COGNITIVE HEURISTICS IN SOFTWARE ENGINEERING: EVALUATING CONFIRMATION BIAS DURING SOFTWARE TESTING

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Shishir Hegde

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Software Engineering

April 2017

Fargo, North Dakota

North Dakota State University
Graduate School

Title

COGNITIVE HEURISTICS IN SOFTWARE ENGINEERING: EVALUATING CONFIRMATION BIAS DURING SOFTWARE TESTING

By

Shishir Hegde

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Gursimran Walia
Chair

Dr. Kendall Nygard

Dr. Achintya Bezbaruah

Approved:

04-13-2017
Date

Brian Slator
Department Chair

ABSTRACT

Software engineering is a human centric activity and the thought processes of software engineers have influence on the quality of software products. Cognitive scientists have identified human errors known as cognitive heuristics which could impact quality of a software product. However, there is little empirical evidence to substantiate this assertion in software engineering. In this research we study a specific heuristic and evaluate its impact in software testing. One of the factors that lead to poor quality of testing is to only verify if the system works as expected and ignore negative tests. This can be attributed to a heuristic called as confirmation bias which is defined as the tendency of people to verify their hypothesis rather than refuting them. The experiment design evaluates confirmation bias of software testers and measures their quality of testing. The results indicate that testers with low confirmation bias obtain better overall testing results.

ACKNOWLEDGEMENTS

I’m grateful to my adviser Dr. Gursimran Walia for his continuous help, support, patience and guidance in the development and completion of this research study and paper. He has been a helpful advisor who has motivated me at every step and guided me during my master’s program.

I’m grateful to Dr. Kendall Nygard for giving me the opportunity to pursue my master’s program and always being helpful with various stages in the progress of my study and for taking out the time to be a part of my supervisory committee.

I’m grateful to Dr. Achintya Bezbaruah for taking out the time to be a part of my supervisory committee.

I’m grateful to the Software Engineering department faculty and staff in all the ways I could use their help in the progress and completion of my Master’s program.

Finally, I’m grateful to my parents, sister and wife for being supportive all the way.

TABLE OF CONTENTS

ABSTRACT iii

ACKNOWLEDGEMENTS iv

LIST OF TABLES vi

LIST OF FIGURES vii

1. INTRODUCTION 1

2. BACKGROUND AND RELATED WORK 4

2.1. Psychological Perspective on Cognitive Heuristics 4

2.2. Confirmation Bias studies in Software Engineering 5

2.3. Motivation 7

3. EXPERIMENT DESIGN 8

3.1. Experiment Methodology 8

3.1.1. Research Questions and Hypotheses 8

3.1.2. Variables 9

3.1.3. Participating Subjects 9

3.1.4. Artifacts 9

3.2. Experiment Procedure 10

4. ANALYSIS AND RESULTS 11

4.1. Number of Negative Test cases and Invalid Classes (H1) 11

4.2. Overall Effectiveness of Defect Detection (H2) 12

4.3. Effect of Experience on Negative test case detection (H3) 13

5. DISCUSSION OF RESULTS 16

6. CONCLUSION 18

REFERENCES 19

LIST OF TABLES

Table Page

1. Example of Cognitive Heuristics Applicable to Software Engineering.....5

LIST OF FIGURES

Figure Page

1. Comparison of negative tests detected by falsifier and verifier groups 11
2. Comparison of invalid classes covered by falsifier and verifier groups 12
3. Comparison of overall test coverage by falsifier and verifier groups 13
4. Comparison of invalid classes covered based on experience of subjects 14
5. Comparison of overall test coverage based on experience of subjects 15

1. INTRODUCTION

Software Engineering is the study and application of engineering principles to all the different processes involved in creating a software product. Software production at the abstract level involves the processes of Requirements engineering, software design, development, testing and maintenance of software. Software testing is an important process in the Software development life cycle to identify defects in the software. The quality of a software product can be determined by measuring the number of defects present in it and software testing helps in identifying defects so that it can be fixed. Poor or inadequate testing can lead to higher number of defects which negatively impacts the quality of software because the product is released to customers with defects existing in the software product. It is important to investigate the effects of the human cognitive process and its fallibilities during software testing in order to prevent inadequate testing. Since, software development and testing is human-centric, analysis of project failures consistently revolves around various human errors made by software engineers and testers. It is not surprising that most of the project failures are human in nature [4]. Test design logic could be incomplete and some of the necessary test conditions could be missing in test-case specifications. A common example of this situation is a lack of negative test cases. By definition, a test case is negative if it exercises abnormal conditions by using either invalid data input or the wrong user action. A lack of negative test cases in test specifications is a common cause of missed defects [5]. This can be attributed from psychology literature to a human cognitive heuristic called as confirmation bias which was first defined by Peter Wason in his rule discovery experiment and later in his selection task experiment [3]. It is defined as the tendency of people to seek evidence to verify their hypothesis and ignore evidence that falsify hypothesis. There are very few empirical evidences that establish proof for the assertion that confirmation bias can lead to low coverage of negative cases in test design and hence it is important to evaluate human error patterns like confirmation bias that could result in a lack of negative testing. The problem

statement in this research is to find out if confirmation bias has any impact on the quality of software testing done by software engineers/testers.

The notion of cognitive biases was first introduced by Kahneman [1]. There are various cognitive biases documented in the psychology literature and some of them are availability, representativeness, anchoring and adjustment, and confirmation bias [1, 7]. Although an intensive amount of research about cognitive biases exists in the field of cognitive psychology, interdisciplinary studies about the effects of cognitive biases in the software development life cycles are at an immature level. Stacy and MacMillian are the two pioneers who recognized the possible effects of cognitive biases on software engineering [2]. They discussed how cognitive biases might have an effect in software engineering activities by giving examples from several contexts. However, their work contains no empirical evidence and the authors concluded that their ideas can be used for further research and empirical validation.

In this paper, we study confirmation bias and extract confirmation bias metrics of software testers using a psychology test based on Wason's selection task [3] and then correlate it with the results from a software testing task to understand if confirmation bias influences the quality of negative testing among software testers. There is evidence from previous research that show testers carry out more positive tests as compared to negative tests due to positive test bias [6]. In our study we focus only on the quality of negative tests and correlate it with the results from a confirmation bias test from the psychology literature to understand if confirmation bias influences the ability of software testers to identify negative tests.

The research goal is to analyze the confirmation bias of testers for the purpose of understanding its impact on software testing with respect to the defect detection effectiveness from the point of view of researchers in the context of computer science students doing software testing.

The research design includes student subjects doing a software testing task to identify test cases. The same students are then subjected to a confirmation bias psychology test based

on Wason's selection task [3] to get their confirmation bias metrics. The results obtained from the psychology test are then correlated with the results from the testing task to measure the quality of software testing.

The rest of the paper is organized as follows: In Section 2, we discuss the related work on confirmation bias in software engineering. Section 3 describes the study design. Section 4 describes data analysis and results. Section 5 discusses the results obtained in this study with respect to the research questions and hypothesis and section 6 will have the conclusions of the paper.

2. BACKGROUND AND RELATED WORK

2.1. Psychological Perspective on Cognitive Heuristics

Psychological study of human errors received increased attention beginning in the early 1970's [1, 7]. Systematic models of human error capitalized on basic theoretical research in human cognition, especially related to information processing. It quickly became apparent that errors generally were not the result of irrational or maladaptive tendencies, but instead resulted from normal psychological processes gone awry. Generally the use of heuristics increases the speed and decreases the effort involved in information processing at the expense of accuracy [8]. Heuristics reduce effort by 1) examining fewer options before arriving at a solution, reducing the effort of information processing; 3) integrating less information before arriving at a solution [9]. Generally, the literature indicates that the loss of accuracy is not significant in most contexts, however software engineer's performance is affected by such cognitive biases. The psychological literature on human errors takes the perspective that human beings are cognitive misers that rely on the use of cognitive heuristics to make decisions. While much of the time, use of heuristics leads to accurate decision-making, research indicates that this is not always the case [8, 10].

We are interested in the conditions where the use of heuristic decision-making leads to erroneous decisions that introduce errors into software. At the individual level, psychology research focuses on the use of heuristics as a source of errors in decision-making. There are a wide variety of heuristics that could negatively affect software development. Table 1 illustrates a selection of heuristics from [8, 19] that can be used to understand the software engineering errors.

Table 1. Example of Cognitive Heuristics Applicable to Software Engineering

Heuristic	Definition
Availability	A mental shortcut used by individuals to estimate the likelihood of an event by how easily such events come to mind
Anchoring and Adjustment	A mental shortcut in which initial information on a topic forms an initial rough estimate or anchor point. Subsequent information leads an individual to adjust their estimate with the anchor point as the frame of reference.
Representativeness	A mental shortcut in which individuals classify a person or thing into a category to the extent that it is like a typical instance of the category
Less is More	A mental shortcut when less information or thought processing leads to more accurate judgments.
Recognition	A mental shortcut in which when choosing between two options, the more familiar option will be deemed the correct one, regardless of actual accuracy.
Fluency	A mental shortcut in which multiple solutions are recognized, but one is faster, individuals will choose the faster option.
Take the First	A mental shortcut in which individuals chose the first option that comes to mind.
One Reason	A mental shortcut in which a decision is made based on the first good reason that comes to mind rather than any subsequent reasons (good or bad).
Trade Off	A mental shortcut in which a decision is made by weighing all alternatives equally and examining their trade-off.
Tallying	A mental shortcut in which a decision is made by the quantity of an option rather than any other merits.

The thought processes of developers are a fundamental concern in software development. Stacy and Macmillan recognized the potential effects of cognitive biases on software engineering [2]. The authors discussed how cognitive biases might show up in software engineering activities by giving examples from several contexts. However, this work contains no empirical investigations. The authors put forth some ideas with explanations and potential areas that require further research.

2.2. Confirmation Bias studies in Software Engineering

A few previous studies analyzed some of the factors affecting confirmation bias and their effects on software development and testing. One study provided empirical evidence that supports the existence of positive test bias among software testers [6]. The authors found that testers are four times more likely to choose positive tests than to choose negative ones and that professional testers are no better at obtaining coverage of the test space than

novices. They also found that professional testers run many more test cases than less expert testers and professional testers are not significantly affected by the level of detail of the specifications. Results from another study showed that having strong logical reasoning and hypothesis testing skills are differentiating factors in the software developer/tester performance in terms of defect rates. Experience factor did not affect confirmation bias levels significantly. Individuals who are experienced but inactive in software development/testing scored better than active experienced software developers/testers. The recommendation was that companies should focus on improving logical reasoning and hypothesis testing skills of their employees by designing training programs to improve these skills [11]. Another study provided evidence that the size of company was not a differentiating factor in abstract reasoning skills. Software Engineers of large scale telecommunication company had comparatively less confirmation bias compared to small and medium size companies but graduate students of computer engineering showed less confirmation bias compared to software engineers of all companies. Results indicated experience did not play a role even in familiar situations such as problems about the software domain. Students showed less confirmation bias compared to experienced software engineers by applying more reasoning skills to solve software domain problems [12]. Results from another study analyzed factors such as company culture and education and results indicated that problem solving methodology followed by a large software development company in North America was better than a large telecommunication company in Europe since confirmation bias was significantly less according to the metrics evaluated. Results also indicated that more education among software engineers can mitigate confirmation bias since graduate students performed better in some of the confirmation bias metrics compared to software engineers from the telecommunication company in Europe [13]. One more previous study used confirmation bias metrics in building defect prediction models and the results were compared with defect prediction models built using static code metrics and churn metrics. The improvement in

defect prediction performance as a result of using confirmation bias metrics was not significant but gave comparable performance results [14].

Other cognitive biases like Anchoring and adjustment bias within the context of software development was studied by performing two experiments to investigate the existence of the bias in software artifact reuse. The first experiment they conducted examined the reuse of object classes in a programming task, while their second experiment investigated how anchoring and adjustment bias affected the reuse of software design artifacts [15]. Another study discussed how over-optimism and over-confidence of software engineers contaminated the results obtained by software effort predictors, making them far from objective [16]. One more study empirically investigated some cognitive bias types within the scope of software development effort estimation. According to these empirical findings, an increase in the effort spent on risk identification during software development effort estimations leads to an illusion of control, which in turn leads to more over-optimism and overconfidence [18]. Moreover, as a result of availability bias, risk scenarios that are more easily recalled are overemphasized so that inaccurate effort estimations are made. This study also empirically investigated how anchoring and adjustment heuristics leads to inaccurate effort estimates [17].

2.3. Motivation

The study described in this paper is motivated from the research of Stacy and Macmillan who recognized the potential effects of cognitive biases on software engineering [2]. The authors discussed how cognitive biases might show up in software engineering activities by giving examples from several contexts. However, this work contains no empirical investigations. The authors put forth some ideas with explanations and potential areas that require further research.

3. EXPERIMENT DESIGN

The major goal of this study is to investigate with empirical evidence for claims from previous studies that confirmation bias of software testers can result in inadequate negative testing.

The experiment to test this claim is a repeated measures design in which we perform a software testing task to get the test cases identified by the participants in the experiment and then perform a psychology test based on Wason's selection task to understand the confirmation bias levels of the testers and then correlate the results from the two tests.

3.1. Experiment Methodology

3.1.1. Research Questions and Hypotheses

The research questions investigated in this study were:

Research Question 1: Do software testers who have low confirmation bias levels detect more negative test cases and obtain better coverage of invalid classes during software testing?

Research Question 2: Does the confirmation bias levels of software testers have an impact on the overall defect detection effectiveness during software testing?

Research Question 3: Do software testers with low confirmation bias and having high work experience in the software development/testing industry detect more invalid classes and obtain better overall test coverage during software testing?

The hypotheses related to the above questions are:

Hypotheses 1: Software testers with low confirmation bias levels detect more negative test cases and obtain better coverage of invalid classes during software testing.

Hypotheses 2: The overall effectiveness of defect detection can be improved with low confirmation bias levels.

Hypotheses 3: Software testers with high experience will be able to detect more invalid classes and obtain better overall test coverage during software testing.

3.1.2. Variables

3.1.2.1. Independent variables

a. Requirement specification for testing task

b. Wasons selection task specifications

3.1.2.2. Dependent Variables

a. Number of subjects categorized as Falsifiers from Wasons selection task

b. Number of subjects categorized as Verifiers from Wasons selection task

c. Number of negative test cases detected by each subject in testing task

d. Number of Invalid classes detected by each subject in testing task

e. Total testing coverage percentage of each subject in testing task

3.1.3. Participating Subjects

Twenty graduate students enrolled in Software Testing and Debugging course at North Dakota State University (NDSU) participated in this study. This course was primarily focused on the goals, practices, evaluation and limitations of software testing and software debugging. Students receive practice in developing and using test plans and various testing and debugging techniques in this course.

3.1.4. Artifacts

The software testing task was administered by providing a testing task based on a given requirement specification to the subjects. The subjects were asked to do black box testing for this task by using the requirement specification. The psychology test was administered using a survey website in which the subjects were given a link to the website with four questions based on Wason's selection task and were asked to select their answer from the list of available choices.

3.2. Experiment Procedure

The experiment was designed to contain a single group of subjects to evaluate the hypotheses posed in section 3.1.1. The details of the experiment are provided in the following subsections.

- Software testing task: The subjects were given a testing task with specifications and asked to do black box testing to identify test cases for the task. Black-box testing is a method of software testing that examines the functionality of an application based on the specifications and no access to code. The testing task was given as a take home assignment to give ample time for the subjects to do testing. The subjects were asked to record the test cases according to a template with input values and expected output.
- Psychology test: The same group of subjects were administered a psychology test based on Wason's selection task. In this task the subjects were asked to provide their answers to four different variations of Wason's selection task through a website with multiple choice answers. This task was also a take home task with ample time to provide the answers to the questions.
- Classification of Falsifier/Verifier/Matcher from the Psychology test: The classification strategy was employed based on existing psychology literature by Reich and Ruth [20]. For the conditional rule of the form "if P, then Q", the subject who selects the choices that could break the rule is classified as a falsifier which in our tests is the answer choice of "P, not-Q". The rest of the subjects would belong to the category of Verifiers. They are categorized based on answer choices that verify the rule but may not have the potential to break the rule. There can also exist a tendency for subjects to just select an answer choice based on what is specified in the rule and they can be called as matchers. Matchers are classified in the same group as Verifiers since for the conditional rule of the form "if P, then Q", the subject who selects P,Q as the answer can either be a verifier or matcher.

4. ANALYSIS AND RESULTS

This section provides an analysis of the data collected during the study. This section is organized around the hypotheses presented in Section 3.1.1. An alpha value of 0.05 was selected for judging the significance of results.

4.1. Number of Negative Test cases and Invalid Classes (H1)

The results from the psychology test was used to classify the subjects into falsifier and verifier categories. Falsifiers exhibit lower confirmation bias levels as compared to verifiers. An independent samples t-test was chosen to do the analysis to compare the average number of negative test cases detected by the falsifier and verifier groups. The falsifier group detected significantly more negative test cases than the verifier group finding an average of 11.75 negative test cases as compared to an average of 6.58 negative test cases (p=0.038). These results are shown in Figure 1.

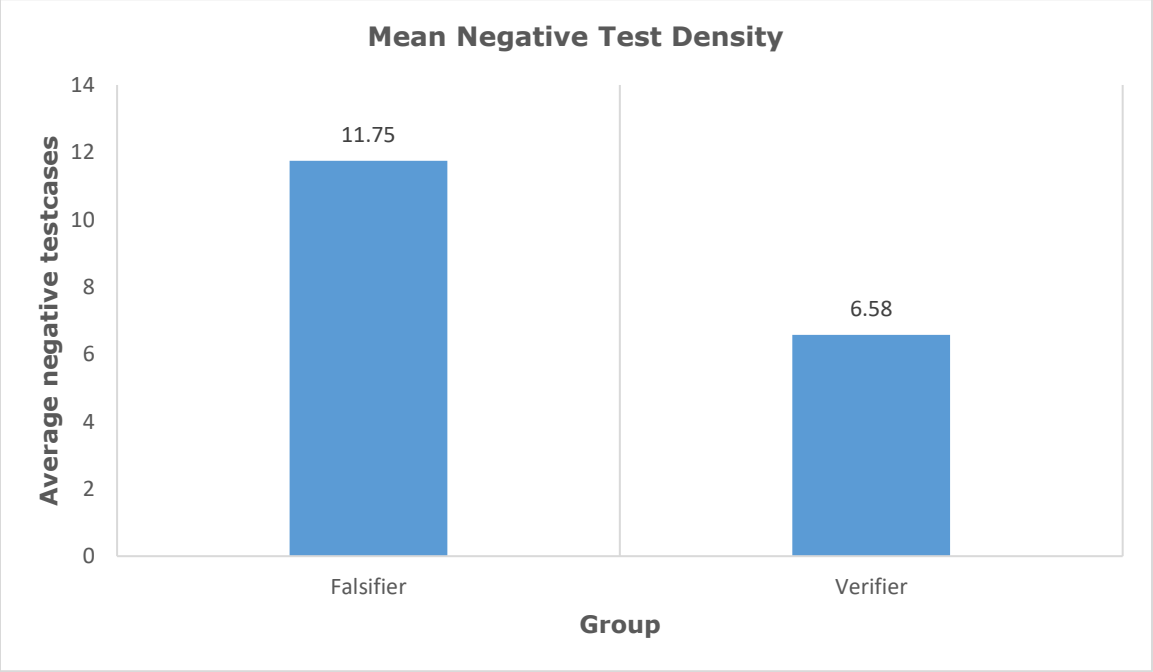

Figure 1. Comparison of negative tests detected by falsifier and verifier groups

The quality of negative test cases detected by the subjects was further validated by employing equivalence partitioning and boundary value analysis method [21] to get the invalid classes covered and eliminate redundant test cases. The results indicated the falsifier

group covered significantly more invalid classes than the verifier group covering an average of 8.88 classes as compared to an average of 6.08 classes (p=0.047). These results are shown in Figure 2.

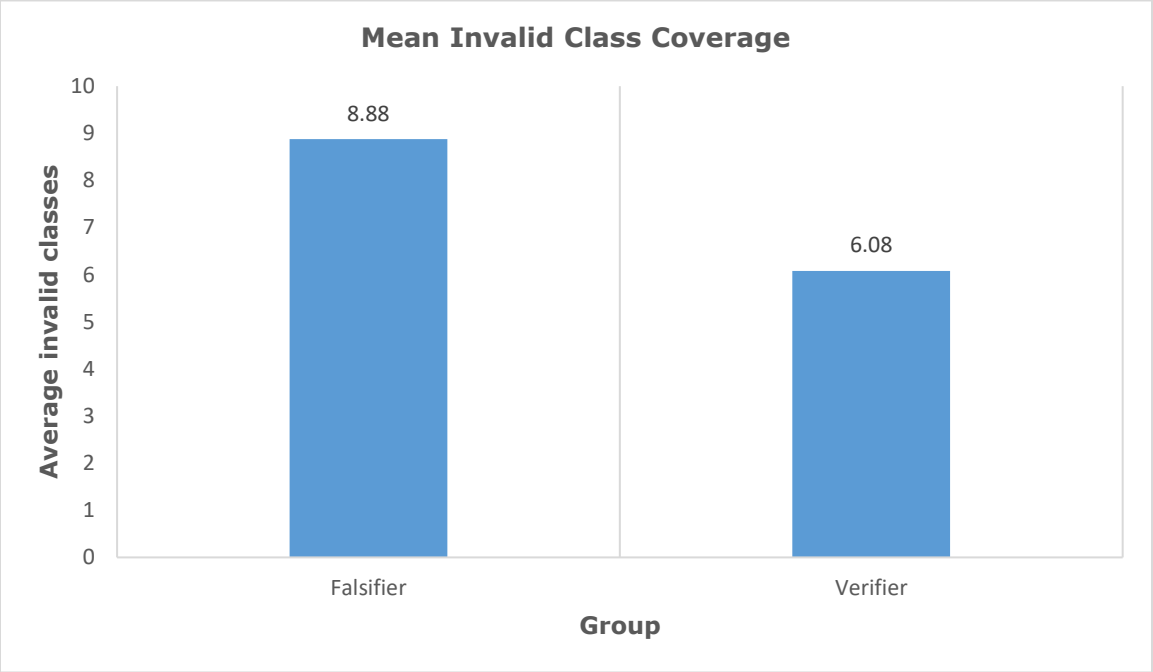

Figure 2. Comparison of invalid classes covered by falsifier and verifier groups

4.2. Overall Effectiveness of Defect Detection (H2)

The overall effectiveness of testing was measured by employing equivalence class and boundary value analysis method [21]. Every possible equivalence class and boundary value class was identified for the task and the percent coverage of the test cases identified by the subjects was measured. Percent coverage was calculated as the ratio of the equivalence classes or boundary conditions actually tested by the subjects to the total possible. This indicates the quality as opposed to quantity of the tests. An independent samples t-test was chosen to do the analysis to compare the average percent coverage by the falsifier and verifier groups. The results indicated the falsifier group obtained significantly more overall percent coverage than the verifier group obtaining an average of 41.40% coverage as compared to an average of 31.24% (p=0.036). These results are shown in Figure 3.

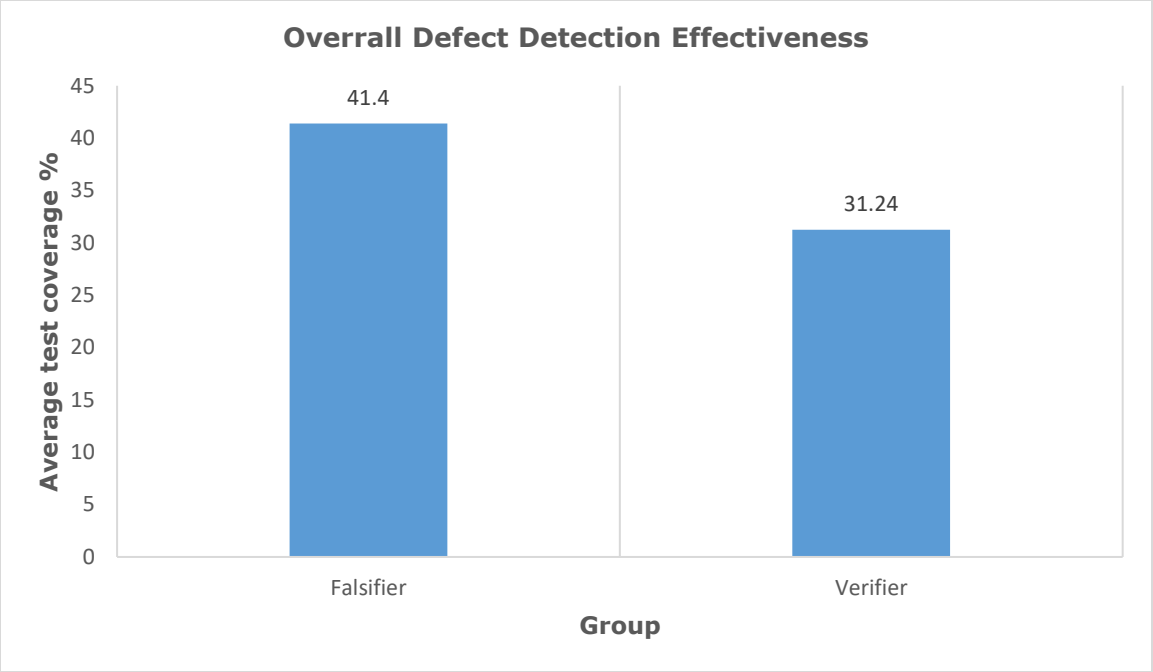

Figure 3. Comparison of overall test coverage by falsifier and verifier groups

4.3. Effect of Experience on Negative test case detection (H3)

In order to evaluate hypothesis 3 to see how experience in software development/testing industry effects negative test detection, we analyzed the number of negative test cases detected by subjects more than 14 months of experience and less than 14 months. The average months of experience among the subjects was 14 months and hence this value was chosen. An independent samples t-test was done to compare the average invalid classes' covered and overall percent coverage obtained by the experienced and less experienced subjects within the falsifier and verifier groups. The results indicated no significant difference between the average invalid classes covered by the different levels of experience in the two groups. In the falsifier category the average invalid classes detected by experienced and less experienced subjects were 8.75 and 9 respectively ($p=0.927$). In the verifier category the average invalid classes detected by experienced and less experienced subjects were 7.25 and 5.50 respectively ($p=0.258$). These results are shown in Figure 4.

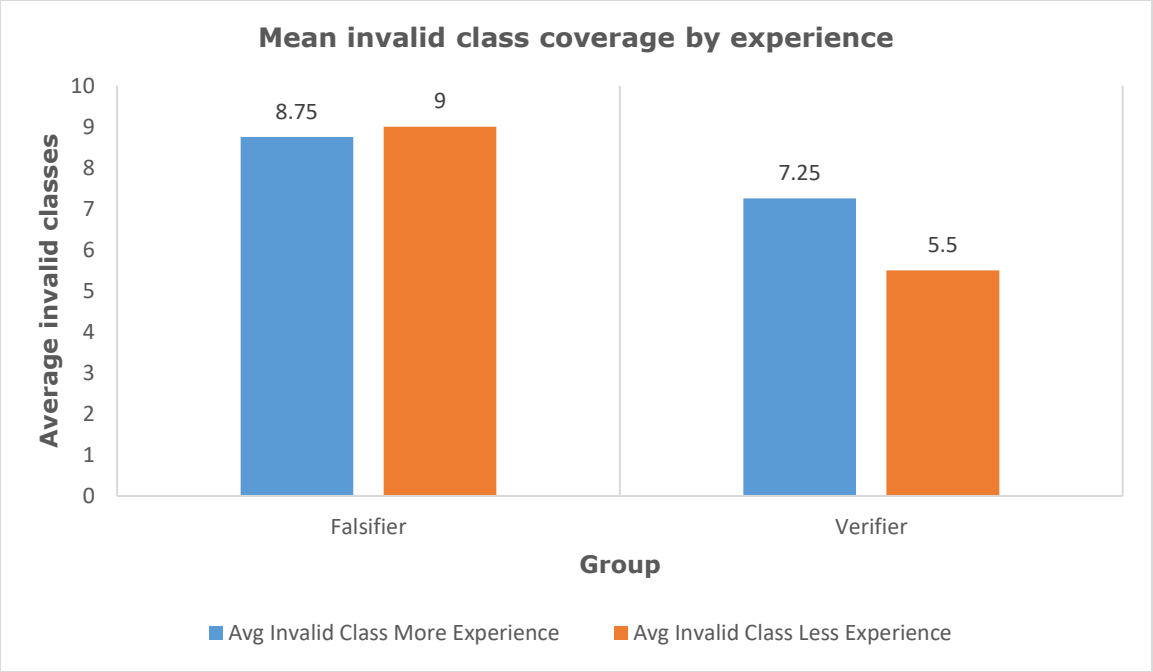

Figure 4. Comparison of invalid classes covered based on experience of subjects

The results also indicated no significant difference between the average overall coverage percentages covered by the different levels of experience in the two groups. In the falsifier category the average overall percent coverage obtained by experienced and less experienced subjects were 41.14% and 41.66% respectively (p=0.912). In the verifier category the average overall percent coverage obtained by experienced and less experienced subjects were 34.89% and 29.42% respectively (p=0.470).This result was consistent with previous studies which implied that experience does not affect the confirmation bias levels among software developers/testers. These results are shown in Figure 5.

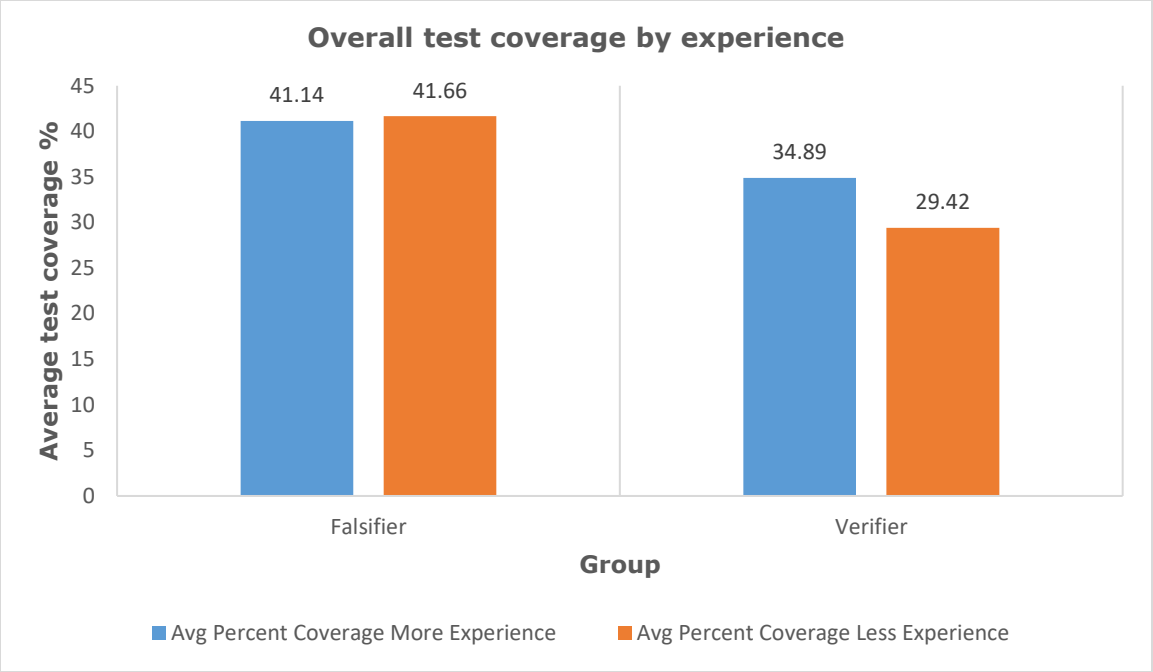

Figure 5. Comparison of overall test coverage based on experience of subjects

5. DISCUSSION OF RESULTS

This section discusses the results from Section 4 by accepting or rejecting the hypothesis and answering the research questions.

Research Question 1: Do software testers who have low confirmation bias levels detect more negative test cases and obtain better coverage of invalid classes during software testing?

Results indicate that the subjects in the group classified as falsifier who have low confirmation bias levels detected significantly more negative test cases and obtained better coverage of the invalid classes than the group classified as verifier having high confirmation bias levels. This result thus supports hypothesis 1:

Hypotheses 1: Software testers with low confirmation bias levels detect more negative test cases and obtain better coverage of invalid classes during software testing.

This implies that more test coverage through negative test cases can be obtained by testers with low confirmation bias levels. This result signifies the importance of doing negative tests and encourages software testers to test scenarios that can potentially break the system or a specific rule in addition to tests that only verify if the system works as expected.

Research Question 2: Does the confirmation bias levels of software testers have an impact on the overall defect detection effectiveness during software testing?

Results indicate that the overall defect detection effectiveness of subjects with low confirmation bias (falsifiers) was significantly better than the subjects with high confirmation bias levels (verifiers). This result thus supports hypothesis 2:

Hypotheses 2: The overall effectiveness of defect detection can be improved with low confirmation bias levels.

This implies that confirmation bias can impact the overall quality of testing and provides empirical evidence which signifies that software testers with low confirmation bias

can be more effective in terms of overall defect detection as compared to software testers with high confirmation bias.

Research Question 3: Do software testers with low confirmation bias and having high work experience in the software development/testing industry detect more invalid classes and obtain better overall test coverage during software testing?

Results indicate that there was no significant difference observed between the average number of invalid classes covered and the overall test coverage percentage detected within the Falsifier and Verifier groups. This result thus does not support hypothesis 3:

Hypotheses 3: Software testers with high experience will be able to detect more invalid classes and obtain better overall test coverage during software testing.

This implies that the experience of software developer/tester does not play a significant role in detecting invalid classes and obtaining better overall test coverage. Previous studies [6, 12, 13] have shown same results and our results further prove that experience do not significantly impact the confirmation bias levels of software testers.

6. CONCLUSION

Based on the results obtained from the study in this paper, we can understand the impact of confirmation bias on the quality of functional software testing. The results provide empirical evidence to suggest that testers who were classified as having low confirmation bias were better testers as compared to testers with high levels of confirmation bias. The study also shows that testers who do effective negative testing are much effective in terms of quality of testing and obtaining good test coverage. The results also indicate that experience factor of testers do not have any impact on the confirmation bias. Hence, we can interpret that exercising negative test cases and testing to see if a system fails should be an integral part of software testing. Software testers should be made more aware and encouraged to exercise tests that break the system in addition to tests that verify if the system works as specified in the requirements. This would result in overall improvement of the testing process and lead to better quality of software products. In future, we can plan to replicate this study with different settings and data sets like capstone projects and in industrial settings with professional testers.

REFERENCES

[1] A. Tversky and D. Kahneman, "Judgment under Uncertainty: Heuristics and Biases", *Science*, vol. 185, no. 4157, pp. 1124-1131, 1974.

[2] W. Stacy and J. MacMillan, "Cognitive bias in software engineering", *Communications of the ACM*, vol. 38, no. 6, pp. 57-63, 1995.

[3] P. Wason, "On the failure to eliminate hypotheses in a conceptual task", *Quarterly Journal of Experimental Psychology*, vol. 12, no. 3, pp. 129-140, 1960.

[4] Demarco, T. and Lister, T., *Peopleware: Productive Projects and Teams.* New York: Dorset House Publishing, 1987.

[5] Y. Chernak, "Validating and improving test-case effectiveness", *IEEE Software*, vol. 18, no. 1, pp. 81-86, 2001.

[6] B. Teasley, L. Leventhal and S. Rohlman, "Positive test bias in software engineering professionals: What is right and what's wrong", in *Proceedings of the 5th workshop on empirical studies of programmers*, 1993.

[7] A. Tversky and D. Kahneman, "Availability: A heuristic for judging frequency and probability", *Cognitive Psychology*, vol. 5, no. 2, pp. 207-232, 1973.

[8] G. Gigerenzer and W. Gaissmaier, "Heuristic Decision Making", *Annual Review of Psychology*, vol. 62, no. 1, pp. 451-482, 2011.

[9] A. Shah and D. Oppenheimer, "Heuristics made easy: An effort-reduction framework.", *Psychological Bulletin*, vol. 134, no. 2, pp. 207-222, 2008.

[10] R. Cialdini, *Influence: Science and Practice*, 5th ed. Boston: Allyn & Bacon, 2009.

[11] G. Calikli and A. Bener, "Empirical analyses of the factors affecting confirmation bias and the effects of confirmation bias on software developer/tester performance", in *PROMISE*

'10 Proceedings of the 6th International Conference on Predictive Models in Software Engineering, Timisoara, Romania, 2010.

[12] G. Calikli, B. Arslan and A. Bener, "Confirmation Bias in Software Development and Testing: An Analysis of the Effects of Company Size, Experience and Reasoning Skills", *22nd Annual Psychology of Programming Interest Group Workshop*, 2010.

[13] G. Calikli, A. Bener and B. Arslan, "An analysis of the effects of company culture, education and experience on confirmation bias levels of software developers and testers", *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10*, 2010.

[14] G. Calikli and A. Bener, "Influence of confirmation biases of developers on software quality: an empirical study", *Software Quality Journal*, vol. 21, no. 2, pp. 377-416, 2012.

[15] J. Parsons and C. Saunders, "Cognitive heuristics in software engineering applying and extending anchoring and adjustment to artifact reuse", *IEEE Transactions on Software Engineering*, vol. 30, no. 12, pp. 873-888, 2004.

[16] C. Mair and M. Shepperd, "Human judgement and software metrics", *Proceeding of the 2nd international workshop on Emerging trends in software metrics - WETSoM '11*, 2011.

[17] M. Jorgensen, "Forecasting of software development work effort: Evidence on expert judgement and formal models", *International Journal of Forecasting*, vol. 23, no. 3, pp. 449-462, 2007.

[18] M. Jorgensen, "Identification of more risks can lead to increased over-optimism of and over-confidence in software development effort estimates", *Information and Software Technology*, vol. 52, no. 5, pp. 506-516, 2010.

[19] D. Griffin, R. Gonzalez and C. Varey, "The Heuristics and Biases Approach to Judgment Under Uncertainty", *Blackwell Handbook of Social Psychology: Intraindividual Processes*, pp. 207-235, 2007.

[20] S. Reich and P. Ruth, "Wason's selection task: Verification, falsification and matching", *British Journal of Psychology*, vol. 73, no. 3, pp. 395-405, 1982.

[21] G. Myers, T. Badgett, T. Thomas and C. Sandler, *The art of software testing*, 1st ed. Hoboken, N.J.: John Wiley & Sons, 2004.