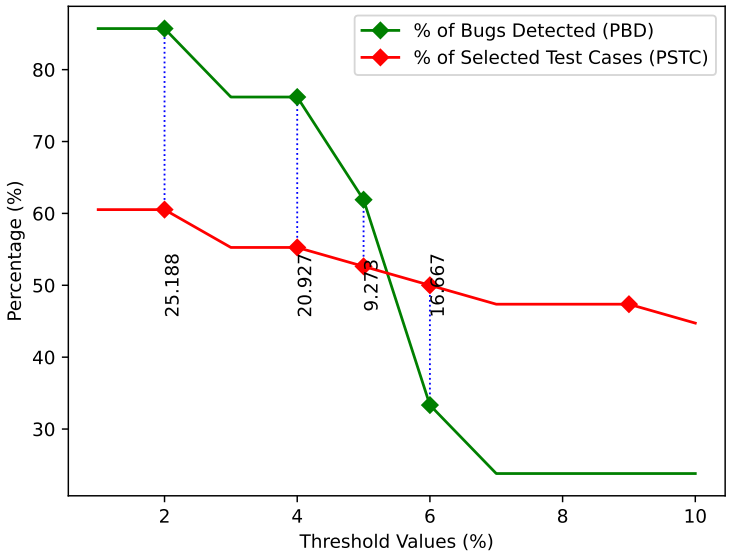


Figure 4: PSTC and PBD values obtained for project P1, version V1.71.0.0 for increasing threshold values regarding the number of changes.

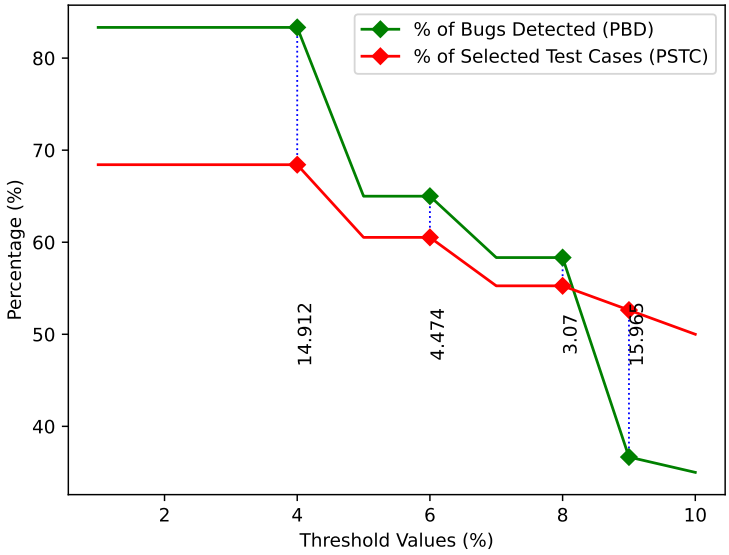


Figure 5: PSTC and PBD values obtained for project P1, version V1.48.0.0 for increasing threshold values regarding the number of changes.

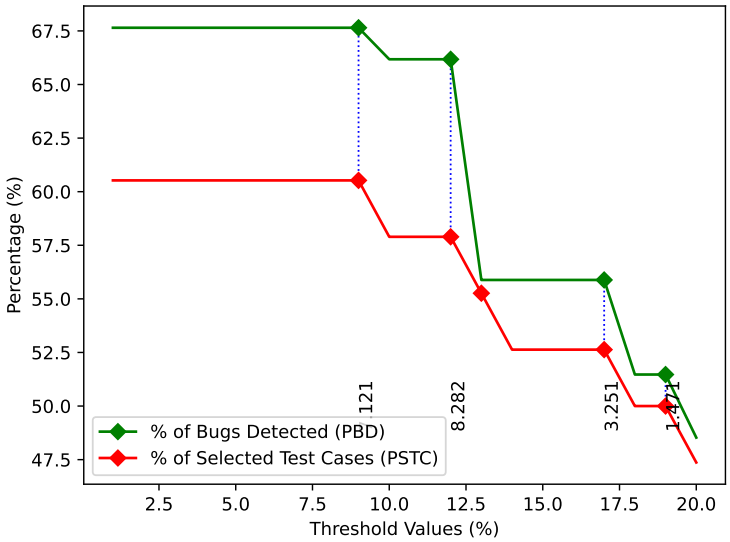


Figure 6: PSTC and PBD values obtained for project P1, version V1.37.0.0 for increasing threshold values regarding the number of changes.

The reason for sharp declines observed in PBD while S increases turns out to be the high variance in the number of bugs detected by test cases. Figure 7 8 9 depict this variation. For instance, elimination of *TC6* when S is increased to 6 in version V1.71.0.0 caused the sharp decrease in PBD. Likewise, *TC6* is eliminated in version V1.48.0.0, when S is increased to 9. We see in Figure 9 that *TC4* (*Standby off on*) in version V1.37.0.0 detects almost 25% of bugs detected by the whole test suite. This is because there is a large number of standby functions in changed files, and in general the target software incorporates the resolution of standby defects.

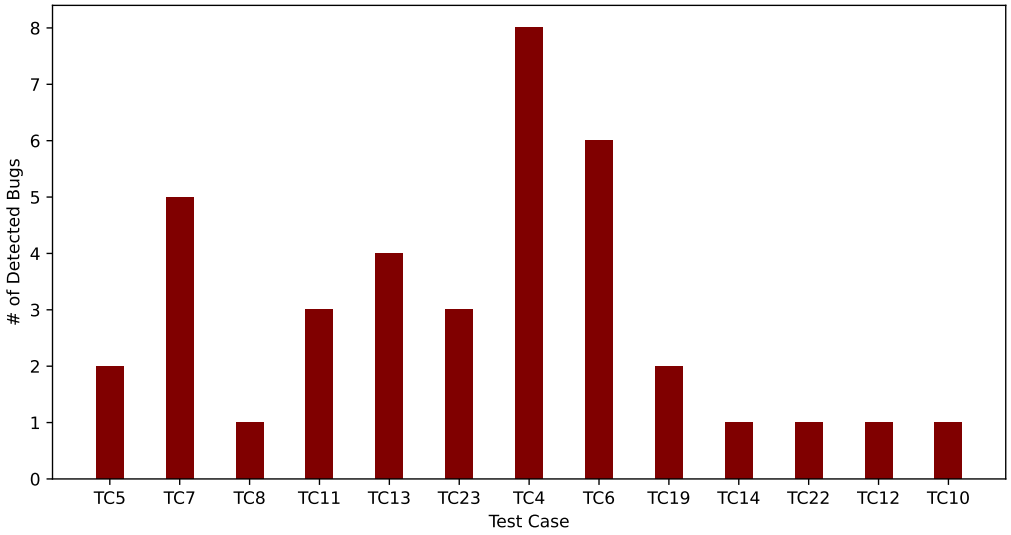


Figure 7: The number of detected bugs each test cases are presented for project P1, version V1.71.0.0.

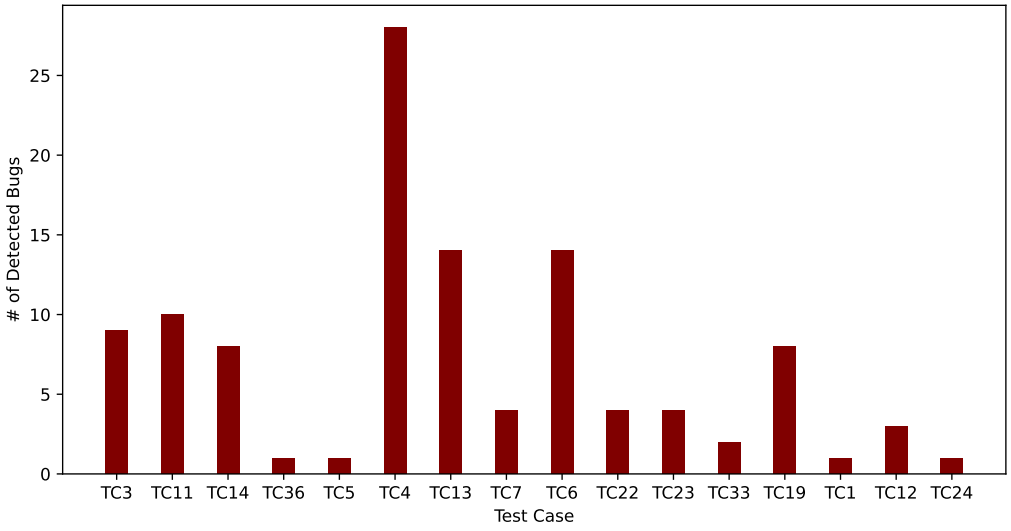


Figure 8: The number of detected bugs each test cases are presented for project P1, version V1.48.0.0.

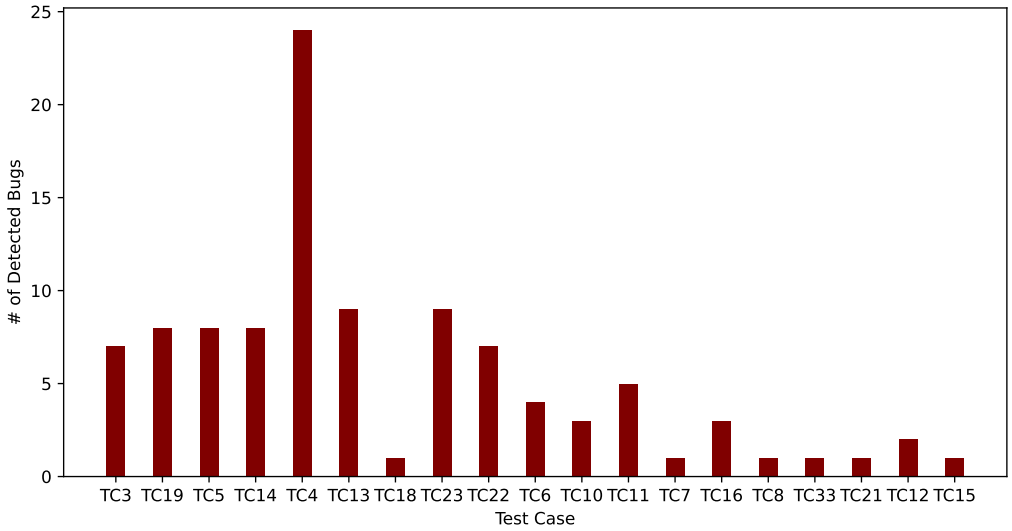


Figure 9: The number of detected bugs each test cases are presented for project P1, version V1.37.0.0.

10, Figure 11, Figure 12 depicts the results for *P2*. 63% of the bugs can be detected in version V1.53.0.0 (See Figure 10) with 36% of the test cases when $S \leq 2$. We observe a sharp decrease in PBD when S is increased to 2 and 8. We also see that PBD remains constant for values of S between 5 and 8 although PSTC decreases. Hereby, *TC6* and *TC7* are eliminated, which do not detect any bugs (See Figure 13).

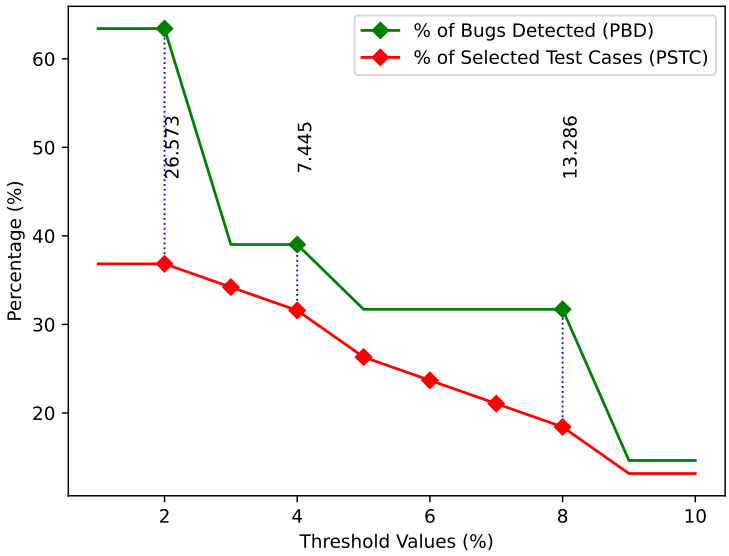


Figure 10: PSTC and PBD values obtained for project P2, version V1.53.0.0 for increasing threshold values regarding the number of changes.

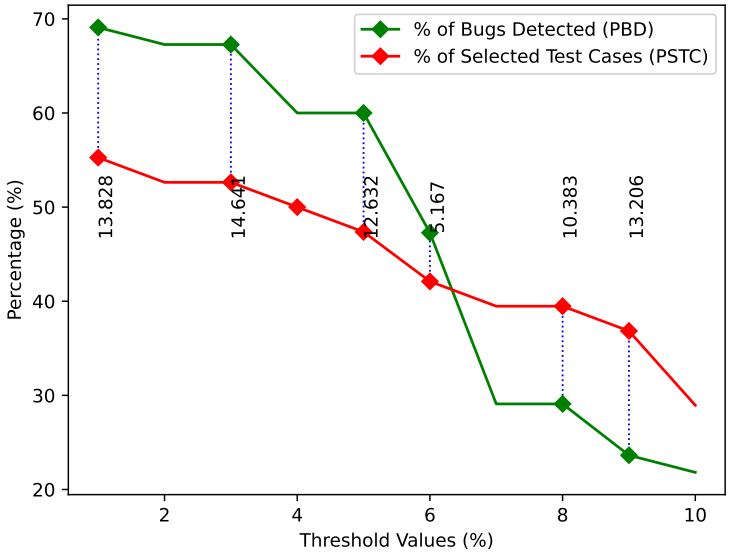


Figure 11: PSTC and PBD values obtained for project P2, version V1.47.0.0 for increasing threshold values regarding the number of changes.

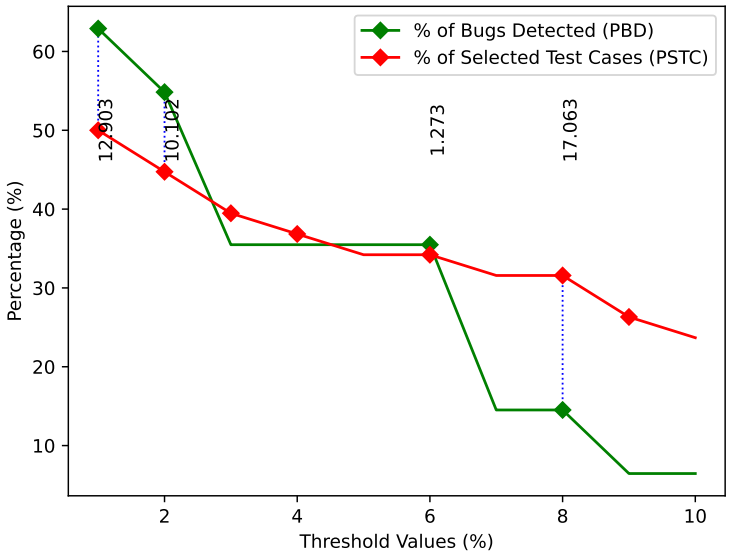


Figure 12: PSTC and PBD values obtained for project P2, version V1.44.0.0 for increasing threshold values regarding the number of changes.

PBD decreases from 60% to 29% when S is increased from 5 to 7 in version V1.47.0.0 (See Figure 11). This is due to the elimination of *TC4* and *TC15*, which account for the detection of a high number of bugs (See Figure 14). Sharp decreases in PBD for version V1.44.0.0 (Figure 12) is again attributed to high variation in the number of bugs detected by test cases (See Figure 15). The sudden degradation of PBD when S is increased from 6 to 7 is again caused by the elimination of *TC4*, which detects the highest number of bugs.

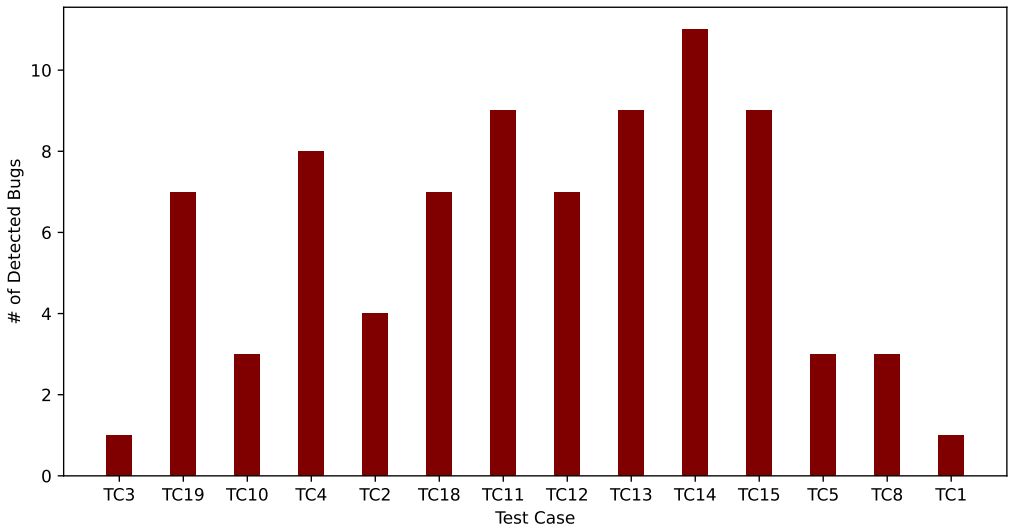


Figure 13: The number of detected bugs each test cases are presented for project P2, version V1.53.0.0.

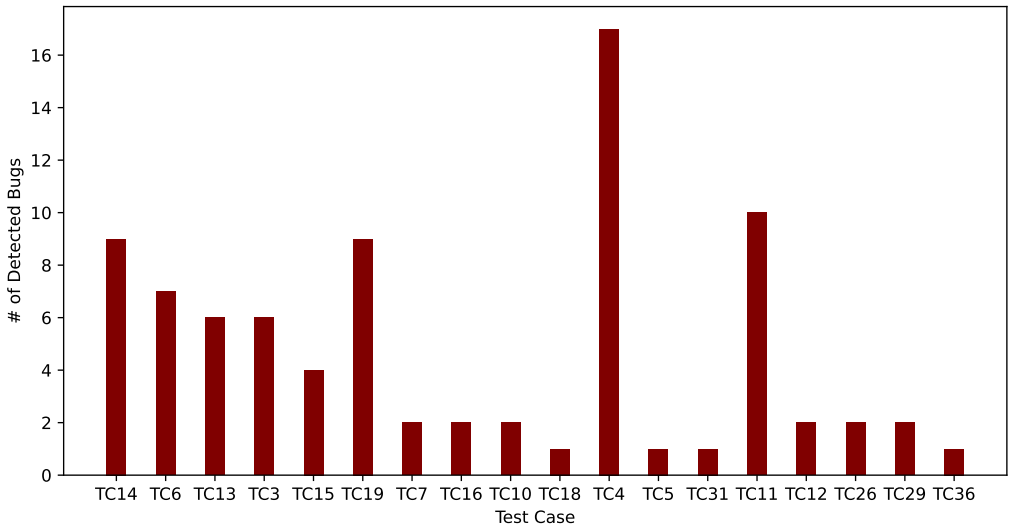


Figure 14: The number of detected bugs each test cases are presented for project P2, version V1.47.0.0.

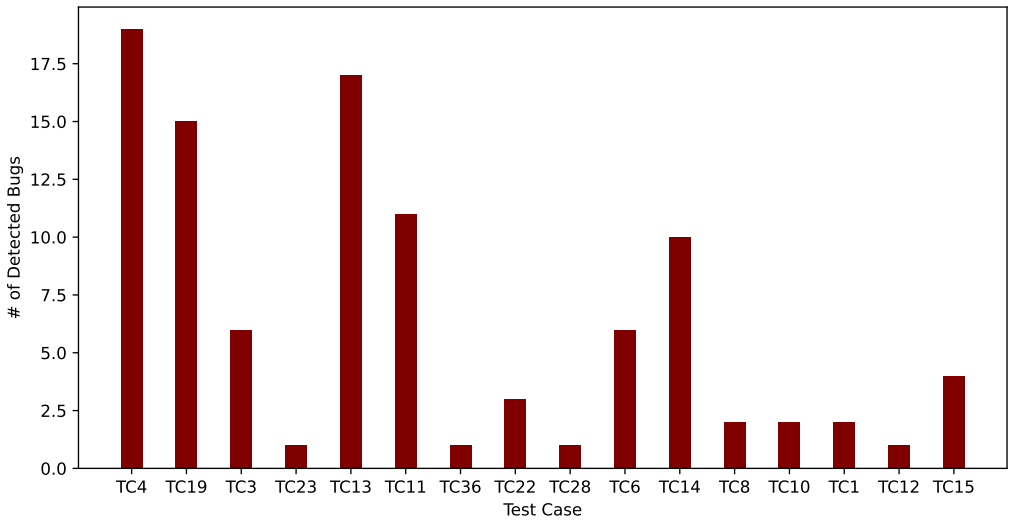


Figure 15: The number of detected bugs each test cases are presented for project P2, version V1.44.0.0.

Figure 16 and Figure 17, depict the results for *P3*. We analyzed two versions of this project. PDB is at its maximum value (65%) for version V0.51.19.0 (See Figure 16) when $S \leq 3$. It remains very high for version V0.51.28.0 in general. As seen in Figure 19, 24 out of 40 errors were related to TC6, which may explain this high PBD value. These defects were all related to the channel navigation functionality.

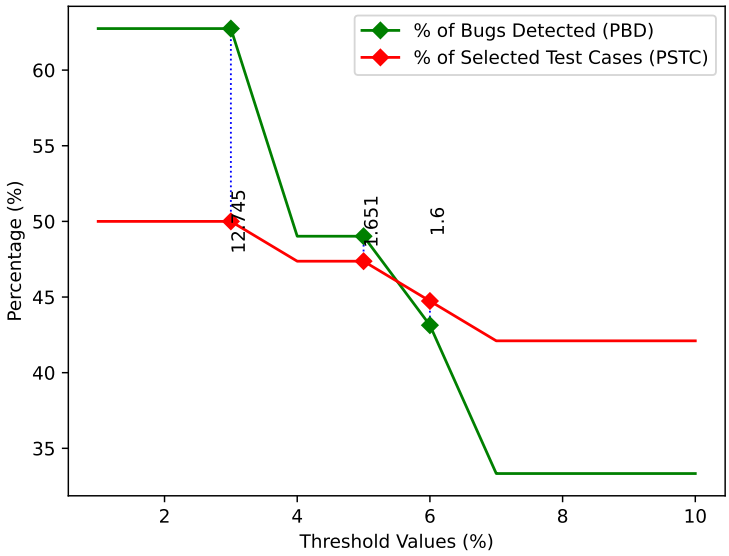


Figure 16: PSTC and PBD values obtained for project P3, version V0.51.19.0 for increasing threshold values regarding the number of changes.

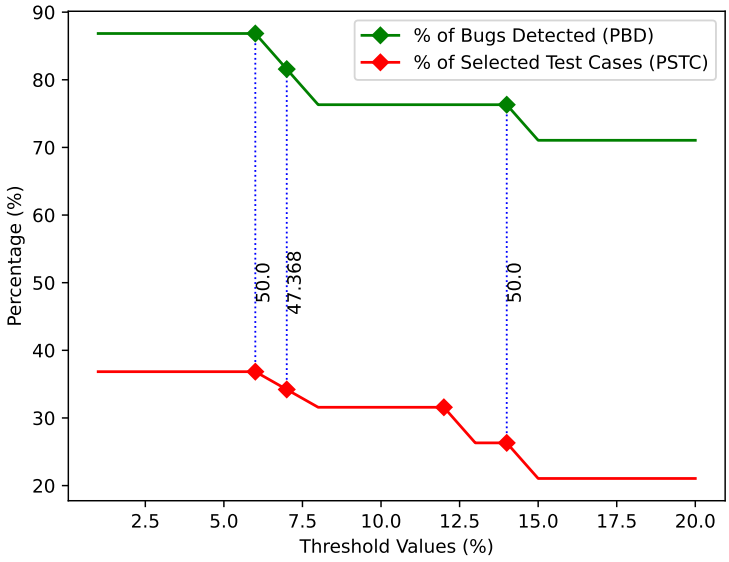


Figure 17: PSTC and PBD values obtained for project P3, version V0.51.19.0 for increasing threshold values regarding the number of changes.

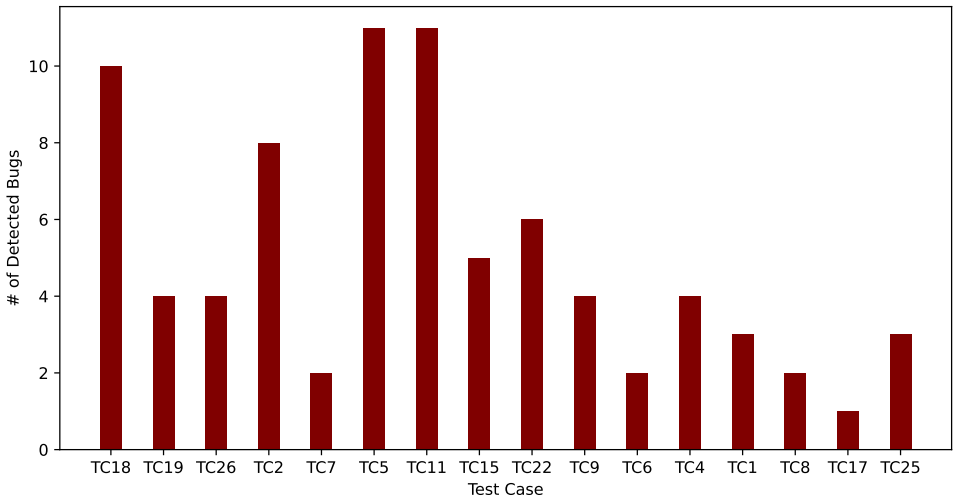


Figure 18: The number of detected bugs each test cases are presented for project P3, version V0.51.19.0.

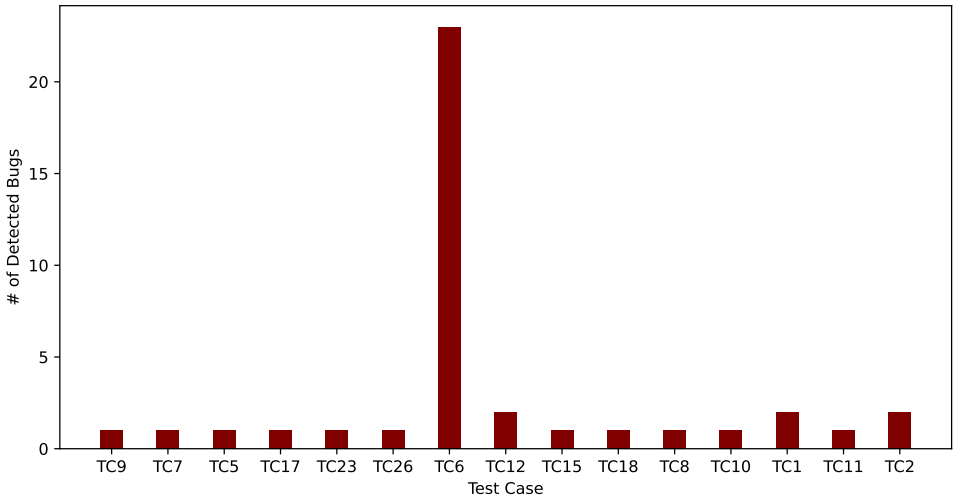


Figure 19: The number of detected bugs each test cases are presented for project P3, version V0.51.28.0.

In addition to the bugs detected during regression testing, we also collected customer complaints for the corresponding products in the field. These complaints arise because not all the test cases can be executed after every revision due to resource constraints including both human resources (especially for manual tests) and time. That is also why a cost-effective test selection strategy is necessary. A total of 5 defects were reported for version V0.51.19.0, which could be detected by the regression test suite.

We summarized all the results for *P1*, *P2* and *P3* in Tables 6, 7, 8, 9, 10, 11, 12 and 13 respectively.

We can conclude regarding *RQ1* that significant cost reduction is possible as a result of test case selection. The ratio of selected test cases is below 70% for all the versions of all the projects. Cost reduction is not only achieved in terms of the number of test cases but also the actual time necessary for test execution. Test execution time for the whole test suite is 437 minutes. It is possible to reduce this duration by at least 30% (considering the duration for the largest selected test set for version V1.48.0.0 of *P1* as 299 minutes). Test duration can be even reduced down to 69 minutes (See Table 9 for version V1.53.0.0 of *P2*) if the resources are very limited. Obviously, the more we reduce cost, the more we compromise from defect detection ability. However, the amount of decrease in PBD is mostly proportional to PSTC as discussed in the following.

The drawback of test case selection is the risk of compromising from defect detection ability, which was the motivation for defining *RQ2*. We observed that PBD indeed decreases; however, the amount of decrease can be acceptable in case of limited resources. Consumer electronics companies have to manage hundreds of brands from hundreds of companies at the same time [19, 20]. There is simply not enough time to always execute the whole test suite for every product. Results showed that one might still detect the majority of the defects rather than following an ad-hoc approach or not executing any tests at all. For instance, it is possible to detect from 76.32% to 86.84% of the test cases by executing only 26.31% to

36.84% of the test cases for version V0.51.28.0 of *P3* (See Table 13). In general, it is possible to detect from 65% up to 85% of the defects detected by the whole test suite by selecting from 30% up to 70% of the test cases when the threshold value is set as the minimum.

We observed that the inevitable trade-off discussed above can be balanced systematically by adjusting the threshold parameter, S. In general, we observed that the amounts of decrease in PBD and PSTC are aligned for increasing values of S up to a certain point. Potential trade-off points can be identified especially for $S \leq 10$ as listed in Tables 6, 7, 8, 9, 10, 11, 12 and 13 respectively. Therefore, we concluded regarding *RQ3* that S is useful in balancing the trade-off between PBD and PSTC.

We observed regarding *RQ4* that we could indeed detect field defects that could be otherwise omitted, by balancing the trade-off in test case selection systematically. In one of the versions of *P3*, 3 out of 5 field defects could be prevented by reducing the test duration from 437 minutes down to 166 minutes. A field defect could be prevented in another version of the project by just 82 minutes of text execution time. Although the number of these error could seem to be insignificant, their impact is very significant.

Table 6: Overall results for *P1* V1.71.0.0

S (%)	**V1.71.0.0**		
	STCD	***PBD***	***PSTC***
1	177	85.71	60.53
4	162	76.19	55.26
5	147	61.90	52.53
6	142	33.33	50

Table 7: Overall results for *P1* V1.48.0.0

S (%)	**V1.48.0.0**		
	STCD	***PBD***	***PSTC***
4	299	83.33	68.42
6	249	65	60.52
8	234	58.33	55.26
9	142	33.33	50

Table 8: Overall results for *P1* V1.37.0.0

S (%)	**V1.37.0.0**		
	STCD	***PBD***	***PSTC***
9	292	67.65	60.52
12	282	66.18	57.89
17	252	55.88	52.63
19	142	33.33	50

Table 9: Overall results for *P2* V1.53.0.0

S (%)	**V1.53.0.0**		
	STCD	***PBD***	***PSTC***
2	121	63.41	36.84
4	106	39.02	31.58
8	69	31.71	18.42

Table 10: Overall results for *P2* V1.47.0.0

S (%)	**V1.47.0.0**		
	STCD	***PBD***	***PSTC***
1	281	69.09	55.26
3	266	67.28	52.63
5	231	60	47.37
6	196	47.27	42.10
8	194	29.09	39.47
9	192	23.64	36.84

Table 11: Overall results for *P2* V1.44.0.0

S (%)	**V1.44.0.0**		
	STCD	***PBD***	***PSTC***
1	199	62.90	60.52
2	189	54.84	44.74
6	138	35.48	34.21
8	136	14.52	31.58

Table 12: Overall results for *P3* V0.51.19.0

S (%)	**V0.51.19.0**			
	STCD	***PBD***	***PSTC***	***RFE***
3	178	62.75	50	4/5
5	176	49.02	47.37	3/5
6	166	43.14	44.74	3/5

Table 13: Overall results for *P3* V0.51.19.0

S (%)	**V0.51.28.0**			
	STCD	***PBD***	***PSTC***	***RFE***
6	114	86.84	36.84	1/4
7	99	81.58	34.21	1/4
14	82	76.32	26.31	1/4

4.3 Threats to Validity

Our evaluation is subject to an *external validity* threat since we conducted a case study targeting at embedded systems that are developed and maintained within the context of a single company. More case studies in different companies and domains must be conducted to be able to generalize the results.

There are *internal validity* threats due to the dataset we used in our experimental setup. We collected data automatically from the bug and code repositories. However, the data itself is manually provided by developers and test engineers. Hence, there might be incorrect or missing entries. Some of the source code modifications might not be correctly linked to the corresponding defect items for instance. We also employed the regression test suite being used in the company as is. This test suite was prepared before our study.

In our study, we analyzed a previously produced project in which we examined field returns. We did not perform evaluations for a larger number of field projects since source code designs are not accessible in practice. In addition, there was not much practical data in the scope of regression tests from user complaints in projects produced in the past and in the field.

One might consider a *construct and conclusion validity* threat due to the size of the regression test suite that is being reduced. There are 38 test cases in the regression test suite in total. Although this number does not seem to be high, the total test execution duration is more than 7 hours. We observed that significant reduction is possible both in the number of test cases and in test execution time as a result of the test cases selection process. Moreover, such a reduction is possible without compromising from the defect detection ability. We present our concluding remarks in the following section.

CHAPTER V

CONCLUSION

We introduced an approach for black-box test case selection. Our approach relies on the assumption that the changed source code files are error-prone. Therefore, only those test cases that are related with these files should be selected. However, it is challenging to establish a relation between source code files and black-box test cases. We analyzed information regarding the previously fixed defects to address this challenge. We associated test cases that detect a defect with those source code files that had to be modified to fix the defect. Then, a regression test suite for the subsequent test cycle is determined by selecting those test cases that are associated with the source code files modified in the last revision.

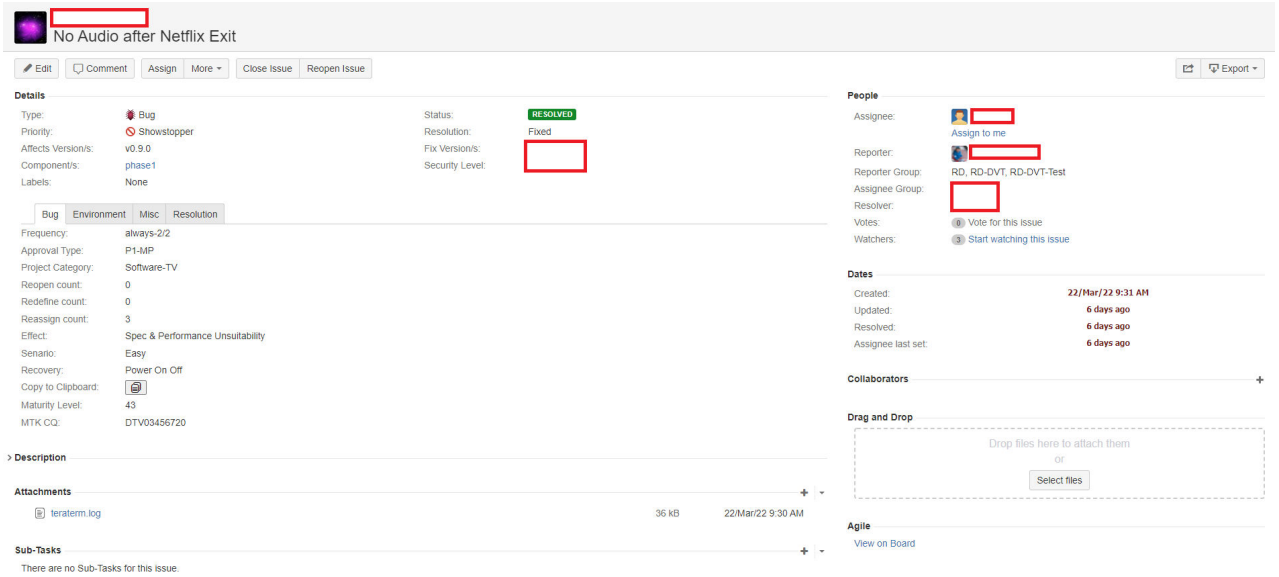
We conducted an industrial case study by collecting data from 8 versions of 3 real projects from the consumer electronics domain. We analyzed 482 defect reports regarding these projects. We associated these defects with modifications performed on 438 source code files. We observed that the strength of the relation between test cases and source code files can be subject to a high variation. Some of the files are changed only once due a defect detected by a test case. Some others are changed a dozen times due to various defects detected by the same test case. Hence, the test case selection process is subject to a trade-off between cost and effectiveness that can be balanced by adjusting a threshold parameter for the considered relation strength. Such a parameter is useful in balancing this trade-off because we observed that the amount of decrease in both cost and effectiveness is aligned for increasing values of the threshold up to a certain point. The exact value to be assigned for this parameter would depend on the available resources and the criticality of the project. Our results showed that it is possible to detect from 65% up to 85% of the defects detected by the whole test suite by selecting from 30%

up to 70% of the test cases when the threshold value is set as the minimum.

In our future work, we would like to investigate the effectiveness of black box test case selection technique to other functional and non functional tests. We would like to investigate alternative approach that can be adapt for our prioritization approach.

APPENDIX A

A SAMPLE BUG REPORT



APPENDIX B

A SAMPLE TEST CASE: NFX VIDEOPLAY AND EXIT

Table 14: Detailed test steps for *NFX Videoplay and Exit* test case.

TS ID	**Stimulation**	**Verification**
1	Press the Netflix button on the remote. Search for Pokemon : Indigo Legend from Netflix. Open the first episode.	There should be no problem on video playback, A/V should be fine. The video should appear in 4:3format.
2	Change the volume with V+/-, mute/unmute while any video is playing.	V+/-, mute/unmute changes should not be a problem. Volume OSD should appear on the screen.
3	Exit Netflix by pressing the Exit button.	Exit Netflix by pressing the Exit button.
4	Log in to Netflix via Home Launcher. Go to Netflix search field and search for ILIZA.While the video is playing, press the subtitle button on the remote to select the subtitle language in Turkish and check the subtitle.	There should be no problem when subtitle is displayed.
5	While the video is playing, change the audio information from the netflix options menu.	There should be no problem in audio changes, audio distortion, cutting etc. should not be a problem.
6	Exit Netflix by pressing the Exit button.	There should be no problem exiting Netflix, A/V should be ok when returning to the channel.
7	Enter Netflix again. Go to Netflix search field and search for IRON FIST. While the video is playing, check Pause/Play/FF/FR.	There should be no problem with Pause/Play/FF/FR operations. A/V should be OK.
8	Exit Netflix by pressing the Exit button.	There should be no problem exiting Netflix, A/V should be ok when returning to the channel.
9	Enter Netflix by pressing the netflix button on the remote. Go to the Netflix search field and search for IRON FIST. Select the series with OK button and open the first episode.	The desired video can be selected with OK button. A/V should be OK, no problem should be observed in video playback.
10	While the video is playing, press the back button on the remote.	Video playback should stop, menu options should appear.
11	Select the Resume option.	The video should continue playing where it left off.
12	Press the back button again and select Play from beginning.	The video should start from the beginning.
13	Log into Netflix via Home Launcher. Search and play for 4K (Moving Art: Flowers / Oceans) video.	A/V should be OK (if the project does not support 4K, it will appear in HD)
14	Press the info button while the video is playing.	Video duration, bitrate, resolution, etc. should appear on the screen. The video should appear in 4K resolution (if the project does not support 4K, it will appear in HD). Audio/text information should appear on the screen.

REFERENCES

[1] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[2] K. Herzig, M. Greiler, J. Czerwonka, and B. Murphy, "The art of testing less without sacrificing quality," in *Proceedings of the 37th International Conference on Software Engineering*, pp. 483–493, 2015.

[3] R. Kazmi, D. Jawawi, R. Mohamad, and I. Ghani, "Effective regression test case selection: A systematic literature review," *ACM Computing Surveys*, vol. 50, no. 2, pp. 1–32, 2017.

[4] E. Engström, P. Runeson, and M. Skoglund, "A systematic review on regression test selection techniques," *Information and Software Technology*, vol. 52, no. 1, pp. 14–30, 2010.

[5] E. Engström, P. Runeson, and G. Wikstrand, "An empirical evaluation of regression testing based on fix-cache recommendations," in *Proceedings of the International Conference on Software Testing, Verification and Validation*, pp. 75–78, 2010.

[6] G. Wikstrand, R. Feldt, J. Gorantla, W. Zhe, and C. White, "Dynamic regression test selection based on a file cache an industrial evaluation," in *Proceedings of the International Conference on Software Testing Verification and Validation*, pp. 299–302, 2009.

[7] P. Sapna and H. Mohanty, "Clustering test cases to achieve effective test selection," in *Proceedings of the 1st Amrita ACM-W Celebration on Women in Computing in India*, pp. 1–8, 2010.

[8] E. Rogstad, L. Briand, and R. Torkar, "Test case selection for black-box regression testing of database applications," *Information and Software Technology*, vol. 55, no. 10, pp. 1781–1795, 2013.

[9] X. Qu, M. Acharya, and B. Robinson, "Impact analysis of configuration changes for test case selection," in *IEEE 22nd International Symposium on Software Reliability Engineering*, pp. 140–149, 2011.

[10] E. Ekelund and E. Engström, "Efficient regression testing based on test history: An industrial evaluation," in *IEEE International Conference on Software Maintenance and Evolution*, pp. 449–457, 2015.

[11] G. Rothermel and M. Harrold, "A safe, efficient regression test selection technique," *ACM Transactions on Software Engineering and Methodology*, vol. 6, no. 2, pp. 173–210, 1997.

[12] T. Graves, M. Harrold, J. Kim, A. Porter, and G. Rothermel, "An empirical study of regression test selection techniques," *ACM Transactions on Software Engineering and Methodology*, vol. 10, no. 2, pp. 184–208, 2001.

[13] S. Elbaum, P. Kallakuri, A. Malishevsky, G. Rothermel, and S. Kanduri, "Understanding the effects of changes on the cost-effectiveness of regression testing techniques," *Software Testing, Verification and Reliability*, vol. 13, no. 2, pp. 65–83, 2003.

[14] Y. Chen, D. Rosenblum, and K. Vo, "Testtube: A system for selective regression testing," in *Proceedings of the 16th International Conference on Software Engineering*, pp. 211–220, 1994.

[15] K. Fischer, F. Raji, and A. Chruscicki, "A methodology for retesting modified software," in *Proceedings of the National Telecommunications Conference*, pp. 1–6, 1981.

[16] R. Gupta, M. J. Harrold, and M. Soffa, "An approach to regression testing using slicing," in *Proceedings of the Conference on Software Maintenance*, pp. 299–308, 1992.

[17] S. Kim, T. Zimmermann, E. J. Whitehead, and A. Zeller., "Predicting faults from cached history," in *Proceedings of the 29th International Conference on Software Engineering*, pp. 489–498, 2007.

[18] S. Dirim and H. Sozer, "Prioritization of test cases with varying test costs and fault severities for certification testing," in *Proceedings of the IEEE International Conference on Software Testing, Verification and Validation Workshops*, pp. 386–391, 2020.

[19] C. Gebizli, D. Metin, and H. Sozer, "Combining model-based and risk-based testing for effective test case generation," in *Proceedings of the 8th International Conference on Software Testing, Verification and Validation Workshops*, pp. 1–4, 2015.

[20] C. Gebizli, A. Kirkici, and H. Sozer, "Increasing test efficiency by risk-driven model-based testing," *Journal of Systems and Software*, vol. 144, pp. 356–365, 2018.

[21] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, 2001.

[22] S. Elbaum, A. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 159–182, 2002.

[23] A. Avritzer and E. Weyuker, "The automatic generation of load test suites and the assessment of the resulting software," *IEEE Transactions on Software Engineering*, vol. 21, no. 9, pp. 705–716, 1995.

VITA

Tutku Çıngıl received the degree of Bachelor of Science in Computer Engineering from Dokuz Eylul University, in June 2017. He worked as a software developer for one year and continued his career in software testing. Currently, he is a Software Test Specialist of Design Verification and Test Group of Television at Vestel Electronics which is one of the largest TV manufacturers in Europe. His main area of research interest is on Software Testing. In particular, he conducts research on test case selection and test case prioritization to increase test efficiency. Besides, he has been involved with automation testing and automation testing tools to avoid significant time and effort when running software functional and non-functional tests.